



PALASM[®] 4 Reference Guide

1992

Advanced
Micro
Devices



PALASM 4 release 1.5

PALASM® 4 USER'S MANUAL

VOLUME 2 - PALASM 4 REFERENCE GUIDE

© 1991 Advanced Micro Devices, Inc.
901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088

TEL: 408-732-2400
TWX: 910339-9280
TELEX: 34-6306
TOLL FREE: 800-538-8450

APPLICATIONS HOTLINE: 800-222-9323

Advanced Micro Devices reserves the right to make changes in specifications at any time and without notice. The information furnished by Advanced Micro Devices is believed to be accurate and reliable. However, no responsibility is assumed by Advanced Micro Devices for its use, nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Advanced Micro Devices.

IBM® is a registered trademark of International Business Machines Corporation.

ABEL™ is a trademark of Data I/O Corporation.

IBM PC™, PC AT™, PC AT™, and PS/2™ are trademarks of International Business Machines Corporation.

MS-DOS™ is a trademark of Microsoft Corporation.

PAL® and PALASM® are registered trademarks and MACH™ is a trademark of Advanced Micro Devices, Inc.

OrCAD® and OrCAD SDT/III® are registered trademarks of OrCAD.

PALASM 4 USER'S MANUAL

VOLUME 1 - PALASM 4 Getting Started and MACH Workbook

Preface

Acknowledgements

Section I: Getting Started

Chapter 1: Installation Guide

Chapter 2: Design Entry Demonstration

Chapter 3: Schematic-Based MACH Design Demonstration

Section II: Designer's Guide

Chapter 4: Entry

Chapter 5: Compilation / Fitting

Chapter 6: Simulation

MACH Design Workbook

VOLUME 2 - PALASM 4 Reference Guide

Section III: Library Reference

Chapter 7: Introduction

Chapter 8: Macro and Schematic Datasheets

DISCONTINUED

Section IV: Software Reference

Chapter 9: Menus and Commands

Chapter 10: Language Reference

Chapter 11: Device Programming Reference

Section V: Appendices

Appendix A: PLD Text Editor

Section VI: Glossary / Index

Glossary

Index

TABLE OF CONTENTS

VOLUME 1 - Getting Started and MACH Workbook

SECTION I: GETTING STARTED

CHAPTER 1: INSTALLATION GUIDE

1.1	REQUIREMENTS	1-2
1.1.1	HARDWARE.....	1-2
1.1.2	SOFTWARE	1-3
1.2	STEPS	1-4
1.2.1	INSTALLATION.....	1-4
1.2.2	CONFIGURATION.....	1-11
1.3	FILE UPDATES	1-12
1.3.1	AUTOEXEC.....	1-12
1.3.2	CONFIG.SYS	1-12

CHAPTER 2: DESIGN ENTRY

2.1	CREATING A TEXT-BASED DESIGN	2-3
2.1.1	BEGIN THE DESIGN	2-5
2.1.2	FILL IN THE PDS DECLARATION-SEGMENT FORM.....	2-7
2.1.3	COMPLETE THE PDS FILE USING A TEXT EDITOR.....	2-14
2.2	SCHEMATIC-BASED DESIGN ENTRY	2-17
2.2.1	BEGIN THE DESIGN	2-19
2.2.2	FILL IN THE SCHEMATIC CONTROL-FILE FORM.....	2-22
2.2.3	CREATE THE SCHEMATIC.....	2-24
2.3	MERGING MULTIPLE PDS FILES.....	2-30
2.3.1	SET UP.....	2-31
2.3.2	RETRIEVE FILES.....	2-33
2.3.3	RESOLVE CONFLICTS	2-36
2.3.4	MERGE FILES	2-44
2.3.5	RECOMPILE THE COMBINED DESIGN	2-48

CHAPTER 3: SCHEMATIC-BASED MACH DESIGN DEMONSTRATION

3.1	DESIGN EXAMPLE	3-2
3.1.1	ADDRESS COUNTER	3-4
3.1.2	WORD COUNTER	3-6
3.2	CONFIRM THE SETUP	3-8
3.2.1	RETRIEVE THE DESIGN	3-9
3.2.2	VERIFY SETUP	3-11
3.3	VIEW DESIGN FILES	3-14
3.3.1	VIEW THE SCHEMATIC	3-14
3.3.2	VIEW THE CONTROL FILE	3-19
3.4	COMPLETE THE DESIGN	3-21
3.5	SIMULATE THE DESIGN	3-33

SECTION II: DESIGNER'S GUIDE

CHAPTER 4: ENTRY

4.1	OVERVIEW	4-2
4.2	DESIGN FLOW	4-3
4.2.1	TEXT ENTRY	4-4
4.2.2	SCHEMATIC ENTRY	4-4
4.2.3	COMBINED ENTRY METHODS	4-6
4.3	BOOLEAN DESIGN STRATEGIES	4-7
4.3.1	OUTPUT POLARITY	4-7
4.3.1.1	The Two Components of Polarity	4-7
4.3.1.2	Controlling Polarity from the Equation	4-8
4.3.1.3	Controlling Polarity from the Pin or Node Statement	4-9
4.3.1.4	Creating Equivalent Logic	4-10
4.3.2	CONTROLLING OUTPUT BUFFERS USING .TRST	4-11
4.3.2.1	Bank Output Enable	4-11
4.3.2.2	Individual Output Enable	4-12
4.3.2.3	Grouped Output Enable	4-14
4.3.3	CONTROLLING CLOCKS WITH .CLKF	4-16
4.3.4	CONTROLLING SET/RESET USING .SETF AND .RSTF	4-17
4.3.4.1	Banking Set and Reset in MACH Devices	4-17

4.3.5	USING HIGH-LEVEL CONSTRUCTS.....	4-18
4.3.5.1	Vector Notation.....	4-18
4.3.5.2	Radix Notation.....	4-20
4.3.5.3	IF-THEN-ELSE Statement.....	4-21
4.3.5.4	CASE Statement.....	4-22
4.3.6	CONTROLLING LOGIC REDUCTION.....	4-25
4.4	STATE-MACHINE DESIGN STRATEGIES.....	4-26
4.4.1	STATE SEGMENT OVERVIEW.....	4-28
4.4.2	DEFINING MOORE AND MEALY MACHINES.....	4-29
4.4.3	CREATING STATE-MACHINE EQUATIONS.....	4-30
4.4.3.1	Condition Equations.....	4-31
4.4.3.2	Transition Equations.....	4-31
4.4.3.3	Output Equations.....	4-32
4.4.3.4	State-Machine Example.....	4-32
4.4.4	DEFAULT BRANCHES.....	4-34
4.4.4.1	Global Defaults.....	4-35
4.4.4.2	Local Defaults.....	4-35
4.4.4.3	Example With Default Branches.....	4-36
4.4.5	ASSIGNING STATE BITS.....	4-38
4.4.5.1	Automatic State-Bit Assignment.....	4-38
4.4.5.2	Manual State-Bit Assignment.....	4-39
4.4.5.3	Choosing State-Bit Assignments.....	4-39
4.4.5.4	Example Using Manual State-Bit Assignment.....	4-41
4.4.6	USING STATE BITS AS OUTPUTS.....	4-42
4.4.7	INITIALIZING A STATE MACHINE.....	4-43
4.4.8	CLOCKING A STATE MACHINE.....	4-44
4.4.8.1	Example Using State Bits as Outputs, Power-Up and Clock Equations.....	4-45
4.5	SCHEMATIC VERSUS TEXT ENTRY.....	4-47
4.5.1	LIBRARY ANALYSIS.....	4-48
4.5.2	SCHEMATIC PARAMETERS.....	4-49
4.5.2.1	Fixing Pin Locations.....	4-50
4.5.2.2	Fixing Node Locations.....	4-51
4.5.2.3	Assigning Logic to a Block.....	4-53
4.5.2.4	Controlling Minimization.....	4-54
4.5.2.5	Controlling Set/Reset.....	4-56
4.5.2.6	Deleting Unused Logic.....	4-59

4.5.3	DESIGN DOCUMENTATION	4-60
4.5.4	CONVERSION, EXISTING SCHEMATICS TO MACH-DEVICE DESIGNS	4-62
4.6.	COMBINING SCHEMATIC AND TEXT DESCRIPTIONS	4-64
4.7	MERGING MULTIPLE PDS FILES	4-66
4.7.1	INPUT FILES.....	4-66
4.7.2	DESIGN EVALUATION.....	4-67
4.7.2.1	Compatibility.....	4-67
4.7.2.2	Inputs, Clock Signals, and Set/Reset Control.....	4-69
4.7.3	GUIDELINES.....	4-69
4.7.3.1	Set up	4-70
4.7.3.2	Retrieve Files	4-72
4.7.3.3	Resolve Conflicts	4-74
4.7.3.4	Merge Files	4-81
4.7.3.5	Re-engineer the Combined Design	4-84

CHAPTER 5: COMPILATION / FITTING

5.1	OVERVIEW.....	5-2
5.2.	THE FITTING PROCESS	5-3
5.2.1	INITIALIZATION	5-3
5.2.2	BLOCK PARTITIONING.....	5-3
5.2.3	RESOURCE ASSIGNMENT.....	5-4
5.3	DESIGNING TO FIT	5-5
5.3.1	METHODOLOGY	5-6
5.3.2	ANALYZE DEVICE RESOURCES	5-7
5.3.2.1	Clock Signals	5-8
5.3.2.2	Set/Reset Signals	5-8
5.3.2.3	Macrocells and I/O Pins.....	5-9
5.3.2.4	Product Terms	5-9
5.3.2.5	Interconnection Resources.....	5-10
5.3.3	ASSIGNING PIN AND NODE LOCATIONS.....	5-11
5.3.3.1	Large Logic Functions	5-13
5.3.3.2	Large Functions at the End of a Block.....	5-14
5.3.3.3	Adjacent Macrocell Use	5-14
5.3.4	GROUPING LOGIC.....	5-15

5.3.5	SETTING COMPILATION AND FITTING OPTIONS	5-16
5.3.5.1	Gate Splitting	5-16
5.3.5.2	Maximize Packing of Logic Blocks.....	5-18
5.3.5.3	Expand Small PT Spacing	5-19
5.3.5.4	Expand All PT Spacing	5-19
5.4	STRATEGIES, DESIGNS THAT DON'T FIT	5-21
5.4.1	MACH REPORT	5-21
5.4.1.1	Flags Used.....	5-22
5.4.1.2	Pair Analysis.....	5-23
5.4.1.3	Pre-Placement & Equation Usage Checks	5-23
5.4.1.4	Timing Analysis for Signals	5-23
5.4.1.5	Device Resource Checks	5-24
5.4.1.6	Block Partitioning	5-25
5.4.1.7	Utilization	5-27
5.4.1.8	Assigning Resources	5-28
5.4.1.9	Signals, Tabular.....	5-29
5.4.1.10	Signals, Equations	5-30
5.4.1.11	Feedback Map	5-32
5.4.1.12	Logic Map	5-33
5.4.1.13	Pin Map	5-34
5.4.1.14	Connection Status	5-35
5.4.1.15	Output Files, Errors and Warnings	5-35
5.4.2	INTERPRETING ERROR MESSAGES.....	5-36
5.4.2.1	Marginal Block Partitioning Measure, Warning F120.....	5-37
5.4.2.2	Partitioning Could Not Place All Signals into Blocks, Error F580.....	5-39
5.4.2.3	Product Term Distribution, Error 610	5-41
5.4.2.4	Not All Input Signals Were Connected, Error F600	5-47
5.4.2.5	Connection Problem (Wiring Congested), Error F590	5-51
5.4.2.6	Mapping Difficulty – No Feasible Solution, Error F620.....	5-54
5.4.2.7	Procedures For Reducing Logic Complexity	5-55
5.4.3	USING DIFFERENT FITTING OPTIONS	5-56
5.4.4	CHOOSING A LARGER MACH DEVICE.....	5-57
5.5	CHANGES AFTER SUCCESSFUL FITTING	5-58
5.5.1	CHANGING THE PIN OUT.....	5-58
5.5.2	CHANGING LOGIC.....	5-59

CHAPTER 6: SIMULATION

6.1	OVERVIEW.....	6-2
6.2	CREATING A SIMULATION FILE	6-3
6.2.1	SIMULATION COMMAND SUMMARY	6-3
6.2.2	SIMULATION SEGMENT VS. AUXILIARY FILE	6-5
6.2.3	CONSIDERATIONS	6-7
6.2.3.1	Flip-Flops	6-8
6.2.3.2	Internal Nodes	6-8
6.2.3.3	Latches	6-9
6.2.3.4	Output Enable.....	6-9
6.2.3.5	Preloaded Registers	6-9
6.2.3.6	Verified Signal Values	6-10
6.3	VIEWING SIMULATION RESULTS.....	6-11
6.3.1	HISTORY FILE	6-11
6.3.2	TRACE FILE.....	6-12
6.4	USING SIMULATION CONSTRUCTS.....	6-15
6.4.1	FOR LOOP	6-15
6.4.2	WHILE LOOP	6-15
6.4.3	IF-THEN-ELSE	6-16
6.5	DESIGN EXAMPLES.....	6-17
6.5.1	BOOLEAN EQUATION DESIGN	6-17
6.5.2	STATE-MACHINE DESIGN	6-20

MACH Design Workbook

PREFACE

1	8-BIT BARREL SHIFT REGISTER: A DESIGN WITH NUMEROUS INTERCONNECTS	1-1
2	TWO 8-BIT COUNTERS WITH A MUX: A DESIGN WITH HIGH DEVICE-RESOURCE REQUIREMENTS	2-1
3	UNIVERSAL ASYNCHRONOUS RECEIVER / TRANSMITTER: A DESIGN WITH SIGNAL GROUPING REQUIREMENTS.....	3-1
INDEX		I-1

VOLUME 2 - Reference Guide

SECTION III: LIBRARY REFERENCE

CHAPTER 7: INTRODUCTION

7.1	MACH LIBRARY OVERVIEW.....	7-2
7.1.1	BUFFER MACROS.....	7-2
7.1.2	COMBINATORIAL MACROS	7-4
7.1.3	SINGLE-PIN MACROS.....	7-4
7.1.4	STORAGE MACROS	7-6
7.1.5	TTL-EQUIVALENT MACROS.....	7-8
7.2	LIBRARY CONSIDERATIONS	7-9
7.2.1	DESIGN EXAMPLES	7-9
7.2.1.1	Example 1.....	7-10
7.2.1.2	Example 2.....	7-12
7.2.1.3	Example 3.....	7-14
7.2.2	LATCH MACROS	7-15
7.2.3	MINIMIZATION	7-15
7.2.4	REFERENCE DESIGNATORS	7-15
7.2.5	SIGNAL NAMES.....	7-16
7.3	ANNOTATED DATASHEET	7-17

CHAPTER 8: MACRO AND SCHEMATIC DATASHEETS

QUICK REFERENCE.....	8-1
DATASHEETS	8-18

SECTION IV: SOFTWARE REFERENCE

CHAPTER 9: MENUS AND COMMANDS

9.1	OVERVIEW.....	9-2
9.1.1	FEATURES	9-3
9.1.2	CONVENTIONS	9-5
9.2	COMMANDS AND OPTIONS.....	9-7
9.2.1	FILE MENU.....	9-8
9.2.1.1	Begin New Design	9-8
9.2.1.2	Retrieve an Existing Design.....	9-15

9.2.1.3	Merge Design Files.....	9-16
9.2.1.4	Change Directory.....	9-32
9.2.1.5	Delete Specified Files.....	9-33
9.2.1.6	Set Up.....	9-34
9.2.1.7	Go To System.....	9-48
9.2.1.8	Quit.....	9-48
9.2.2	EDIT MENU.....	9-49
9.2.2.1	Text File.....	9-49
9.2.2.2	Schematic File.....	9-50
9.2.2.3	Control File for Schematic Design.....	9-50
9.2.2.4	Auxiliary Simulation File.....	9-51
9.2.2.5	Other File.....	9-51
9.2.3	RUN MENU.....	9-52
9.2.3.1	Compile.....	9-53
9.2.3.2	Simulation.....	9-54
9.2.3.3	Both.....	9-55
9.2.3.4	Other Operations.....	9-55
9.2.4	VIEW MENU.....	9-60
9.2.4.1	Execution Log File.....	9-61
9.2.4.2	Design File.....	9-61
9.2.4.3	Reports.....	9-62
9.2.4.4	JEDEC Data.....	9-63
9.2.4.5	Simulation Data.....	9-64
9.2.4.6	Waveform Display.....	9-65
9.2.4.7	Current Disassembled File.....	9-66
9.2.4.8	Pinout.....	9-67
9.2.4.9	Netlist Report.....	9-67
9.2.4.10	Other File.....	9-68
9.2.5	DOWNLOAD MENU.....	9-68
9.2.6	DOCUMENTATION MENU.....	9-69
9.2.6.1	Index of Topics.....	9-69
9.2.6.2	Language Reference.....	9-70
9.2.6.3	Help On Errors.....	9-71
9.2.7	[F1] FOR HELP.....	9-72

CHAPTER 10: LANGUAGE REFERENCE

OVERVIEW.....	10-2
BOOLEAN-EQUATION ELEMENTS.....	10-4
STATE-MACHINE CONSTRUCTS.....	10-5

SPECIFYING OUTPUTS IN IF-THEN-ELSE AND CASE STATEMENTS	10-6
SYNTAX AND EXAMPLES	10-12
ASSIGNMENT OPERATOR	10-14
AUTHOR	10-18
BOOLEAN EQUATION	10-20
CASE	10-24
CHECK	10-30
CHECKQ	10-34
CHIP	10-38
.CLKF	10-40
CLKF	10-44
CLOCKF	10-46
.CMBF	10-48
COMBINATORIAL	10-50
COMMENT	10-52
COMPANY	10-54
CONDITIONS	10-56
DATE	10-60
DECLARATION SEGMENT	10-62
DEFAULT_BRANCH	10-66
DEFAULT_OUTPUT	10-70
EQUATIONS SEGMENT	10-72
EXPRESSION	10-74
FLOATING PINS AND NODES	10-78
FOR-TO-DO	10-82
FUNCTIONAL EQUATIONS	10-86
GND	10-90
GROUP	10-92
IF-THEN-ELSE, EQUATIONS	10-96
IF-THEN-ELSE, SIMULATION	10-100
.J EQUATION	10-102
.K EQUATION	10-104
LATCHED	10-106
LOCAL DEFAULT	10-108
MACH_SEG_A	10-110
MASTER_RESET	10-114
MEALY_MACHINE	10-116
MINIMIZE_OFF	10-118
MOORE_MACHINE	10-120

NODE.....	10-122
OPERATOR.....	10-126
.OUTF.....	10-128
OUTPUT_ENABLE.....	10-132
OUTPUT_HOLD.....	10-134
PAIR.....	10-136
PATTERN.....	10-140
PIN.....	10-142
PRELOAD.....	10-146
.PRLD.....	10-148
PRLDF.....	10-150
.R EQUATION.....	10-152
REGISTERED.....	10-154
REVISION.....	10-156
.RSTF.....	10-158
.S EQUATION.....	10-160
.SETF.....	10-162
SETF.....	10-164
SIGNATURE.....	10-166
SIMULATION.....	10-170
START_UP.....	10-172
STATE.....	10-176
STATE ASSIGNMENT EQUATION.....	10-180
STATE EQUATIONS.....	10-184
STATE OUTPUT EQUATION.....	10-188
STATE TRANSITION EQUATION.....	10-192
STRING.....	10-194
TEST.....	10-198
.T EQUATION.....	10-202
.T1 EQUATION.....	10-204
.T2 EQUATION.....	10-206
TITLE.....	10-208
TRACE_OFF.....	10-210
TRACE_ON.....	10-212
.TRST.....	10-214
VCC.....	10-218
VECTOR.....	10-220
WHILE-DO.....	10-222

CHAPTER 11: DEVICE PROGRAMMING REFERENCE

11.1	PLD INTRODUCTION.....	11-2
11.1.1	PLD NAMING CONVENTIONS.....	11-2
11.1.2	STANDARD PLD DEVICES VERSUS NON-STANDARD PLD DEVICES.....	11-3
11.2	DEVICE FEATURE CROSS-REFERENCE.....	11-4
11.3	GENERAL PLD LANGUAGE SYNTAX	11-6
11.3.1	OUTPUT-ENABLE CONTROL.....	11-7
11.3.1.1	Common External Output-Enable Pin	11-7
11.3.1.2	Individual Product Term Control.....	11-7
11.3.1.3	Common External Pin or Individual Product Term Control.....	11-8
11.3.2	CLOCK CONTROL.....	11-10
11.3.2.1	Common External Clock Control	11-10
11.3.2.2	Individual Product Term Clock Control.....	11-11
11.3.3	PRESET CONTROL.....	11-11
11.3.3.1	Individual Product Term Control.....	11-12
11.3.3.2	Global Product Term Control.....	11-12
11.3.4	RESET CONTROL	11-13
11.3.4.1	Individual Product Term Control.....	11-13
11.3.4.2	Global Product Term Control.....	11-14
11.3.5	DEVICE POLARITY	11-15
11.3.5.1	Active-Low Polarity	11-15
11.3.5.2	Active-High Polarity	11-16
11.3.5.3	Programmable Polarity.....	11-18
11.3.6	COMBINATORIAL LOGIC	11-19
11.3.7	REGISTERED OR LATCHED LOGIC.....	11-20
11.3.7.1	D Flip-Flop	11-20
11.3.7.2	SR Flip-Flop.....	11-21
11.3.7.3	Latch.....	11-22
11.3.8	FEEDBACK.....	11-23
11.3.8.1	Programmable Feedback	11-23
11.3.8.2	Non-Programmable Feedback.....	11-32
11.3.9	PRELOAD CONTROL.....	11-33
11.3.9.1	Supervoltage	11-33
11.3.9.2	Product Term Control.....	11-33
11.3.10	OBSERVABILITY PRODUCT TERM CONTROL.....	11-34
11.3.11	COMPLEMENT ARRAY.....	11-35
11.3.12	ELECTRONIC SIGNATURE.....	11-36
11.4	PLD DEVICE SYNTAX DATASHEETS	11-38

11.4.1	PIN AND NODE DESCRIPTIONS.....	11-38
11.4.2	BLOCK AND MACROCELL DIAGRAM(S).....	11-39
11.4.3	SPECIAL PROGRAMMING FEATURES	11-39
	105	11-40
	167/168.....	11-44
	16RA8.....	11-48
	16V8HD	11-52
	20EG8	11-62
	20EV8.....	11-66
	20RA10.....	11-70
	22IP6.....	11-74
	22V10	11-80
	23S8	11-84
	26V12	11-92
	29M16.....	11-96
	29MA16	11-104
	30S16	11-110
	32VX10.....	11-130
	610	11-142
11.5	MACH 1 AND MACH 2 SERIES DEVICES	11-156
11.5.1	OVERVIEW	11-156
	11.5.1.1 Device Features.....	11-156
	11.5.1.2 Pin and Node Descriptions	11-157
	11.5.1.3 PALASM Programming Features	11-157
11.5.2	SAMPLE EQUATIONS	11-168
	11.5.2.1 I/O Cell and Macrocell	11-169
	11.5.2.2 Pin and Node Feedback	11-170
	11.5.2.3 Registered and Latched Inputs.....	11-171
	MACH 110 Device.....	11-174
	MACH 120 Device.....	11-178
	MACH 130 Device.....	11-182
	MACH 210 Device.....	11-186
	MACH 220 Device.....	11-190
	MACH 230 Device.....	11-194
	MACH 215 Device.....	11-198

SECTION V: APPENDICES

APPENDIX A: PLD TEXT EDITOR

PLD TEXT EDITOR.....	A-1
A.1 FILE MENU.....	A-2
A.2 WINDOW MENU.....	A-3
A.3 BLOCK MENU.....	A-4
A.4 SEARCH MENU.....	A-5
A.5 PRINT MENU.....	A-6
A.6 MACRO MENU.....	A-7
A.7 EDITING MENU.....	A-8
A.8 OTHER MENU.....	A-9
A.9 QUIT MENU.....	A-11

SECTION VI: GLOSSARY / INDEX

GLOSSARY

INDEX

SECTION III

LIBRARY REFERENCE

Chapter 7: Introduction

Chapter 8: Macro and Schematic Datasheets

CHAPTER 7

INTRODUCTION

CONTENTS

INTRODUCTION	1
7.1 MACH LIBRARY OVERVIEW	2
7.2 ANNOTATED SCHEMATIC	3
7.3 CAPTURING A SCHEMATIC	6
7.3.1 SPECIFYING PIN AND NODE NUMBERS	6
7.3.2 GROUPING SIGNALS INTO A BLOCK	7
7.3.3 TURNING MINIMIZATION OFF.....	7
7.3.4 MANUALLY SPLITTING PRODUCT TERMS	8
7.3.5 TERMINATING UNUSED INPUTS AND OUTPUTS.....	8
7.3.6 DEFINING PRESET AND RESET FUNCTIONS.....	9
7.3.7 INTERPRETING REFERENCE DESIGNATORS.....	10
7.3.8 NAMING SIGNALS.....	10
7.4 ANNOTATED DATASHEET	11

This chapter introduces the MACH library¹ and its functional elements, called macros, in four discussions.

- The overview, 7.1, introduces the categories of macros contained within the AMD-supplied library.
- The annotated schematic design discussion, 7.2, illustrates considerations for capturing a schematic and points to the corresponding descriptions in discussion 7.3.
- The capturing a schematic discussion, 7.3, describes how to use the library macros effectively to create a design using OrCAD/SDT III.
- The annotated datasheet, 7.4, provides a sample datasheet from Chapter 8, in this section, with a description of each information field.

¹ Only the MACH family of devices is supported in the library. The library must be purchased separately from the PALASM 4 software.

7.1 MACH LIBRARY OVERVIEW

AMD provides over 100 commonly used logic functions as macros in the MACH library. A summary of the available macros is shown next.²

BUFFER MACROS	COMBINATORIAL MACROS	OTHER MACROS	STORAGE MACROS	TTL-EQUIVALENT MACROS
BUF INV NTRST TRST	AND NAND NOR OR XNOR XOR	AINIT NC NODE PDWN PUP	FD FT LD LDX DFF DLAT DLATX TFF	See Chapter 8

² Refer to Chapter 8, in this section, for a complete list of macros, symbols, and datasheets.

7.2 ANNOTATED SCHEMATIC

This discussion presents an annotated schematic to illustrate the features and considerations of capturing a MACH schematic design. Each lettered paragraph below corresponds to a circled letter in the next figure.

- A. You use the AINIT macro to specify the asynchronous preset and reset for all three terminal storage macros in a design.

In this case, the T-type flip-flops in the 74163 are set and reset by the signals ASYNC_SET and ASYNC_RST. Use five-terminal storage macros to specify multiple asynchronous preset and reset functions.

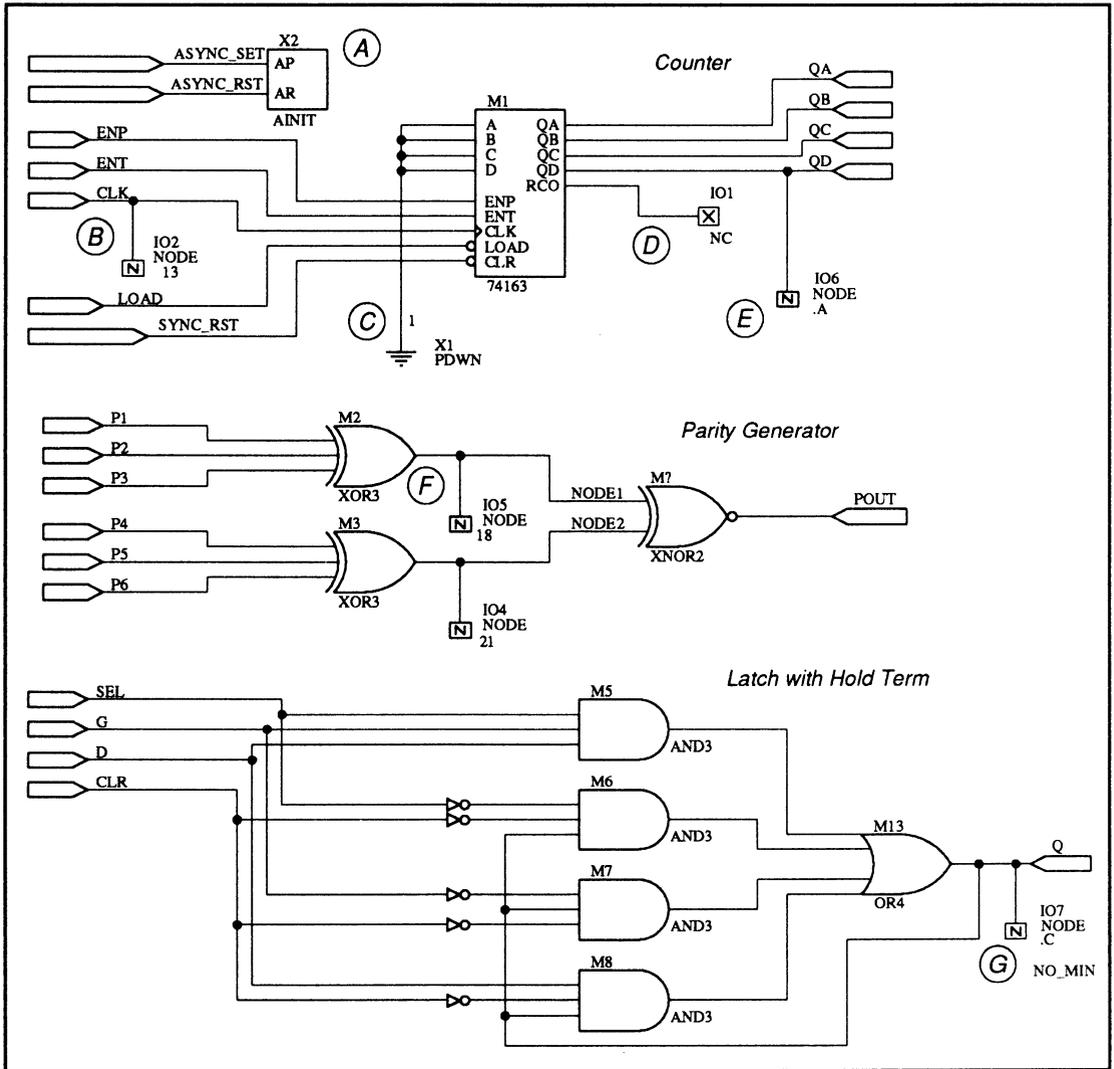
- B. You use the NODE macro on a wire connected to a module port to specify a package pin number. In this case, the clock is connected to pin 13.
- C. You use the PDWN and PUP macros to disable unused TTL-equivalent macro inputs, preventing the unused input signals from being connected to the package pins due to lower-level module ports.

Use the PDWN macro to disable an active-high input and the PUP macro to disable an active-low input.

- D. You use the NC macro to terminate unused TTL-equivalent macro outputs. Any unused logic associated with the designated output is then removed during later processing.
- E. You use the NODE macro to assign logic to specific blocks in the MACH device. In this case, the equation driving QD is grouped into block A.

- F. You use the NODE macro to specify the node number of a buried node signal. In this case, the signal NODE1 is placed at location 18.³
- G. You use the NODE macro to prevent minimization of logic. In this case, inhibiting minimization preserves the redundant hold term of the latch design.

³ Refer to Section IV, Chapter 11, for details on node numbers for each MACH device.



Annotated Schematic

7.3 CAPTURING A SCHEMATIC

You capture a MACH schematic using OrCAD/SDT III and the optional MACH macro library. PALASM 4 provides an interface to the OrCAD/SDT III software.⁴

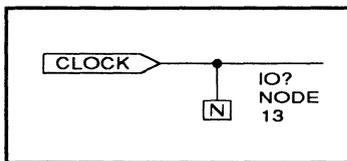
The features and considerations for capturing a MACH schematic design are presented in the following discussions.

- 7.3.1, Specifying Pin and Node Numbers
- 7.3.2, Grouping Signals into a Block
- 7.3.3, Turning Minimization Off
- 7.3.4, Manually Splitting Product Terms
- 7.3.5, Terminating Unused Inputs and Outputs
- 7.3.6, Defining Preset and Reset Functions
- 7.3.7, Interpreting Reference Designators
- 7.3.8, Naming Signals

7.3.1 SPECIFYING PIN AND NODE NUMBERS

I/O pins in a MACH design are automatically defined by unresolved module ports, ones that do not have a complementary connection. All module ports in the design above are unresolved.

Note: A module port is automatically linked to an I/O pin when it is unresolved.



To specify an I/O pin or a node location, you connect a NODE macro to the corresponding signal wire and edit part field 1 to indicate the desired number. You enter only the pin or node number; no alpha characters are allowed.

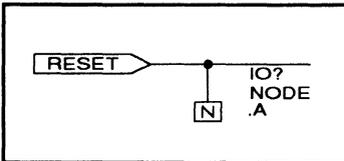
⁴ Refer to Section I, Chapter 1, for details on installing the interface to OrCAD/SDT III.

Important: Assign fixed I/O package pin locations only when necessary. Letting pins float allows more flexibility during the fitting process and results in a higher probability of success.

Also: You must connect the NODE macro to an individual signal wire; connections to bus wires are ignored.

7.3.2 GROUPING SIGNALS INTO A BLOCK

To group the logic associated with a signal into a specific block, you can connect a NODE macro to the corresponding signal wire and edit part field 2. The block letter must be preceded by a period, for example, .A or .B.



Important: You must use the NODE macro to assign attributes to buffer, combinatorial, and module port nets. Storage and TTL-equivalent macros have inherent attribute fields.

7.3.3 TURNING MINIMIZATION OFF

During compilation, logic is replaced with its DeMorgan equivalent if the latter implementation requires fewer product terms.

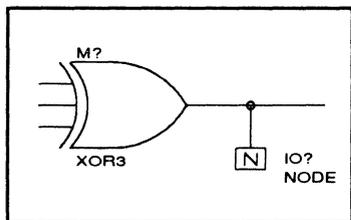
Important: When DeMorganized equivalents are substituted for clocked macros, polarity inversion occurs after the storage element. As a result, the operation of the set/reset logic is reversed when viewed at the pin.

Inhibiting minimization also preserves the redundant hold used in latch designs to prevent timing glitches.

To prevent minimization for the logic associated with a signal, you connect a NODE macro to the corresponding signal wire and edit the name of part field 3 to NO_MIN.

7.3.4 MANUALLY SPLITTING PRODUCT TERMS

The 6-input parity generator, shown in the annotated schematic discussion, is an example of a circuit that requires gate splitting. To implement the complete function requires 32 product terms. However, a MACH 110 device can support only 12 product terms for a single macrocell.⁵

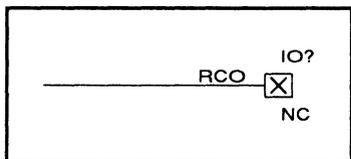


You can split the product terms of this function by placing a NODE macro at the output of each XOR3, thereby creating two buried nodes. The buried node results are fed back into the array to produce the result POUT. Each of the smaller XOR functions fits into a single macrocell; the final implementation of the parity generator fits into a MACH 110 device.

Important: Splitting product terms requires feedback through the array that introduces an additional unit of delay in the signal path. You must account for this added delay.

7.3.5 TERMINATING UNUSED INPUTS AND OUTPUTS

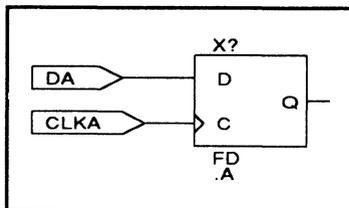
You must disable unused TTL-equivalent macro inputs by connecting them to supply or ground. This prevents the unused input signals from being connected to package pins due to lower-level module ports.



Similarly, you must terminate unused TTL-equivalent macro outputs with an NC macro. Any unused logic associated with the designated output is then removed during later processing.

⁵ Refer to Section IV, Chapter 11, for details on steering product terms.

7.3.6 DEFINING PRESET AND RESET FUNCTIONS



The MACH library contains storage macros with either implied or explicit asynchronous set and reset functions. You specify these functions to define when asynchronous set or reset signals are generated after the device powers up. All flip-flops and latches are automatically reset when the device powers up, independent of the asynchronous control logic.

The AINIT macro represents the implied asynchronous set and reset functions for all of the three-terminal storage macros in a design. The following storage macros require the AINIT macro for asynchronous set and reset definition.

- FD
- FT
- LD
- LDX

Note: Only one AINIT macro can be specified per design. Any additional reset functions must be defined explicitly.

The following storage macros have five terminals, thereby allowing you to explicitly connect the appropriate asynchronous set and reset function.

- DFF
- DLAT
- DLATX
- TFF

Important: Only one asynchronous set and reset signal can be implemented in a single block within the MACH device. Each time you specify a unique set or reset function, the new logic is placed in a separate block.

7.3.7 INTERPRETING REFERENCE DESIGNATORS

Each instance of a macro in a schematic must have a unique reference designator. Reference designators are automatically assigned during the compilation process.

Note: Each reference designator must be unique across all levels of hierarchy, including the subsheets of MACH macros.

The following types of reference designators are used in the AMD-supplied macro libraries.

- **M?** appears with combinatorial, TTL-equivalent, and non-three-state buffer macros.
- **X?** appears with storage and three-state macros.
- **IO?** appears with the NODE and NC macros.

Tip: Error and warning messages include the reference designator and net name of the flagged logic; therefore, it is a good idea to name all nets in the design.

7.3.8 NAMING SIGNALS

Use an alphanumeric string no longer than nine characters for each signal name. Signal names are not case sensitive.

Important: A forward slash, /, does **not** affect the polarity in a schematic signal name.

Note: When a schematic-based design is compiled, a <design>.PDS file is generated. The signal names in this file may have prefixes or suffixes to guarantee uniqueness in the design hierarchy. For example, a forward slash is converted to a _FS_ prefix.

7.4 ANNOTATED DATASHEET

This discussion presents an annotated datasheet to illustrate the layout and information contained therein. Chapter 8, in this section, contains datasheets for each of the TTL-equivalent macros available in the MACH library. Each lettered paragraph below corresponds to a circled letter in the next two figures.

- A. The feature summary is a bulleted list that summarizes the logical features of the macro.
- B. The logic symbol illustrates the macro pin names and how they appear on the symbol.
- C. The MACH resource summary lists the number of resources consumed by a stand-alone macro. This summary does not account for potential minimization or conflicts with other logic in a design. The following utilized resources are listed.
 - Macrocell count
 - Array inputs
 - Product terms used
 - Product terms allocated
 - MACH-family restrictions, if applicable
- D. The functional description explains the logical operation of the macro, and corresponds to the information presented in the function table. It also indicates if the macro is a non-standard TTL implementation.
- E. The sample PDS equivalent represents the PDS file generated when the macro schematic is processed as a stand-alone design.

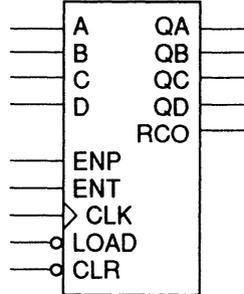
- F. The function table illustrates the functional operation of the macro. Logic states are represented as follows.
- H represents a logical 1, or HIGH, state.
 - L represents a logical 0, or LOW, state.
 - X represents a don't-care condition.
 - An up arrow represents a rising edge.
- G. The schematic is a logical representation of the macro circuit as created in OrCAD/SDT III. The following graphic changes have been made to enhance the clarity of the schematics. No changes have been made to the logic.
- Reference designators have been removed.
 - Part values have been removed from non-storage macros.
 - Module-port names have been relocated to a position above the module-port symbol.
 - Some pin names have been removed to prevent crowding.

(A)

- Synchronous 4-bit binary-loadable up counter
- Synchronous reset
- Carry look-ahead output for making wider counters

(B)

Logic Symbol



(C)

Macrocell count: 5
 Array inputs: 12
 Product terms used: 17
 Product terms allocated: 20

(D)

Functional Description

The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count of 15 has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

(E)

Sample PDS Equivalent

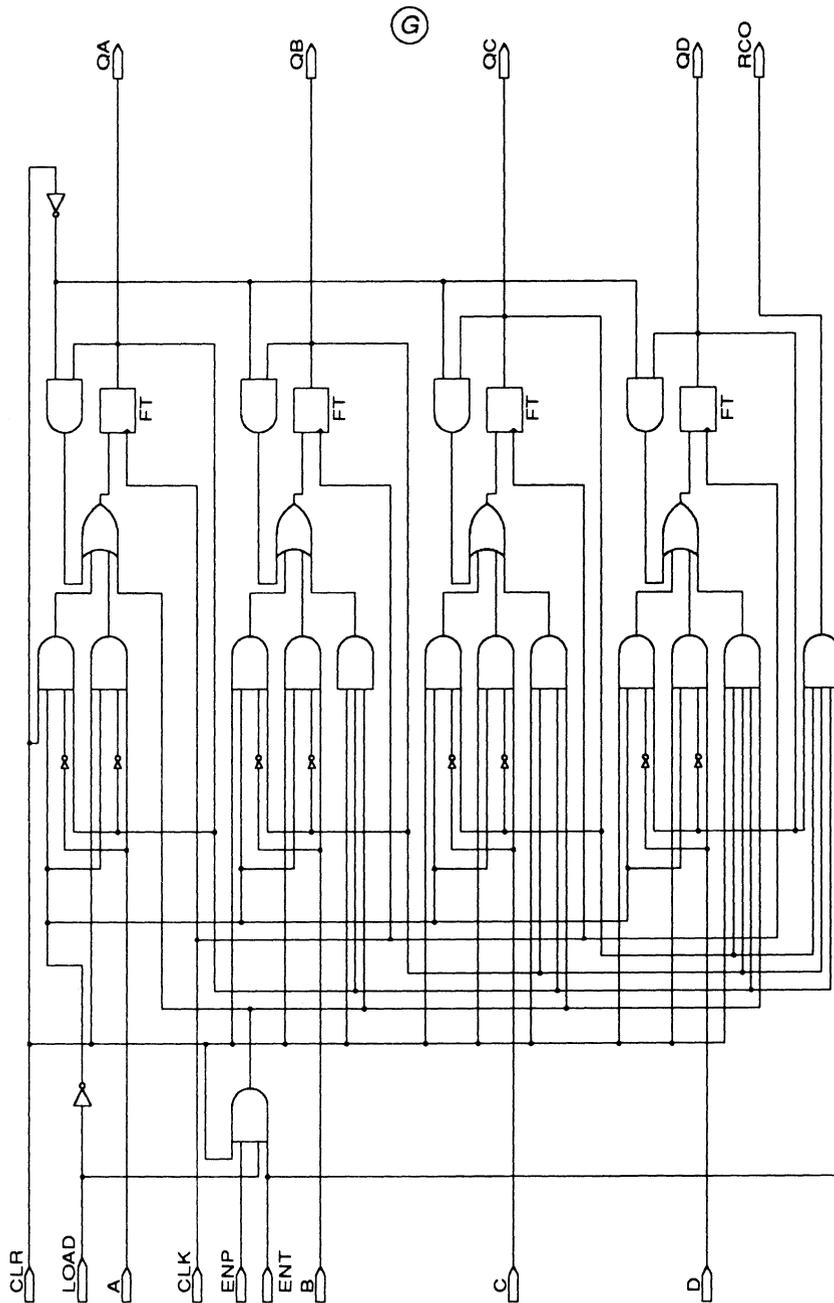
QA.T = ((/CLR * QA) + (CLR * /LOAD * /A * QA) + (CLR * /LOAD * /QA * A) + (CLR * ENP * LOAD * ENT))
 QA.clkf = CLK
 QB.T = ((/CLR * QB) + (CLR * /LOAD * /B * QB) + (CLR * /LOAD * /QB * B) + (CLR * QA * (CLR * ENP * LOAD * ENT)))
 QB.clkf = CLK
 QC.T = ((/CLR * QC) + (CLR * /LOAD * /C * QC) + (CLR * /LOAD * /QC * C) + (CLR * QB * QA * (CLR * ENP * LOAD * ENT)))
 QC.clkf = CLK
 QD.T = ((/CLR * QD) + (CLR * /LOAD * /D * QD) + (CLR * /LOAD * /QD * D) + (CLR * QC * QB * QA * (CLR * ENP * LOAD * ENT)))
 QD.clkf = CLK
 RCO = (QD * QC * QB * QA * ENT)

(F)

Function Table

	Inputs					Outputs			
Mode	CLK	CLR	Load	ENP	ENT	QD	QC	QB	QA
Clear	L	L	X	X	X	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 15 and ENT is HIGH. Otherwise, it stays LOW.



7.2.2 LATCH MACROS

MACH 2 series devices have special resources to implement latch macros directly. MACH 1 series devices implement the latch macros as combinatorial functions.

7.2.3 MINIMIZATION

During compilation, logic is replaced with its DeMorgan equivalent if the latter implementation is more economical.

Important: When DeMorganized equivalents are substituted for clocked macros, polarity inversion occurs after the storage element. As a result, the operation of the Set/Reset logic is reversed when viewed at the pin.

To prevent this reversal, you use the NODE macro and specify the NO_MIN attribute. Inhibiting minimization also preserves the redundant hold used in latch designs to prevent timing glitches.

7.2.4 REFERENCE DESIGNATORS

Each instance of a macro in a schematic must have a unique reference designator. Reference designators are automatically assigned during the compilation process.

Note: Each reference designator must be unique across all levels of hierarchy, including the subsheets of MACH macros.

The following types of reference designators are used in the AMD-supplied macro libraries.

- **M?** appears with combinatorial, TTL-equivalent, and non-three-state buffer macros.
- **X?** appears with storage and three-state macros.

- IO? appears with the NODE and NC macros.

Tip: Error and warning messages include the reference designator and net name of the flagged logic; therefore, it is a good idea to name all nets in the design.

7.2.5 SIGNAL NAMES

Use an alphanumeric string no longer than nine characters for each signal name. Signal names are not case sensitive.

Important: A slash, /, does **not** affect the polarity in a schematic signal name.

7.3 ANNOTATED DATASHEET

This discussion presents an annotated datasheet to illustrate the layout and information contained therein. Chapter 8 contains datasheets for each of the TTL-equivalent macros available in the MACH library. Each lettered paragraph below corresponds to a circled letter in the next two figures.

- A. The feature summary is a bulleted list that summarizes the logical features of the macro.
- B. The logic symbol illustrates the macro pin names and how they appear on the symbol.
- C. The MACH resource summary lists the number of resources consumed by a stand-alone macro. This summary does not account for potential minimization or conflicts with other logic in a design. The following utilized resources are listed.
 - Macrocell count
 - Array inputs
 - Product terms used
 - Product terms allocated
 - MACH-family restrictions, if applicable
- D. The functional description explains the logical operation of the macro, and corresponds to the information presented in the function table. It also indicates if the macro is a non-standard TTL implementation.
- E. The sample PDS equivalent represents the PDS file that is generated when the macro schematic is processed as a stand-alone design.

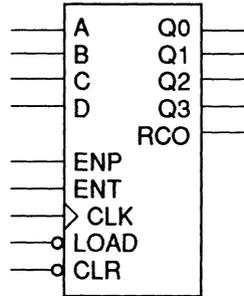
- F. The function table illustrates the functional operation of the macro. Logic states are represented as follows.
- 1 represents a logical 1, or HIGH state
 - 0 represents a logical 0, or LOW state
 - X represents a don't-care condition
 - An arrow represents a rising edge
- G. The schematic is a logical representation of the macro circuit as created in OrCAD/SDT III. The following graphic changes have been made to enhance the clarity of the schematics. No changes have been made to the logic.
- Reference designators have been removed.
 - Part values have been removed from non-storage macros.
 - Module-port names have been relocated to a position above the module-port symbol.
 - Some pin names have been removed to prevent crowding.

(A)

- Synchronous 4-bit binary-loadable up counter
- Synchronous reset
- Carry look-ahead output for making wider counters

(B)

Logic Symbol



(C)

Macrocell count: 5
 Array inputs: 12
 Product terms used: 17
 Product terms allocated: 20

(D)

Functional Description

The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count of 15 has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

(E)

Sample PDS Equivalent

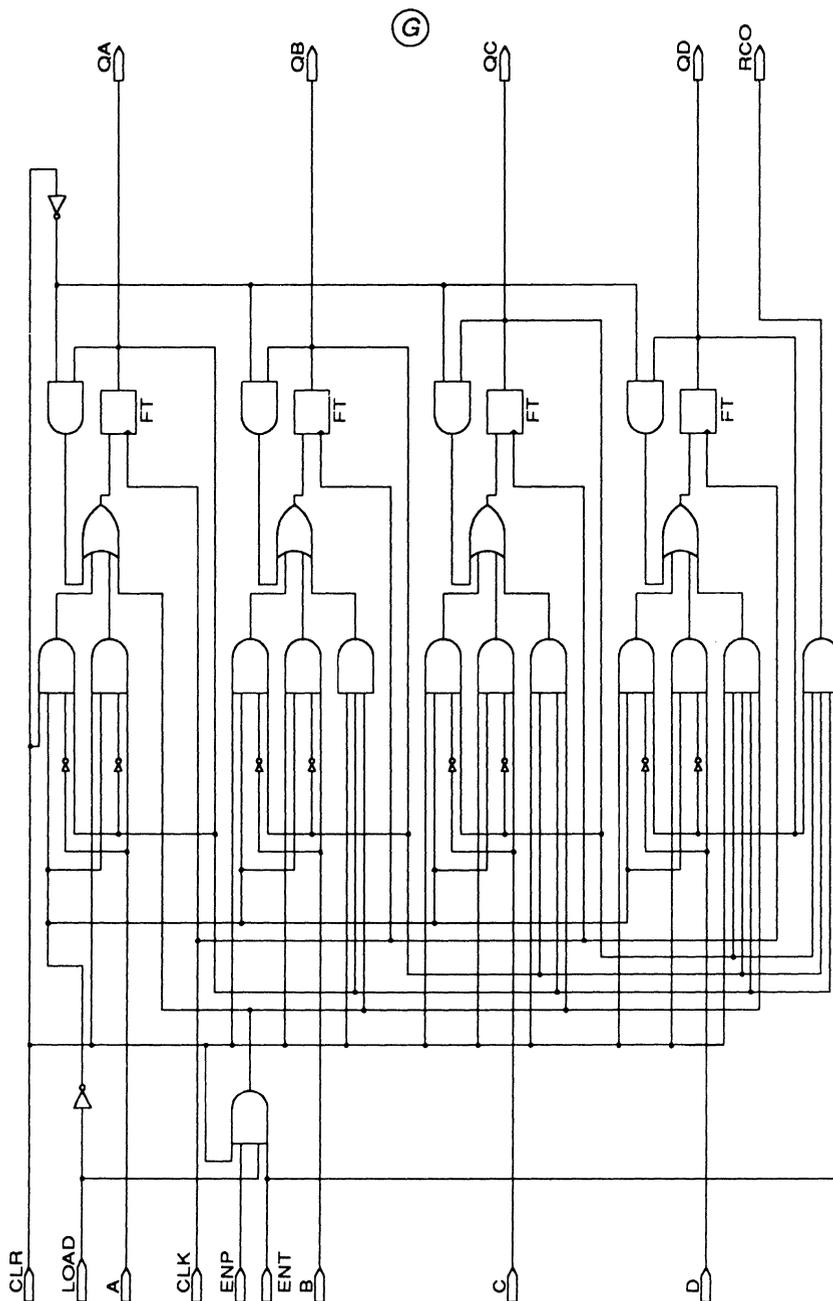
QA.T = (((CLR * QA) + (CLR * /LOAD * /A * QA) + (CLR * /LOAD * /QA * A) + (CLR * ENP * LOAD * ENT))
 QA.clkf = CLK
 QB.T = (((CLR * QB) + (CLR * /LOAD * /B * QB) + (CLR * /LOAD * /QB * B) + (CLR * QA * (CLR * ENP * LOAD * ENT)))
 QB.clkf = CLK
 QC.T = (((CLR * QC) + (CLR * /LOAD * /C * QC) + (CLR * /LOAD * /QC * C) + (CLR * QB * QA * (CLR * ENP * LOAD * ENT)))
 QC.clkf = CLK
 QD.T = (((CLR * QD) + (CLR * /LOAD * /D * QD) + (CLR * /LOAD * /QD * D) + (CLR * QC * QB * QA * (CLR * ENP * LOAD * ENT)))
 QD.clkf = CLK
 RCO = (QD * QC * QB * QA * ENT)

(F)

Function Table

	Inputs					Outputs			
Mode	CLK	CLR	Load	ENP	ENT	QD	QC	QB	QA
Clear	L	L	X	X	X	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 15 and ENT is HIGH. Otherwise, it stays LOW.



CHAPTER 8

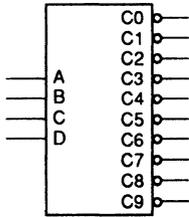
MACRO AND SCHEMATIC DATASHEETS

CONTENTS

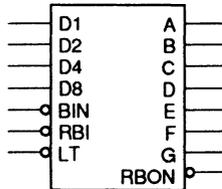
Quick Reference	1
7442 BCD-to-Decimal Decoder	18
Schematic	19
7448 BCD-to-7-Segment Decoder	20
Schematic	21
7477 4-Bit Latch	22
Schematic	23
7482 2-Bit Full Adder	24
Schematic	25
7483 4-Bit Full Adder	26
Schematic	27
7485 4-Bit Magnitude Comparator	28
Schematic	29
7491 8-Bit Shift Register	30
Schematic	31
7494 4-Bit Shift Register	32
Schematic	33
1 Bit of a 7494 Shift Register	34
7496 5-Bit Shift Register	36
Schematic	37
1 Bit of a 7496 Shift Register	38
74116 4-Bit Latch w/Clear	40
Schematic	41
74138 3-to-8 Line Decoder	42
Schematic	43
74139 2-to-4 Line Decoder	43
Schematic	45
74147 10-to-4 Priority Line Encoder	46
Schematic	47
74148 8-to-3 Priority Line Decoder	48
Schematic	49
74150 16-to-1 Multiplexer w/Enable	50
Schematic	51
74151 8-to-1 Multiplexer w/Enable	52
Schematic	53
74153 Dual 4-to-1 Multiplexer w/Enable	54
Schematic	55
74154 4-to-16 Line Decoder	56
Schematic	57
74157 Quad 2-to-1 Multiplexer	58
Schematic	59

74158	Quad 2-to-1 Multiplexer	60
	Schematic	61
74162	4-Bit BCD/Decode Counter w/Synchronous Reset	62
	Schematic	63
74163	4-Bit Binary Counter w/Synchronous Reset	64
	Schematic	65
74164	8-Bit Serial-In Parallel-out Shift Register	66
	Schematic	67
74165	Parallel-Load 8-Bit Shift Register	68
	Schematic	69
	1 Bit of a 74165 Shift Register	70
74166	8-Bit Parallel-In Serial-Out Shift Register	72
	Schematic	73
	1 Bit of a 74166 Shift Register	74
74192	4-Bit Up/Down BCD Counter	76
	Schematic	77
74193	4-Bit Up/Down Binary Counter	78
	Schematic	79
74194	4-Bit Bidirectional Universal Shift Register	80
	Schematic	81
	1 Bit of a 74194 Shift Register	82
74240	Octal Inverting Buffers w/3-State Outputs	84
	Schematic	85
74244	Octal Non-Inverting Buffers w/3-State Outputs	86
	Schematic	87
74245	Octal Bus Transcribers w/3-State Outputs	88
	Schematic	89
74259	8-Bit Addressable Latch	90
	Schematic	91
	1 Bit of a 74259 Latch	92
74273	Octal D-Type Flip-Flops	94
	Schematic	95
74280	9-Bit Parity Generator/Checker	96
	Schematic	97
74298	Quad 2-to-1 Multiplexer w/Storage	98
	Schematic	99
74299X	8-Bit Bidirectional Universal Shift Register	100
	Schematic	101
74373	Octal D-Type Latches w/3-State Outputs	102
	Schematic	103
74374	Octal D-Type Flip-Flops w/3-State Outputs	104
	Schematic	105
74518	8-Bit Identity Comparator	106
	Schematic	107
74521	8-Bit Identity Comparator	108
	Schematic	109

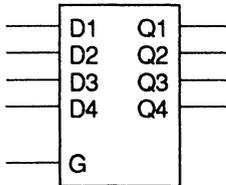
ADD1	1-Bit Full Adder	110
	Schematic	111
DECODE4	2-to-4 Line Decoder	112
	Schematic	113
MUX2	2-to-1 Multiplexer w/Enable	114
	Schematic	115
MUX4	4-to-1 Multiplexer w/Enable	116
	Schematic	117

7442**BCD-to-Decimal Decoder****7442**

The 7442 macro decodes a 4-bit BCD input into a single active-LOW output. All outputs remain HIGH for invalid inputs.

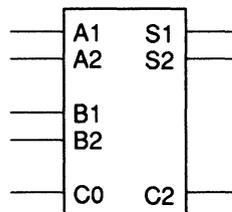
7448**BCD-to-7-Segment Decoder****7448**

The 7448 macro decodes a 4-bit BCD input into a 7-segment-display format.

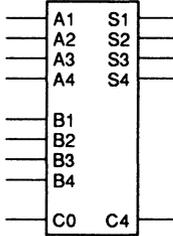
7477**4-Bit Latch****7477**

The 7477 macro is a 4-bit latch. The Q1 – Q4 outputs follow the D1 – D4 input data when the G latch enable is set HIGH. When G is set LOW, the outputs latch the input data and further activity at D1 – D4 is ignored.

Note: The TTL version contains two 4-bit latches.

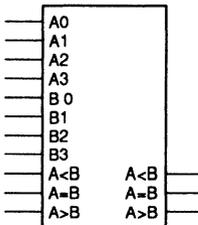
7482**2-Bit Full Adder****7482**

The 7482 macro adds two 2-bit numbers using carry look-ahead logic to generate the carry out, C2.

7483**4-Bit Full Adder****7483**

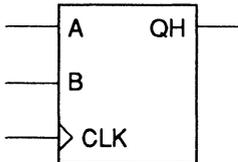
The 7483 macro adds two 4-bit numbers using carry look-ahead logic to generate the internal 2-bit carry, C2, and the final carry out, C4.

Note: The first 2-bit addition is performed as shown in the 7482 data sheet.

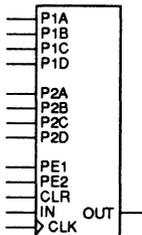
7485**4-Bit Magnitude Comparator****7485**

The 7485 macro compares two 4-bit numbers, then activates one of the three outputs: A=B, A>B, or A<B. For cascaded 7485 macros, the least-significant stage must have the A=B input HIGH and the A>B and A<B inputs LOW.

Note: The TTL version functions differently when more than one cascading input is HIGH or when all cascaded inputs are LOW.

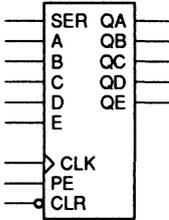
7491**8-Bit Shift Register****7491**

The 7491 macro is an 8-bit serial-in serial-out shift register. The serial-input stream is the result of logically ANDing inputs A and B.

7494**4-Bit Shift Register****7494**

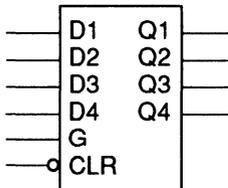
The 7494 macro is a 4-bit serial-in serial-out shift register with asynchronous clear and synchronous preset logic.

Note: The TTL version has asynchronous preset logic.

7496**5-Bit Shift Register****7496**

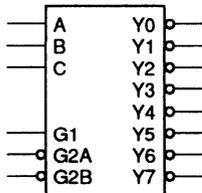
The 7496 macro is a 5-bit shift register that offers access to each flip-flop's input and output. The 5-bit value at inputs A – E is preloaded into the shift register on the rising-edge of CLK when PE is HIGH. The preset function overrides the shift operation.

Note: The TTL version has asynchronous preset logic.

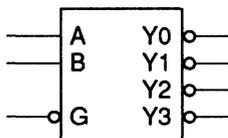
74116**4-Bit Latch w/ Clear****74116**

The 74116 macro is a 4-bit latch with an asynchronous reset.

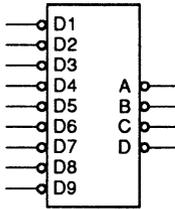
Note: The TTL version contains two 4-bit latches.

74138**3-to-8 Line Decoder****74138**

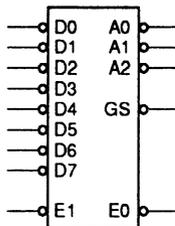
The 74138 macro decodes a 3-bit binary input into a single active-LOW output. You can cascade these macros to implement a decoder with up to 24 outputs via the three enable inputs: G1, G2A, and G2B.

74139**2-to-4 Line Decoder****74139**

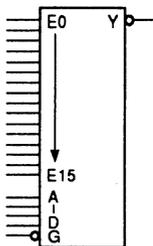
The 74139 macro decodes one of four active-LOW outputs depending on two data inputs. The active-LOW enable input, G, can be used as an input when decoding more output lines.

74147**10-to-4 Priority Line Encoder****74147**

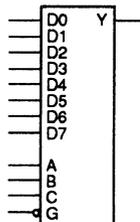
The 74147 macro generates a 4-bit BCD output code that represents the highest-order-LOW data input. Priority encoding of the inputs ensures that only the highest-order data-input line is encoded.

74148**8-to-3 Priority Line Encoder****74148**

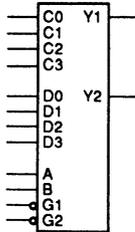
The 74148 macro generates a 3-bit binary output code that represents the highest-order-LOW data input. You can use the input enable, EI, and output enable, EO, to expand priority encoding.

74150**16-to-1 Multiplexer w/ Enable****74150**

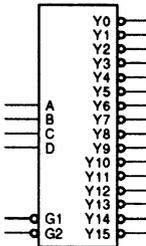
The 74150 macro decodes four data-input lines to select one of 16 data sources. The enable input, G, must be LOW to enable the Y output.

74151**8-to-1 Multiplexer w/ Enable****74151**

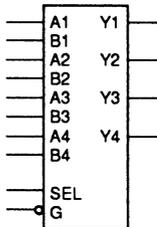
The 74151 macro decodes three data-input lines to select one of eight data sources. The enable input, G, must be LOW to enable the Y output.

74153**Dual 4-to-1 Multiplexer w/ Enable****74153**

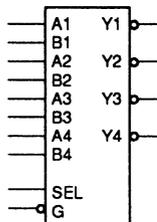
The 74153 macro consists of two 4-to-1 multiplexers with common data-select lines. Each 4-to-1 multiplexer has an active LOW strobe input line to enable the output.

74154**4-to-16 Line Decoder****74154**

The 74154 macro decodes four data-input lines to select one of 16 active LOW outputs. You can use the enable inputs, G1 and G2, to cascade multiple macros.

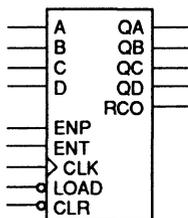
74157**Quad 2-to-1 Multiplexer****74157**

The 74157 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced LOW regardless of the inputs.

74158**Quad 2-to-1 Multiplexer****74158**

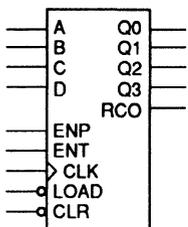
The 74158 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced HIGH regardless of the inputs.

74162 4-Bit BCD/Decade Counter w/ Synchronous Reset 74162



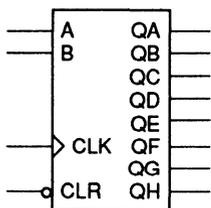
The 74162 macro is a 4-bit BCD-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded.

74163 4-Bit Binary Counter w/ Synchronous Reset 74163



The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded.

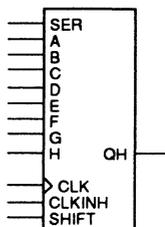
74164 8-Bit Serial-In Parallel-Out Shift Register 74164



The 74164 macro is an 8-bit serial-in parallel-out shift register with synchronous reset. The two serial inputs, A and B, are logically ANDed.

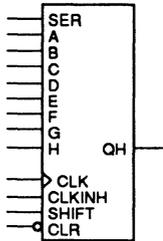
Note: The TTL version has asynchronous reset logic.

74165 Parallel-Load 8-Bit Shift Register 74165



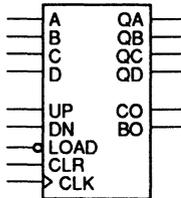
The 74165 macro is an 8-bit parallel-in serial-out shift register. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values. A parallel-load operation overrides the inhibit function.

Note: The TTL version has two clock lines and asynchronous load logic.

74166**8-Bit Parallel-In Serial-Out Shift Register****74166**

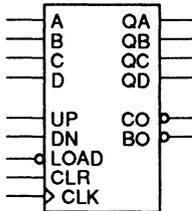
The 74166 macro is an 8-bit parallel-in serial-out shift register with synchronous reset. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values.

Note: The TTL version has asynchronous reset logic.

74192**4-Bit Up/Down BCD Counter****74192**

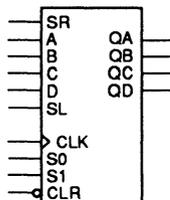
The 74192 macro is a 4-bit up/down BCD counter with synchronous parallel load and asynchronous reset logic.

Note: The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

74193**4-Bit Up/Down Binary Counter****74193**

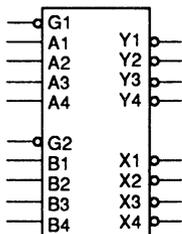
The 74193 macro is a 4-bit up/down binary counter with synchronous load and asynchronous reset logic.

Note: The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

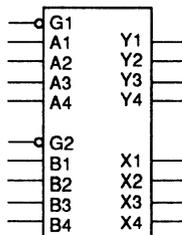
74194**4-Bit Bidirectional Universal Shift Register****74194**

The 74194 macro is a 4-bit bidirectional universal shift register with synchronous reset logic.

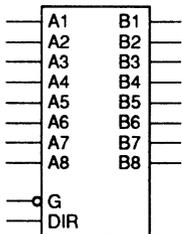
Note: The TTL version has asynchronous reset logic.

74240**Octal Inverting Buffers w/ 3-State Outputs****74240**

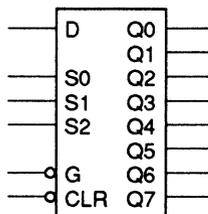
The 74240 macro contains two groups of four inverting buffers. Each group is enabled by an active-LOW input control line.

74244**Octal Non-Inverting Buffers w/ 3-State Outputs****74244**

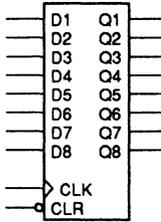
The 74244 macro contains two groups of four non-inverting buffers. Each group is enabled by an active-LOW input control line.

74245**Octal Bus Transceivers w/ 3-State Outputs****74245**

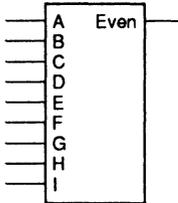
The 74245 macro implements an 8-bit bus transceiver. You can transmit data from bus A to bus B or from bus B to bus A. The data-transfer direction is controlled by the DIR control line. If the enable input, G, is set HIGH, then the buses are disabled and isolated.

74259**8-Bit Addressable Latch****74259**

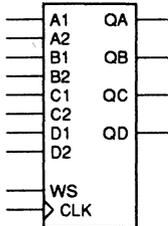
The 74259 macro is an 8-bit addressable latch with asynchronous reset logic. The following four modes of operation are selectable via the CLR and G inputs: addressable latch, memory, active-HIGH 3-to-8 demultiplexer, reset.

74273**Octal D-Type Flip-Flops****74273**

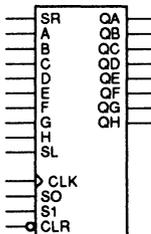
The 74273 macro is an octal D-FF bank with asynchronous reset logic.

74280**9-Bit Parity Generator/Checker****74280**

The 74280 macro is a combinational circuit that generates or checks for even parity on nine data lines. Odd parity is obtained by taking the inversion of the EVEN parity output.

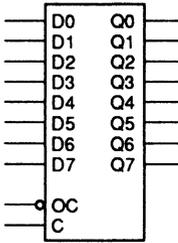
74298**Quad 2-to-1 Multiplexer with Storage****74298**

The 74298 macro selects one of two 4-bit words and stores each bit in a register on the rising edge of CLK.

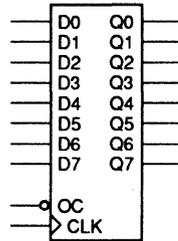
74299X**8-Bit Bidirectional Universal Shift Register****74299X**

The 74299X macro is composed of two 74194s connected to form an 8-bit bidirectional universal shift register with synchronous reset logic.

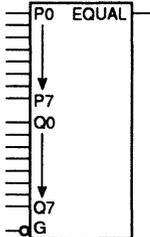
Note: The TTL version has three-state bidirectional I/Os that serve as the parallel-load inputs as well as the Q outputs.

74373**Octal D-Type Latches with 3-State Outputs****74373**

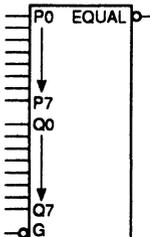
The 74373 macro is an octal D latch with an active-LOW enable input.

74374**Octal D-Type Flip-Flops with 3-State Outputs****74374**

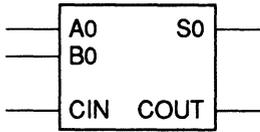
The 74374 macro is an octal D-type register with an active-LOW enable input.

74518**8-Bit Identity Comparator****74518**

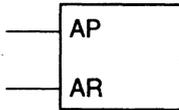
The 74518 macro compares two 8-bit numbers and sets the EQUAL output HIGH if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

74521**8-Bit Identity Comparator****74521**

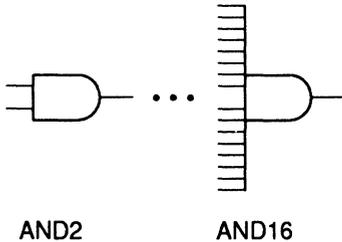
The 74521 macro compares two 8-bit numbers and sets the EQUAL output LOW if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

ADD1**1-Bit Full Adder****ADD1**

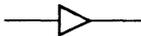
The ADD1 macro adds two 1-bit numbers. You can use the carry-out and carry-in signals to cascade multiple adders

AINIT**Implicit Preset and Reset****AINIT**

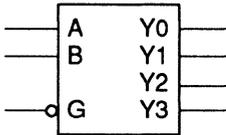
The AINIT macro specifies the asynchronous preset and reset functions for all of the associated three-terminal flip-flops and latches.

AND**AND Gates****AND**

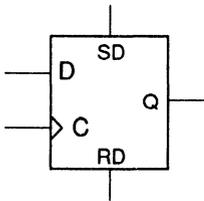
The AND2 thru AND16 macros are AND gates with two to 16 inputs.

BUF**Buffer****BUF**

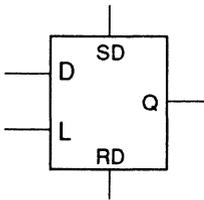
The BUF macro is a single input non-inverting buffer.

DECODE4**2-to-4 Line Decoder****DECODE4**

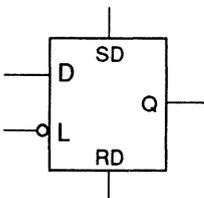
The DECODE4 macro decodes one of four active-HIGH output lines depending on the 2-bit data inputs. The enable input, G, must be LOW to activate the decoder. You can use the enable inputs to cascade multiple decoders.

DFF**D Flip-Flop****DFF**

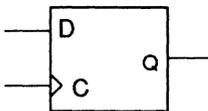
The DFF macro is a five-terminal D-type flip-flop.

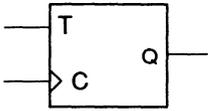
DLAT**D Latch****DLAT**

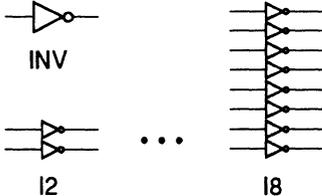
The DLAT macro is a five-terminal latch with an active-high latch input.

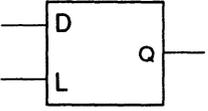
DLATX**D Latch****DLATX**

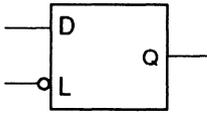
The DLATX macro is a five-terminal latch with an active-low latch input.

FD	D Flip-Flop	FD
	<p>The FD macro is a three-terminal D-type flip-flop. Use the AINIT macro to specify the asynchronous preset and reset functions.</p>	

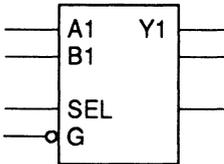
FT	T Flip-Flop	FT
	<p>The FT macro is a three-terminal T-type flip-flop. Use the AINIT macro to specify the asynchronous preset and reset functions.</p>	

INV, I	Inverters	INV, I
	<p>The INV macro is a single input inverter.</p> <p>The I2 through I8 macros are banks of inverters with two to eight elements.</p>	

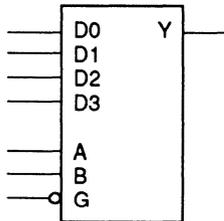
LD	D Latch	LD
	<p>The LD macro is a three-terminal latch with an active-high latch input. Use the AINIT macro to specify the asynchronous preset and reset functions.</p>	

LDX**D Latch****LDX**

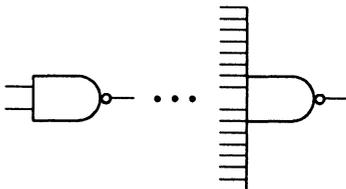
The LDX macro is a three-terminal latch with an active-low latch input. Use the AINIT macro to specify the asynchronous preset and reset functions.

MUX2**2-to-1 Multiplexer w/ Enable****MUX2**

The MUX2 macro decodes one data-input line to select one of two data sources. The enable input, G, must be LOW to enable the Y1 output.

MUX4**4-to-1 Multiplexer w/ Enable****MUX4**

The MUX4 macro decodes two data-input lines to select one of four data sources. The enable input, G, must be LOW to enable the Y output.

NAND**NAND Gates****NAND**

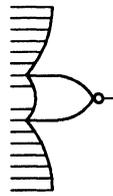
NAND2

NAND16

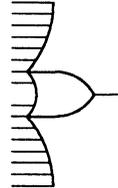
The NAND2 thru NAND16 macros are NAND gates with two to 16 inputs.

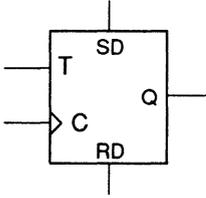
NC	No Connect	NC
	<p>The NC macro terminates unused outputs from other macros.</p>	

NODE	Node	NODE
	<p>The NODE macro forces a signal to an internal node and provides attributes for specifying node #, blockname, and no minimization.</p> <p>When connected to a module port, this macro does not force an internal node, but provides an attribute for specifying the pin #.</p>	

NOR	NOR Gates	NOR
 <p style="text-align: center; margin-top: 5px;">NOR2</p>	<p>...</p>  <p style="text-align: center; margin-top: 5px;">NOR16</p>	<p>The NOR2 through NOR16 macros are NOR gates with two to 16 inputs.</p>

NTRST	Inverting Three-State Buffer	NTRST
	<p>The NTRST macro is an inverting three-state buffer.</p>	

OR	OR Gates	OR
		
OR2	OR16	
<p>The OR2 through OR16 macros are OR gates with two to 16 inputs.</p>		

TFF	T Flip-Flop	TFF
		
<p>The TFF macro is a five-terminal T-type flip-flop.</p>		

TRST	Three-State Buffer	TRST
		
<p>The TRST macro is a non-inverting three-state buffer.</p>		

XNOR	XNOR Gates	XNOR
		
XNOR2	XNOR4	
<p>The XNOR2 through XNOR4 macros are XNOR gates with two to four inputs.</p>		



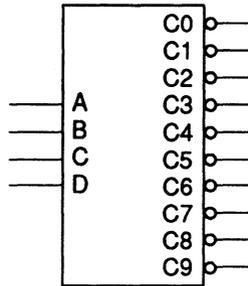
XOR2

XOR4

The XOR2 through XOR4 macros are XOR gates with two to four inputs.

- Active-LOW outputs

Logic Symbol



Macrocell count: 10
 Array inputs: 4
 Product terms used: 10
 Product terms allocated: 40

Functional Description

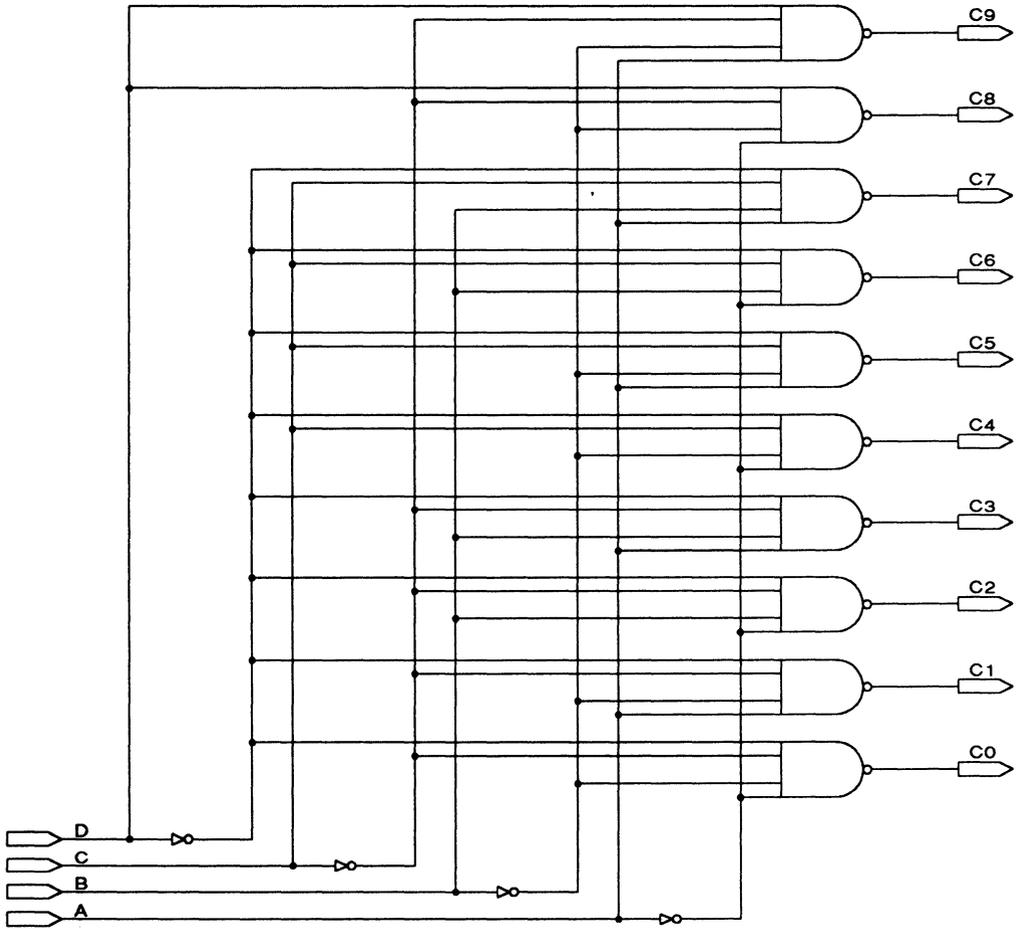
The 7442 macro decodes a 4-bit BCD input into a single active-LOW output. All outputs remain HIGH for invalid inputs.

Sample PDS Equivalent

$C9 = \overline{(\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot A)}$
 $C8 = \overline{(\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A})}$
 $C7 = \overline{(\overline{D} \cdot \overline{C} \cdot B \cdot A)}$
 $C6 = \overline{(\overline{D} \cdot \overline{C} \cdot B \cdot \overline{A})}$
 $C5 = \overline{(\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot A)}$
 $C4 = \overline{(\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A})}$
 $C3 = \overline{(\overline{D} \cdot \overline{C} \cdot B \cdot A)}$
 $C2 = \overline{(\overline{D} \cdot \overline{C} \cdot B \cdot \overline{A})}$
 $C1 = \overline{(\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot A)}$
 $C0 = \overline{(\overline{D} \cdot \overline{C} \cdot \overline{B} \cdot \overline{A})}$

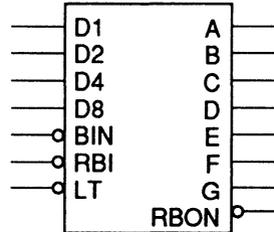
Function Table

Value	BCD Inputs				Decimal Outputs									
	D	C	B	A	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
0	L	L	L	L	L	H	H	H	H	H	H	H	H	H
1	L	L	L	H	H	L	H	H	H	H	H	H	H	H
2	L	L	H	L	H	H	L	H	H	H	H	H	H	H
3	L	L	H	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	L	H	L	H	H	H	H	H	H	L	H	H	H	H
6	L	H	H	L	H	H	H	H	H	H	L	H	H	H
7	L	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	L	L	L	H	H	H	H	H	H	H	H	L	H
9	H	L	L	H	H	H	H	H	H	H	H	H	H	L
10	H	L	H	L	H	H	H	H	H	H	H	H	H	H
11	H	L	H	H	H	H	H	H	H	H	H	H	H	H
12	H	H	L	L	H	H	H	H	H	H	H	H	H	H
13	H	H	L	H	H	H	H	H	H	H	H	H	H	H
14	H	H	H	L	H	H	H	H	H	H	H	H	H	H
15	H	H	H	H	H	H	H	H	H	H	H	H	H	H



- Lamp-test feature
- Leading- and trailing-zero blanking

Logic Symbol



Macrocell count: 8
 Array inputs: 8
 Product terms used: 32
 Product terms allocated: 48

Functional Description

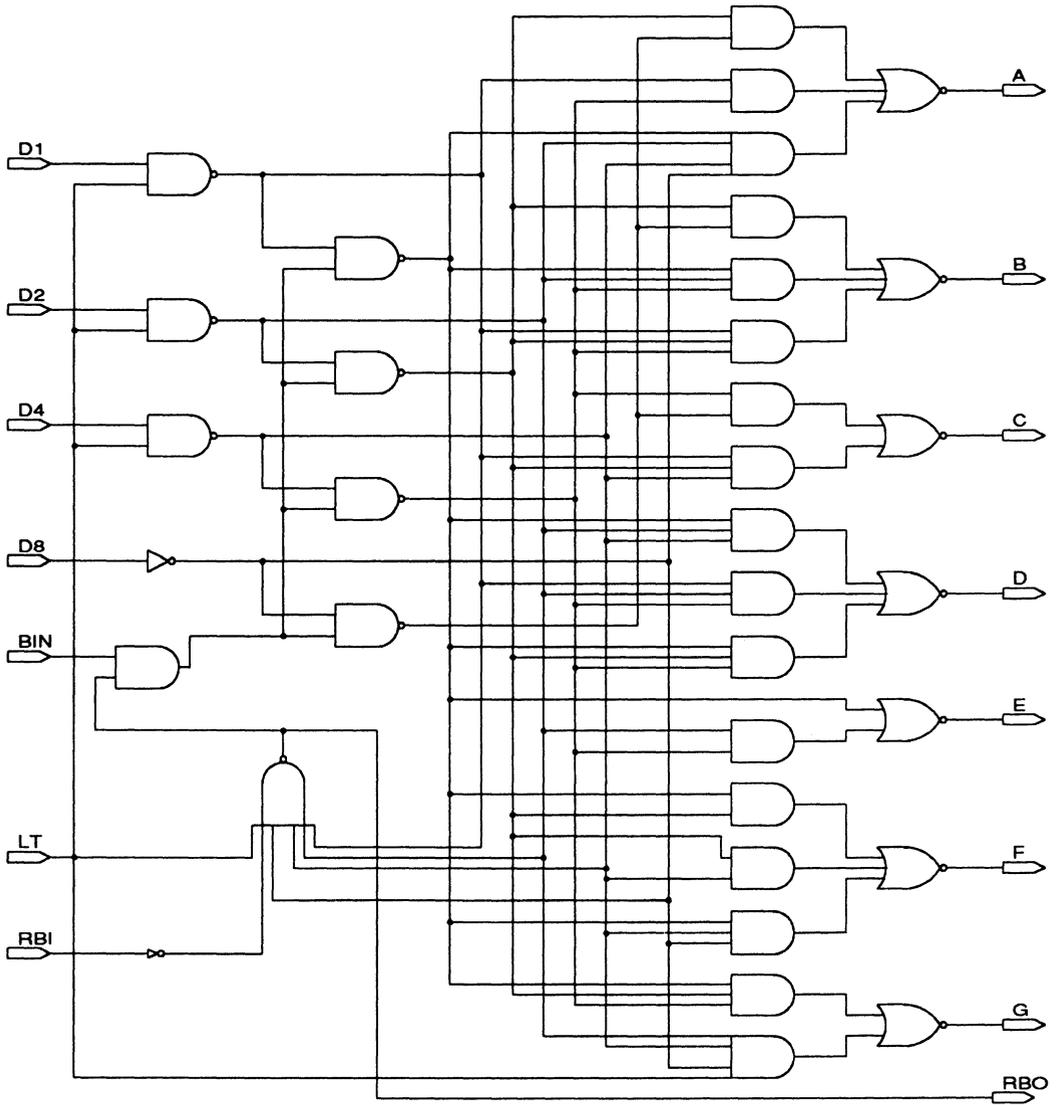
The 7448 macro decodes a 4-bit BCD input into a 7-segment-display format. To turn off all display segments, you can set the lamp-test input, LT, LOW. When the blanking input, BIN, is set LOW, all segments are turned off regardless of the BCD inputs. When the ripple-blanking input, RBI, and all BCD inputs are set LOW, and LT is set HIGH, all segments and the ripple-blanking outputs, RBO, are turned off. You can connect the RBO to the RBI of adjacent 7448 macros to turn off leading or trailing zeros.

Sample PDS Equivalent

$$\begin{aligned}
 A &= \overline{\overline{\overline{(D2 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{D8}} \\
 &\quad \cdot (BIN \cdot RBON)) + \overline{\overline{(D1 \cdot LT)} \cdot \overline{\overline{(D4 \cdot LT)}}} \\
 &\quad \cdot (BIN \cdot RBON)) + \overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D4 \cdot LT)}} \cdot \overline{\overline{D8}} \\
 B &= \overline{\overline{\overline{(D2 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{D8}} \\
 &\quad \cdot (BIN \cdot RBON)) + \overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D4 \cdot LT)}} \cdot (BIN \cdot RBON)) \\
 &\quad + \overline{\overline{(D1 \cdot LT)} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot (BIN \cdot RBON))} \\
 &\quad \cdot \overline{\overline{(D4 \cdot LT)}} \cdot \overline{\overline{(D1 \cdot LT)} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot (BIN \cdot RBON))} \\
 C &= \overline{\overline{\overline{(D4 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{D8}} \\
 &\quad \cdot (BIN \cdot RBON)) + \overline{\overline{(D1 \cdot LT)} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot (BIN \cdot RBON))} \\
 &\quad \cdot \overline{\overline{(D4 \cdot LT)}} \\
 D &= \overline{\overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D4 \cdot LT)}} + \overline{\overline{(D1 \cdot LT)} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D4 \cdot LT)}} \cdot (BIN \cdot RBON))} \\
 &\quad + \overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D4 \cdot LT)}} \\
 &\quad \cdot (BIN \cdot RBON)) \\
 E &= \overline{\overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D4 \cdot LT)}} \cdot (BIN \cdot RBON)) \\
 RBON &= \overline{\overline{(LT \cdot RBI \cdot D8 \cdot D4 \cdot LT)}} \\
 &\quad \cdot \overline{\overline{(D2 \cdot LT)}} \cdot \overline{\overline{(D1 \cdot LT)}} \cdot F = \overline{\overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{(D2 \cdot LT)}} \\
 &\quad \cdot (BIN \cdot RBON)) + \overline{\overline{(D2 \cdot LT)} \cdot (BIN \cdot RBON)}} \cdot \overline{\overline{(D4 \cdot LT)}} + \overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}} \cdot \overline{\overline{(D4 \cdot LT)}} \cdot \overline{\overline{D8}} \\
 G &= \overline{\overline{\overline{(D1 \cdot LT)} \cdot (BIN \cdot RBON)}}} \cdot \overline{\overline{(D2 \cdot LT)}} \cdot (BIN \cdot RBON)) \cdot \overline{\overline{(D4 \cdot LT)}} \\
 &\quad \cdot (BIN \cdot RBON)) + \overline{\overline{(D2 \cdot LT)} \cdot \overline{\overline{(D4 \cdot LT)}}} \\
 &\quad \cdot \overline{\overline{D8}} \cdot \overline{\overline{LT}})
 \end{aligned}$$

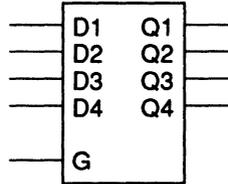
Function Table

Dec	Inputs						Outputs								
	LT	RBI	D8	D4	D2	D1	RBO	BIN	A	B	C	D	E	F	G
0	H	H	L	L	L	L	H	H	H	H	H	H	H	H	L
1	H	X	L	L	L	H	H	H	L	H	L	L	L	L	L
2	H	X	L	L	H	L	H	H	H	H	L	H	H	L	H
3	H	X	L	L	H	H	H	H	H	H	H	H	L	L	H
4	H	X	L	H	L	L	H	H	L	H	H	L	L	H	H
5	H	X	L	H	L	H	H	H	H	L	H	H	L	H	H
6	H	X	L	H	H	L	H	H	L	L	H	H	H	H	H
7	H	X	L	H	H	H	H	H	H	H	H	L	L	L	L
8	H	X	H	L	L	L	H	H	H	H	H	H	H	H	H
9	H	X	H	L	L	H	H	H	H	H	H	L	L	H	H
10	H	X	H	L	H	L	H	H	L	L	L	H	H	L	H
11	H	X	H	L	H	H	H	H	L	L	H	H	L	L	H
12	H	X	H	H	L	L	H	H	L	H	L	L	L	H	H
13	H	X	H	H	L	H	H	H	H	L	L	L	H	L	H
14	H	X	H	H	H	L	H	H	L	L	L	H	H	H	H
15	H	X	H	H	H	H	H	H	L	L	L	L	L	L	L
BIN	X	X	X	X	X	X	L	X	L	L	L	L	L	L	L
RBI	H	L	L	L	L	L	X	L	L	L	L	L	L	L	L
LT	L	X	X	X	X	X	H	H	H	H	H	H	H	H	H



- Enable input

Logic Symbol



Macrocell count: 4
 Array inputs: 9
 Product terms used: 8
 Product terms allocated: 16

Functional Description

The 7477 macro is a 4-bit latch. The Q1 – Q4 outputs follow the D1 – D4 input data when the G latch enable is set HIGH. When G is set LOW, the outputs latch the input data, and further activity at D1 – D4 is ignored.

Note:

The TTL version contains two 4-bit latches.

Sample PDS Equivalent

$$Q1 = ((Q1 * VCC * /G) + (VCC * GND) + (VCC * G * D1) + (D1 * VCC * Q1))$$

$$Q2 = ((Q2 * VCC * /G) + (VCC * GND) + (VCC * G * D2) + (D2 * VCC * Q2))$$

$$Q3 = ((Q3 * VCC * /G) + (VCC * GND) + (VCC * G * D3) + (D3 * VCC * Q3))$$

$$Q4 = ((Q4 * VCC * /G) + (VCC * GND) + (VCC * G * D4) + (D4 * VCC * Q4))$$

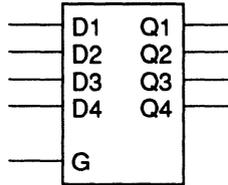
Function Table (for 1 bit)

Inputs		Outputs
G	D	Q
H	H	H
H	H	H
L	X	Q _o

* Q_o = previous state of Q

- Enable input

Logic Symbol



Macrocell count: 4
 Array inputs: 9
 Product terms used: 8
 Product terms allocated: 16

Functional Description

The 7477 macro is a 4-bit latch. The Q1 – Q4 outputs follow the D1 – D4 input data when the G latch enable is set HIGH. When G is set LOW, the outputs latch the input data, and further activity at D1 – D4 is ignored.

Note:

The TTL version contains two 4-bit latches.

Sample PDS Equivalent

$$Q1 = ((Q1 * VCC * /G) + (VCC * GND) + (VCC * G * D1) + (D1 * VCC * Q1))$$

$$Q2 = ((Q2 * VCC * /G) + (VCC * GND) + (VCC * G * D2) + (D2 * VCC * Q2))$$

$$Q3 = ((Q3 * VCC * /G) + (VCC * GND) + (VCC * G * D3) + (D3 * VCC * Q3))$$

$$Q4 = ((Q4 * VCC * /G) + (VCC * GND) + (VCC * G * D4) + (D4 * VCC * Q4))$$

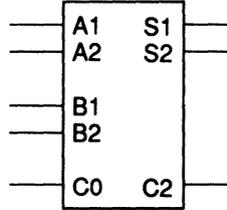
Function Table (for 1 bit)

Inputs		Outputs
G	D	Q
H	H	H
H	H	H
L	X	Q _o

* Q_o = previous state of Q

- Carry output and input

Logic Symbol



Macrocell count: 3
 Array inputs: 5
 Product terms used: 23
 Product terms allocated: 24

Functional Description

The 7482 macro adds two 2-bit numbers using carry look-ahead logic to generate the carry out, C2.

Sample PDS Equivalent

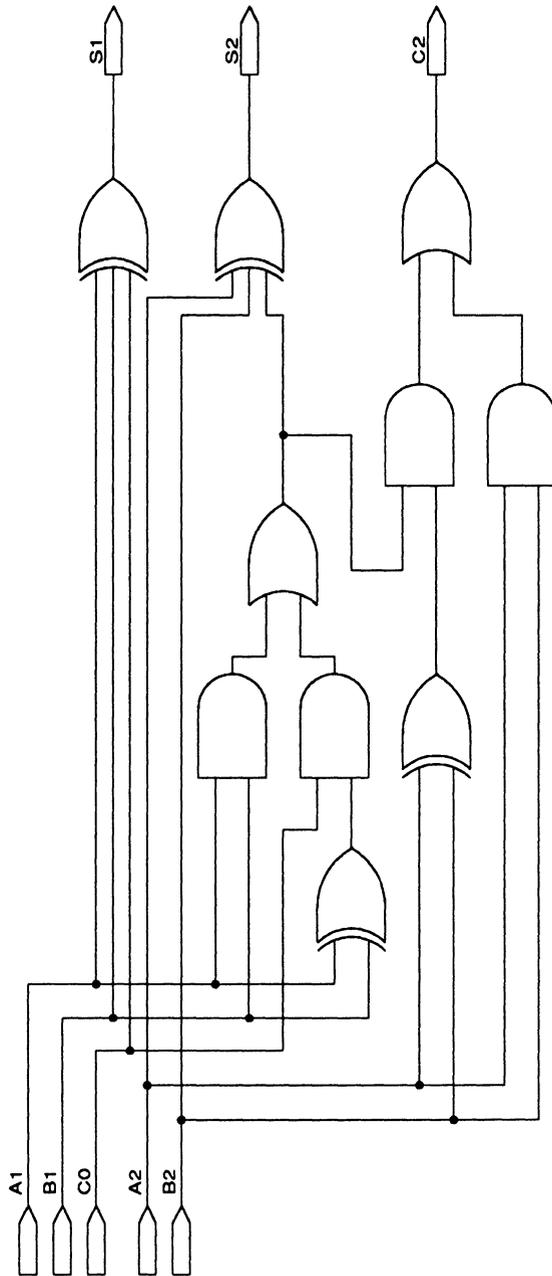
$$S1 = (A1 \oplus B1 \oplus C0)$$

$$S2 = (A2 \oplus B2 \oplus ((A1 * B1) + (C0 * (A1 \oplus B1))))$$

$$C2 = (((((A1 * B1) + (C0 * (A1 \oplus B1))) * (A2 \oplus B2)) + (A2 * B2)))$$

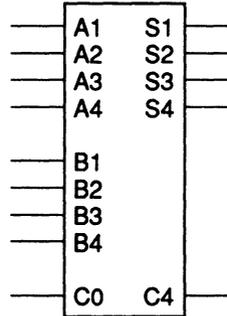
Function Table

Inputs				Outputs					
				C0=L			C0=H		
A2	A1	B2	B1	C2	S2	S1	C2	S2	S1
L	L	L	L	L	L	L	L	L	H
L	L	L	H	L	L	H	L	H	L
L	L	H	L	L	H	L	L	H	H
L	L	H	H	L	H	H	H	L	L
L	H	L	L	L	L	H	L	H	L
L	H	L	H	L	H	L	L	H	H
L	H	H	L	L	H	H	H	L	L
L	H	H	H	H	L	L	H	L	H
H	L	L	L	L	H	L	L	H	H
H	L	L	H	L	H	H	H	L	L
H	L	H	L	H	L	L	H	L	H
H	L	H	H	H	L	H	H	H	L
H	H	L	L	L	H	H	H	L	L
H	H	L	H	H	L	L	H	L	H
H	H	H	L	H	L	H	H	H	L
H	H	H	H	H	H	L	H	H	H



- Carry output and input

Logic Symbol



Macrocell count: 6
 Array inputs: 10
 Product terms used: 46
 Product terms allocated: 48

Functional Description

The 7483 macro adds two 4-bit numbers using carry look-ahead logic to generate the internal 2-bit carry, C2, and the final carry out, C4.

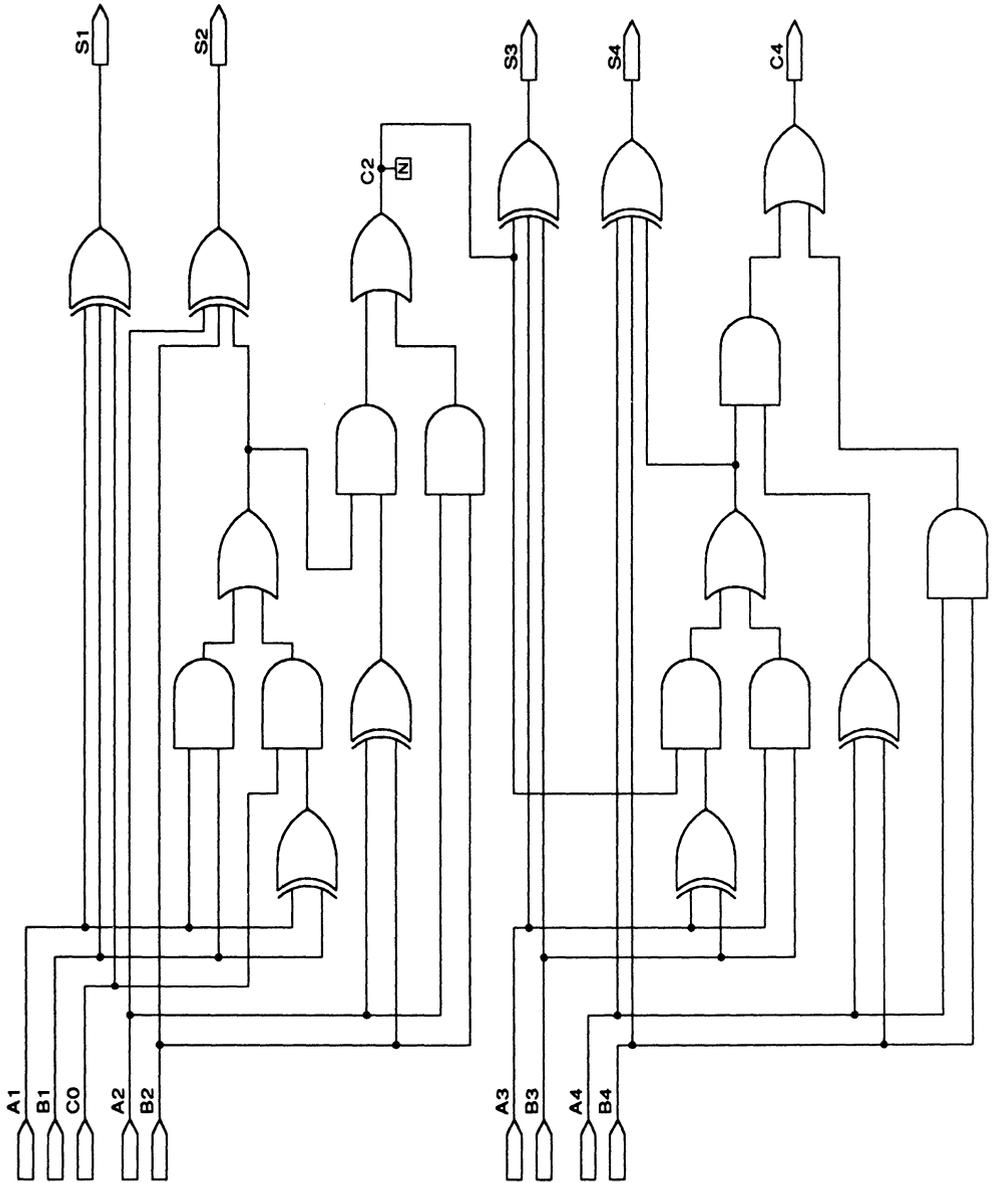
Note: The first 2-bit addition is performed as shown in the 7482 data sheet.

Sample PDS Equivalent

$$\begin{aligned}
 S1 &= (A1 \oplus B1 \oplus C0) \\
 S2 &= (A2 \oplus B2 \oplus ((A1 * B1) \\
 &\quad + (C0 * (A1 \oplus B1)))) \\
 C2 &= (((A1 * B1) \\
 &\quad + (C0 * (A1 \oplus B1))) * (A2 \oplus B2)) \\
 &\quad + (A2 * B2) \\
 S3 &= (C2 \oplus A3 \oplus B3) \\
 S4 &= (A4 \oplus B4 \oplus ((C2 * (A3 \oplus B3)) \\
 &\quad + (A3 * B3))) \\
 C4 &= (((C2 * (A3 \oplus B3)) \\
 &\quad + (A3 * B3)) * (A4 \oplus B4)) + (A4 * B4)
 \end{aligned}$$

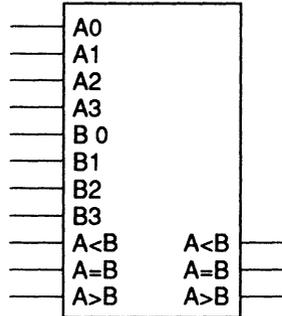
Function Table

Inputs				Outputs (second 2-bit stage*)					
				C2=L			C2=H		
A4	A3	B4	B3	C4	S4	S3	C4	S4	S3
L	L	L	L	L	L	L	L	L	H
L	L	L	H	L	L	H	L	H	L
L	L	H	L	L	H	L	L	H	H
L	L	H	H	L	H	H	H	L	L
L	H	L	L	L	L	H	L	H	L
L	H	L	H	L	H	L	L	H	H
L	H	H	L	L	H	H	H	L	L
L	H	H	H	H	L	L	H	L	H
H	L	L	L	L	L	L	L	H	H
H	L	L	H	L	H	H	H	L	L
H	L	H	L	H	L	L	H	L	H
H	L	H	H	H	L	H	H	H	L
H	H	L	L	L	H	H	H	L	L
H	H	L	H	H	L	L	H	L	H
H	H	H	L	H	L	H	H	H	L
H	H	H	H	H	H	L	H	H	H



- Equal to, greater than, and less than three cascadable compare inputs

Logic Symbol



Macrocell count: 7
 Array inputs: 15
 Product terms used: 25
 Product terms allocated: 28

Functional Description

The 7485 macro compares two 4-bit numbers, then activates one of the three outputs: A=B, A>B, or A<B. For cascaded 7485 macros, the least-significant stage must have the A=B input HIGH and the A>B and A<B inputs LOW. For intermediate stages, connect the outputs of a previous stage to the A=B, A>B, and A<B inputs.

Note:

The TTL version functions differently when more than one cascading input is HIGH or when all cascaded inputs are LOW.

Sample PDS Equivalent

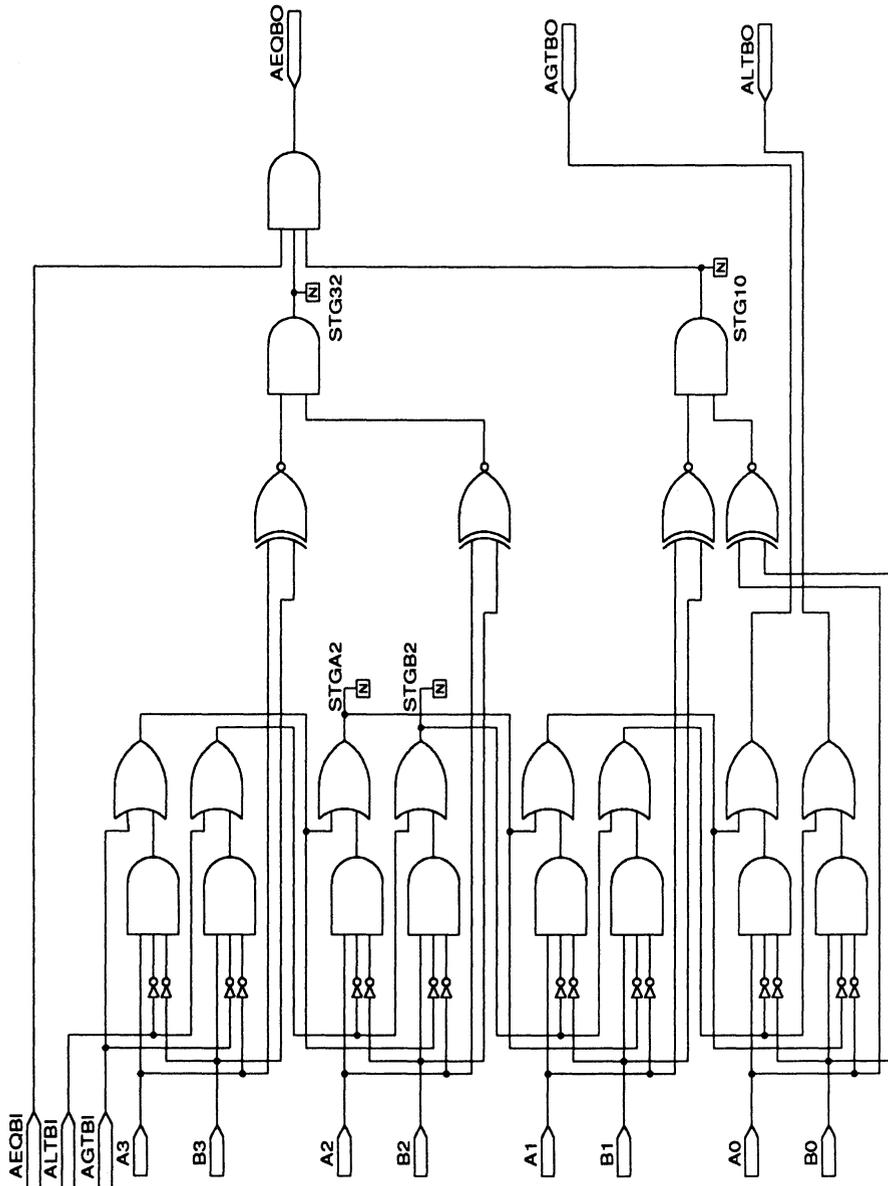
$$\begin{aligned}
 STG32 &= ((A3 \text{ :+} B3) * (A2 \text{ :+} B2)) \\
 AEQBO &= (AEQBI * STG32 * STG10) \\
 STGA2 &= ((AGTBI + (A3 * /ALB1 * /B3)) \\
 &+ (A2 * /(ALB1 + (B3 * /AGTBI * /A3)) \\
 &* /B2)) \\
 STGB2 &= ((ALB1 + (B3 * /AGTBI * /A3)) \\
 &+ (B2 * /(AGTBI + (A3 * /ALB1 * /B3)) \\
 &* /A2)) \\
 STG10 &= ((A1 \text{ :+} B1) * (A0 \text{ :+} B0)) \\
 AGTBO &= ((STGA2 + (A1 * /STGB2 \\
 &* /B1)) + (A0 * /(STGB2 + (B1 \\
 &* /STGA2 * /A1)) * /B0)) \\
 ALBTO &= ((STGB2 + (B1 * /STGA2 \\
 &* /A1)) + (B0 * /(STGA2 + (A1 \\
 &* /STGB2 * /B1)) * /A0))
 \end{aligned}$$

Function Table

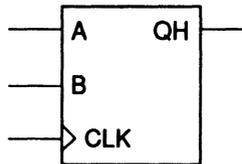
Comparing Inputs				Comparing Inputs			Outputs		
A3,B3	A2,B2	A1,B1	A0,B0	A>B	A<B	A=B	A>B	A<B	A=B
A3>B3	X	X	X	X	X	X	H	L	L
A3<B3	X	X	X	X	X	X	L	H	L
A3=B3	A2>B2	X	X	X	X	X	H	L	L
A3=B3	A2<B2	X	X	X	X	X	L	H	L
A3=B3	A2=B2	A1>B1	X	X	X	X	H	L	L
A3=B3	A2=B2	A1<B1	X	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0>B0	X	X	X	H	L	L
A3=B3	A2=B2	A1=B1	A0<B0	X	X	X	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	H	L	L	H	L	L
A3=B3	A2=B2	A1=B1	A0=B0	L	H	L	L	H	L
A3=B3	A2=B2	A1=B1	A0=B0	L	L	H	L	L	H

Invalid Conditions

A3=B3	A2=B2	A1=B1	A0=B0	X	X	H	X	X	X
A3=B3	A2=B2	A1=B1	A0=B0	H	H	L	X	X	X
A3=B3	A2=B2	A1=B1	A0=B0	L	L	L	X	X	X



- Two serial-input lines

Logic Symbol

Macrocell count: 8
 Array inputs: 9
 Product terms used: 8
 Product terms allocated: 32

Functional Description

The 7491 macro is an 8-bit serial-in serial-out shift register. The serial-input stream is the result of logically ANDing inputs A and B.

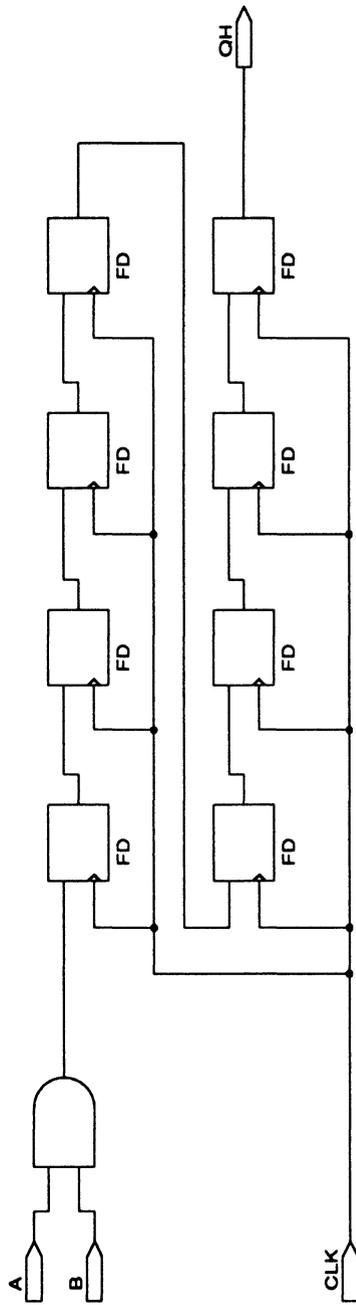
Sample PDS Equivalent

X2_D = (A * B)
 X2_D.clkf = CLK
 X3_D = X2_D
 X3_D.clkf = CLK
 X4_D = X3_D
 X4_D.clkf = CLK
 X4_Q = X4_D
 X4_Q.clkf = CLK
 X6_D = X4_Q
 X6_D.clkf = CLK
 X7_D = X6_D
 X7_D.clkf = CLK
 X8_D = X7_D
 X8_D.clkf = CLK
 QH = X8_D
 QH.clkf = CLK

Function Table

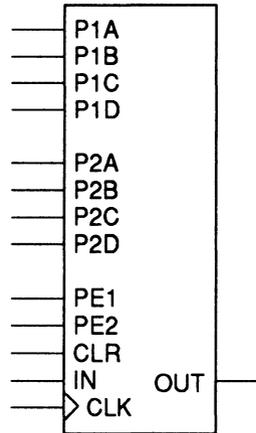
Inputs			Outputs
CLK	A	B	Q
↑	H	H	H
↑	L	X	L
↑	X	L	L
L	X	X	Q0

- Q = (AT TIME t+8)
 Q0 = previous state of Q



- Synchronous preset
- Dual preset inputs
- Asynchronous reset

Logic Symbol



Macrocell count: 4
 Array inputs: 15
 Product terms used: 12
 Product terms allocated: 16

Functional Description

The 7494 macro is a 4-bit serial-in serial-out shift register with asynchronous clear and synchronous preset logic.

You can select either 4-bit preset value by setting one preset-enable input, PE1 or PE2, HIGH. If you set both HIGH, then the preset value is the OR of the two sets of data inputs. The preset function overrides the shift operation.

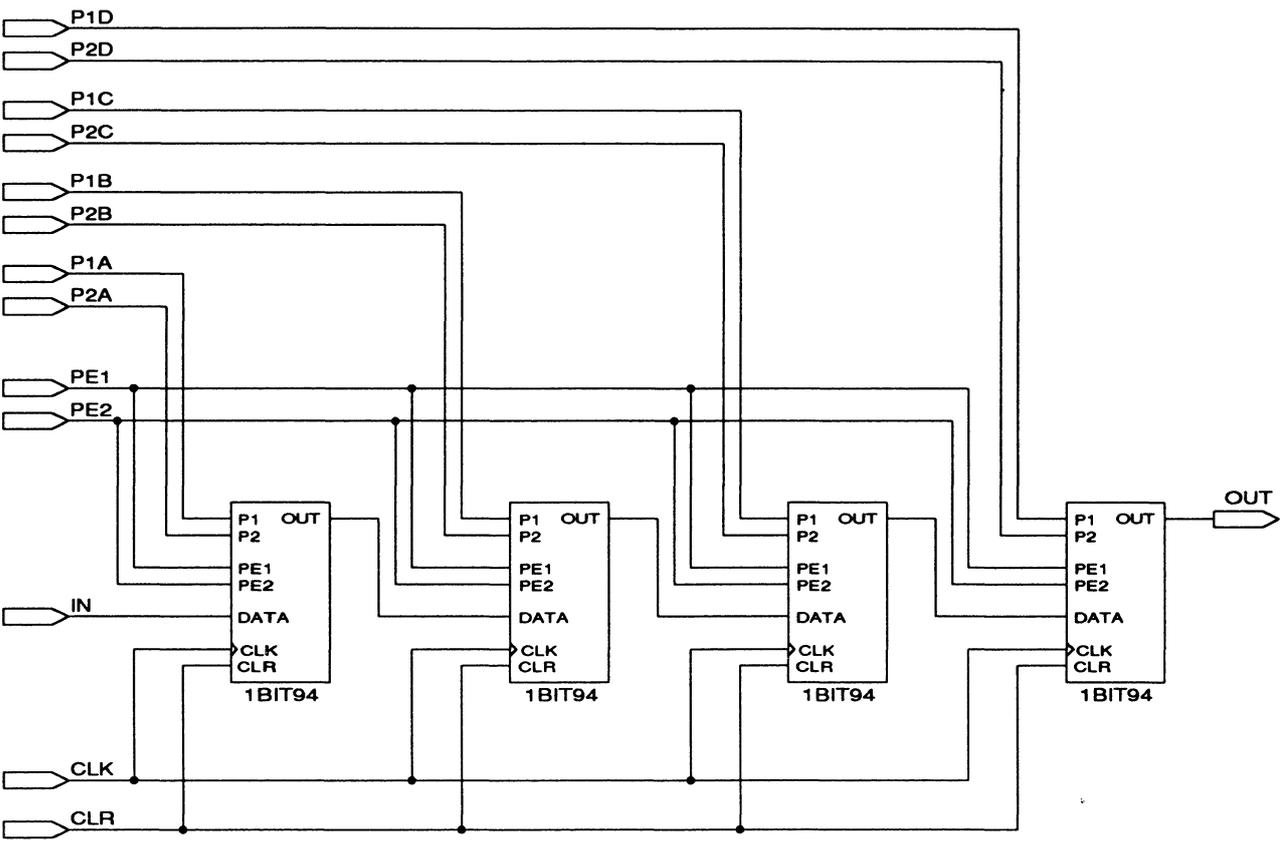
Note: The TTL version has asynchronous preset logic.

Sample PDS Equivalent

```
M1_OUT = ((P1A * PE1) + (P2A * PE2)
+ (/PE2 * /PE1 * IN))
M1_OUT.clkf = CLK
M1_OUT.rstf = CLR
M2_OUT = ((P1B * PE1) + (P2B * PE2)
+ (/PE2 * /PE1 * M1_OUT))
M2_OUT.clkf = CLK
M2_OUT.rstf = CLR
M3_OUT = ((P1C * PE1) + (P2C * PE2)
+ (/PE2 * /PE1 * M2_OUT))
M3_OUT.clkf = CLK
M3_OUT.rstf = CLR
OUT = ((P1D * PE1) + (P2D * PE2)
+ (/PE2 * /PE1 * M3_OUT))
OUT.clkf = CLK
OUT.rstf = CLR
```

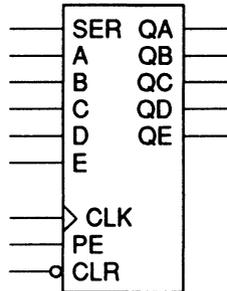
Function Table

Inputs				Outputs
PE1	P1A	PE2	P2A	Resulting Preset Value
L	X	L	X	L
L	X	X	L	L
X	L	L	X	L
X	L	X	L	L
H	H	X	X	H
H	L	L	X	L
H	L	H	L	L
H	L	H	H	H
X	X	H	H	H
L	X	H	L	L
H	L	H	L	L
H	H	H	L	H



- Synchronous preset
- Asynchronous reset
- Parallel-to-serial converter
- Serial-to-parallel converter

Logic Symbol



Macrocell count: 5
 Array inputs: 12
 Product terms used: 10
 Product terms allocated: 20

Functional Description

The 7496 macro is a 5-bit shift register that offers access to each flip-flop's input and output. The 5-bit value at inputs A – E is preloaded into the shift register on the rising-edge of CLK when PE is HIGH. The preset function overrides the shift operation.

Note:

The TTL version has asynchronous preset logic.

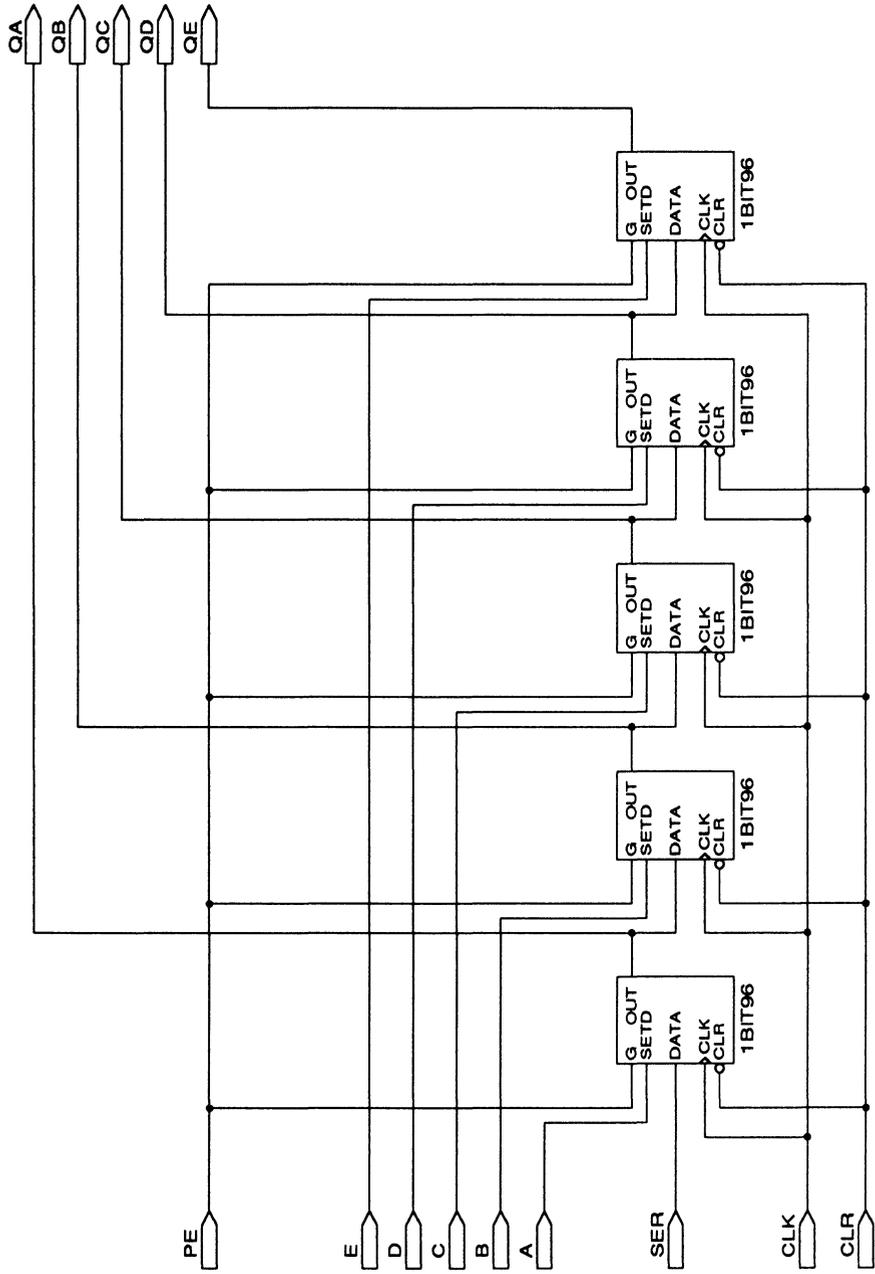
Sample PDS Equivalent

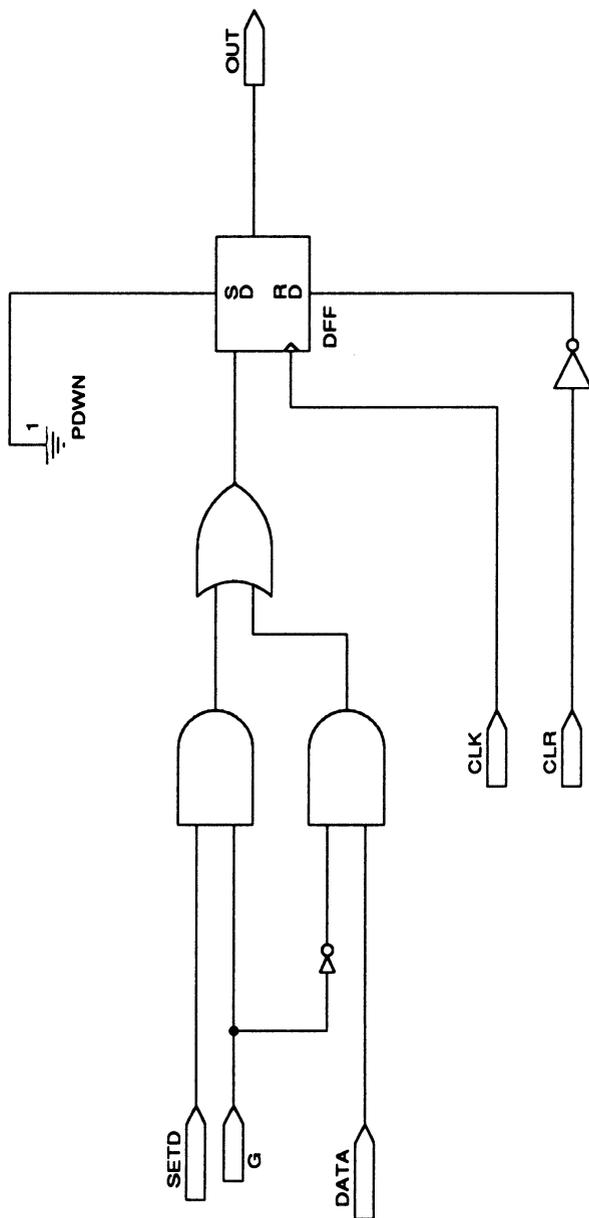
$QA = ((A * PE) + (/PE * SER))$
 QA.clkf = CLK
 QA.rstf = /CLR
 $QB = ((B * PE) + (/PE * QA))$
 QB.clkf = CLK
 QB.rstf = /CLR
 $QC = ((C * PE) + (/PE * QB))$
 QC.clkf = CLK
 QC.rstf = /CLR
 $QD = ((D * PE) + (/PE * QC))$
 QD.clkf = CLK
 QD.rstf = /CLR
 $QE = ((E * PE) + (/PE * QD))$
 QE.clkf = CLK
 QE.rstf = /CLR

Function Table

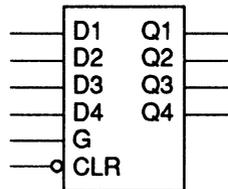
Inputs									Outputs				
CLR	PE	A	B	C	D	E	CLK	SER	QA	QB	QC	QD	QE
L	L	X	X	X	X	X	X	X	L	L	L	L	L
H	H	H	H	H	H	H	↑	X	H	H	H	H	H
H	H	L	L	L	L	L	↑	X	L	L	H	H	L
H	H	L	L	L	L	L	L	X	QA0	QB0	QC0	QD0	QE0
H	L	X	X	X	X	X	L	X	QA0	QB0	QC0	QD0	QE0
H	L	X	X	X	X	X	↑	L	L	QA0	QB0	QC0	QD0
H	L	X	X	X	X	X	↑	H	H	QA0	QB0	QC0	QD0

* QA0 to QE0 = previous state of registers QA to QE





- Asynchronous reset

Logic Symbol

Macrocell count: 4
 Array inputs: 10
 Product terms used: 8
 Product terms allocated: 16

Functional Description

The 74116 macro is a 4-bit latch with an asynchronous reset.

Note: The TTL version contains two 4-bit latches.

Sample PDS Equivalent

$$Q1 = ((Q1 * CLEAR * /G) + (CLEAR * GND) + (CLEAR * G * D1) + (D1 * CLEAR * Q1))$$

$$Q2 = ((Q2 * CLEAR * /G) + (CLEAR * GND) + (CLEAR * G * D2) + (D2 * CLEAR * Q2))$$

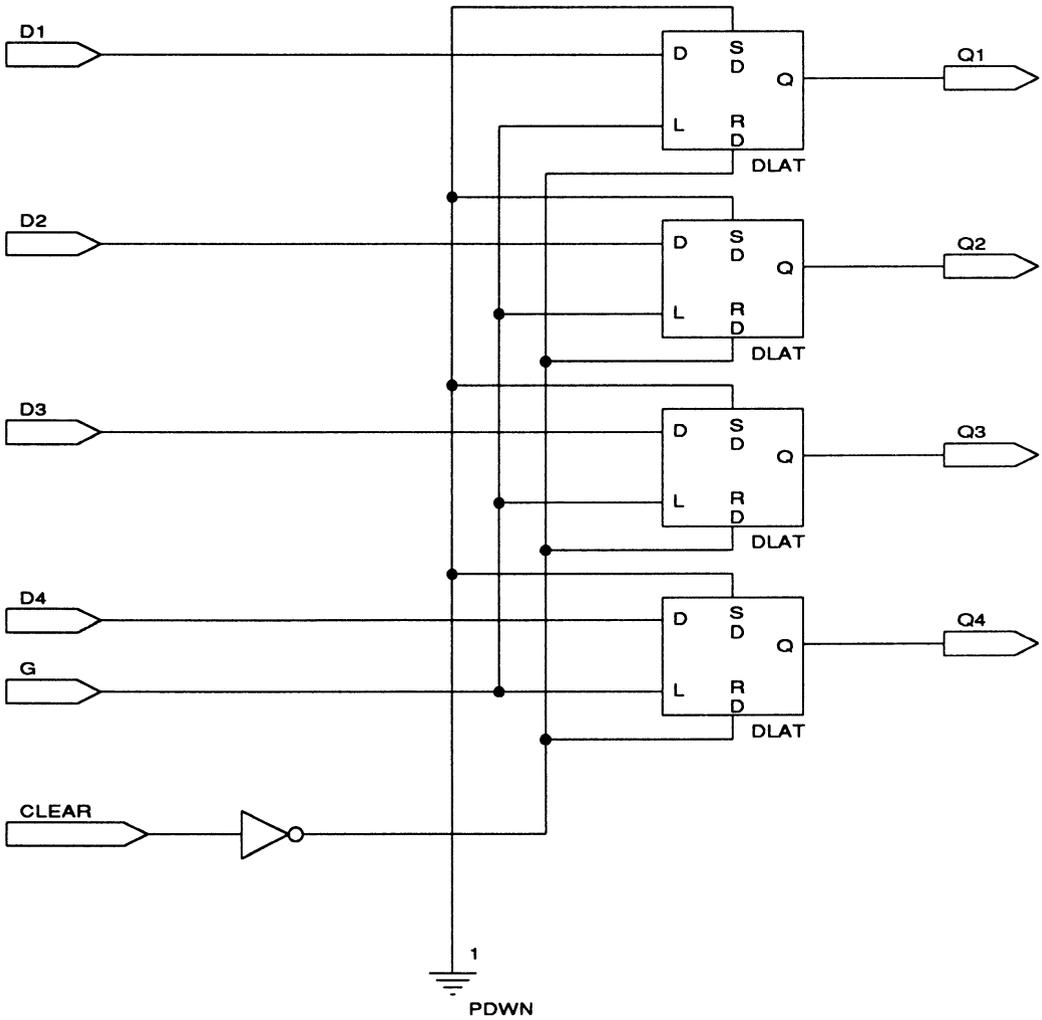
$$Q3 = ((Q3 * CLEAR * /G) + (CLEAR * GND) + (CLEAR * G * D3) + (D3 * CLEAR * Q3))$$

$$Q4 = ((Q4 * CLEAR * /G) + (CLEAR * GND) + (CLEAR * G * D4) + (D4 * CLEAR * Q4))$$

Function Table

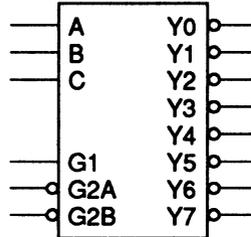
Inputs			Outputs
CLR	G	D	Q
H	L	X	L
L	H	H	H
L	H	L	L
L	L	X	Qo*

* Qo = previous state of Q



- Active LOW outputs
- Three enable inputs

Logic Symbol



Macrocell count: 8
 Array inputs: 6
 Product terms used: 8
 Product terms allocated: 32

Functional Description

The 74138 macro decodes a 3-bit binary input into a single active-LOW output.

You can cascade these macros to implement a decoder with up to 24 outputs via the three enable inputs, G1, G2A, and G2B.

Sample PDS Equivalent

$$Y7 = ((G1 * /G2A * /G2B) * (C * B * A))$$

$$Y6 = ((G1 * /G2A * /G2B) * (C * B * /A))$$

$$Y5 = ((G1 * /G2A * /G2B) * (C * /B * A))$$

$$Y4 = ((G1 * /G2A * /G2B) * (C * /B * /A))$$

$$Y3 = ((G1 * /G2A * /G2B) * (/C * B * A))$$

$$Y2 = ((G1 * /G2A * /G2B) * (/C * B * /A))$$

$$Y1 = ((G1 * /G2A * /G2B) * (/C * /B * A))$$

$$Y0 = ((G1 * /G2A * /G2B) * (/C * /B * /A))$$

Function Table

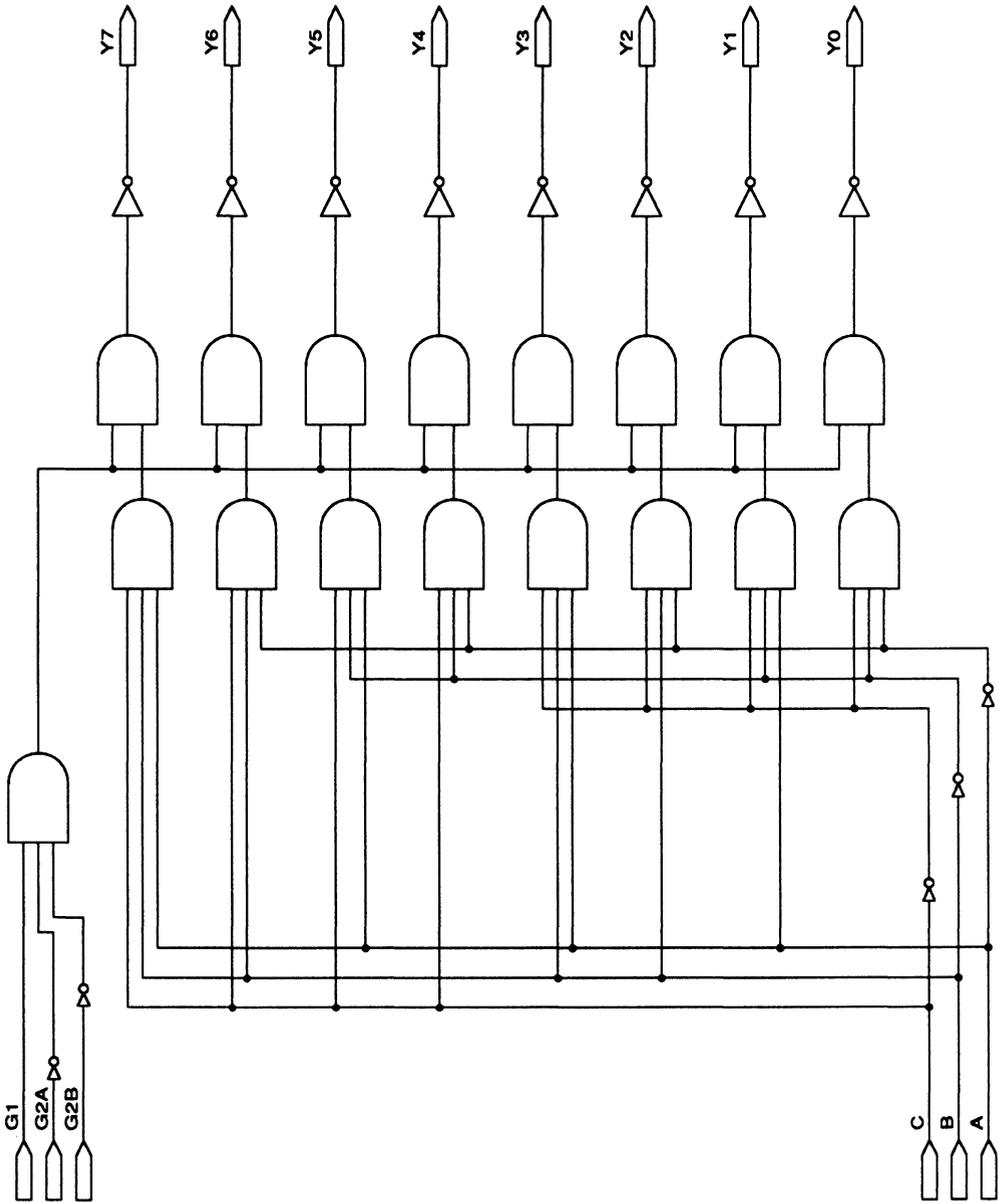
Inputs					Outputs							
Enable		Select										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

* G2 = G2A + G2B

74138

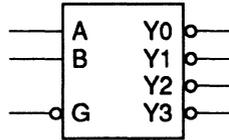
3-to-8 Line Decoder

74138



- Enable input

Logic Symbol



Macrocell count: 4
 Array inputs: 3
 Product terms used: 4
 Product terms allocated: 16

Functional Description

The 74139 macro decodes one of four active-LOW outputs depending on two data inputs. The active-LOW enable input, G, can be used as an input when decoding more output lines.

PDS Equivalent

$$Y3 = \overline{(\overline{G} * B * A)}$$

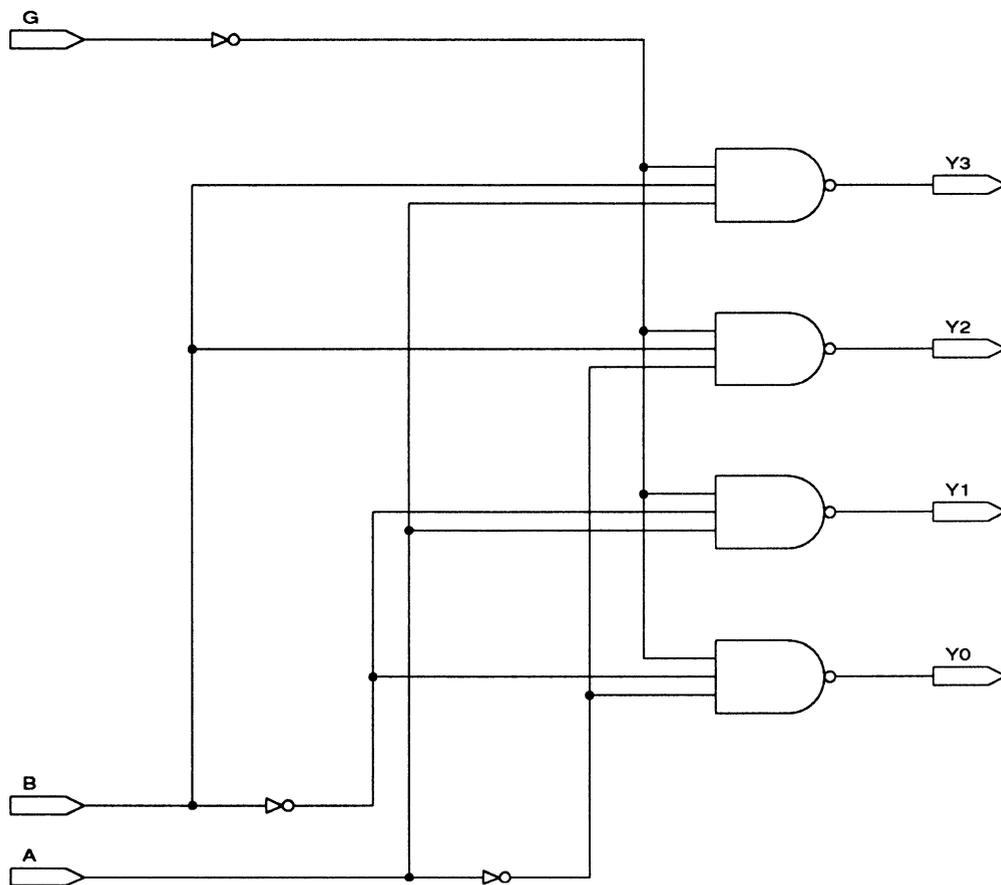
$$Y2 = \overline{(\overline{G} * B * \overline{A})}$$

$$Y1 = \overline{(\overline{G} * \overline{B} * A)}$$

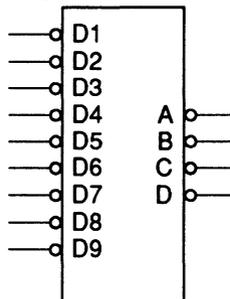
$$Y0 = \overline{(\overline{G} * \overline{B} * \overline{A})}$$

Function Table

Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L



Logic Symbol



Macrocell count: 4
 Array inputs: 9
 Product terms used: 13
 Product terms allocated: 20

Functional Description

The 74147 macro generates a 4-bit BCD-output code that represents the highest-order-LOW data input. Priority encoding of the inputs ensures that only the highest-order data-input line is encoded.

Sample PDS Equivalent

$$A = ((/D1 * D2 * D4 * D6 * (/D8 + /D9)) + (/D3 * D4 * D6 * (/D8 + /D9)) + (/D5 * D6 * (/D8 + /D9)) + (/D7 * (/D8 + /D9)) + /D9)$$

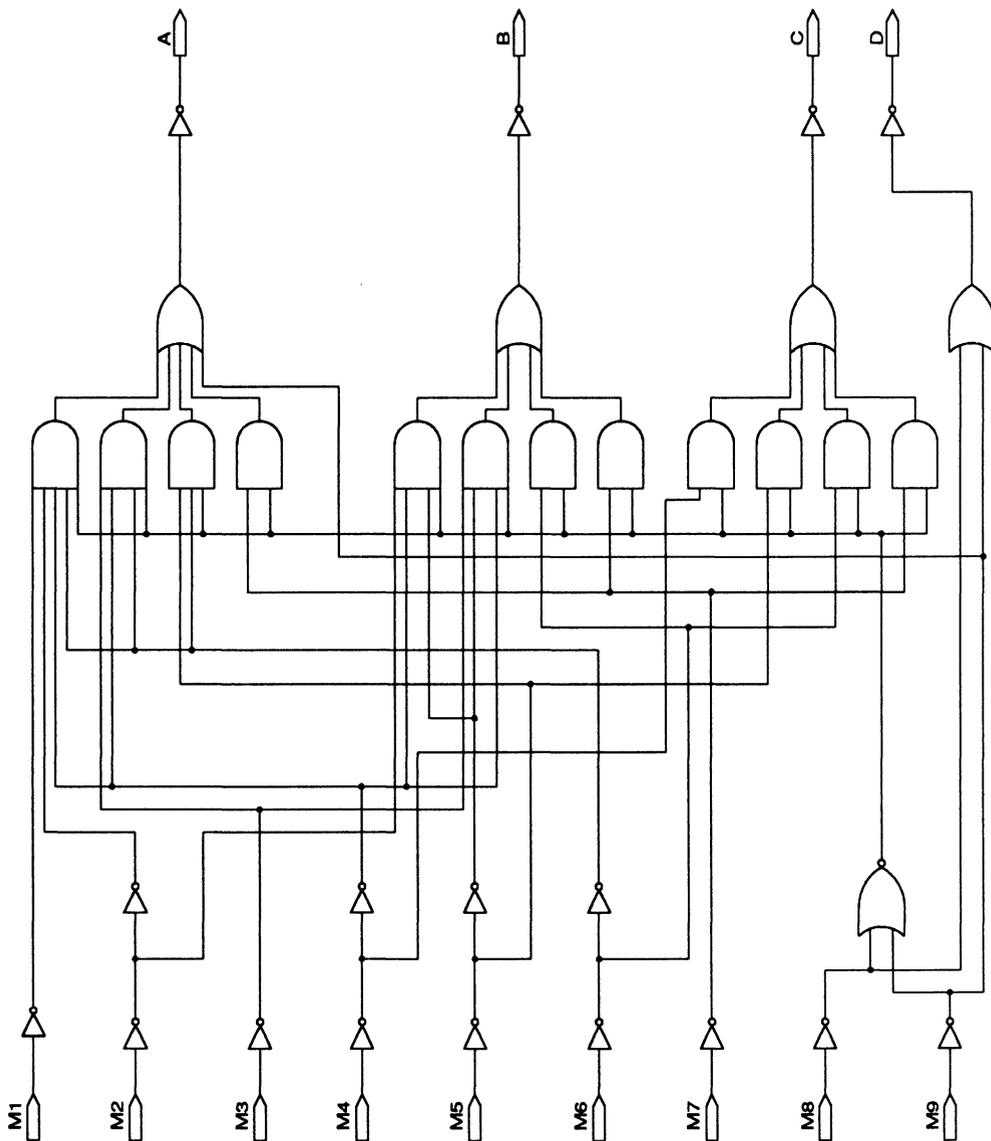
$$B = ((/D2 * D4 * D5 * (/D8 + /D9)) + (/D3 * D5 * D4 * (/D8 + /D9)) + (/D6 * (/D8 + /D9)) + (/D7 * (/D8 + /D9)))$$

$$C = ((/D4 * (/D8 + /D9)) + (/D5 * (/D8 + /D9)) + (/D6 * (/D8 + /D9)) + (/D7 * (/D8 + /D9)))$$

$$D = (/D8 + /D9)$$

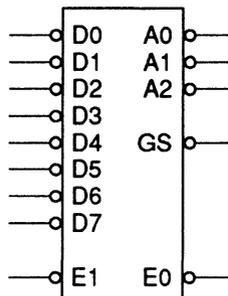
Function Table

Inputs									Outputs			
D1	D2	D3	D4	D5	D6	D7	D8	D9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L



- Enable input and output for cascading

Logic Symbol



Macrocell count: 5
 Array inputs: 10
 Product terms used: 12
 Product terms allocated: 20

Functional Description

The 74148 macro generates a 3-bit binary output code that represents the highest-order-LOW data input. You can use the input enable, EI, and output enable, EO, to expand priority encoding.

PDS Equivalent

$$EO = (D0 * D1 * D2 * D3 * D4 * D5 * D6 * /EI * D7)$$

$$GS = (EO + EI)$$

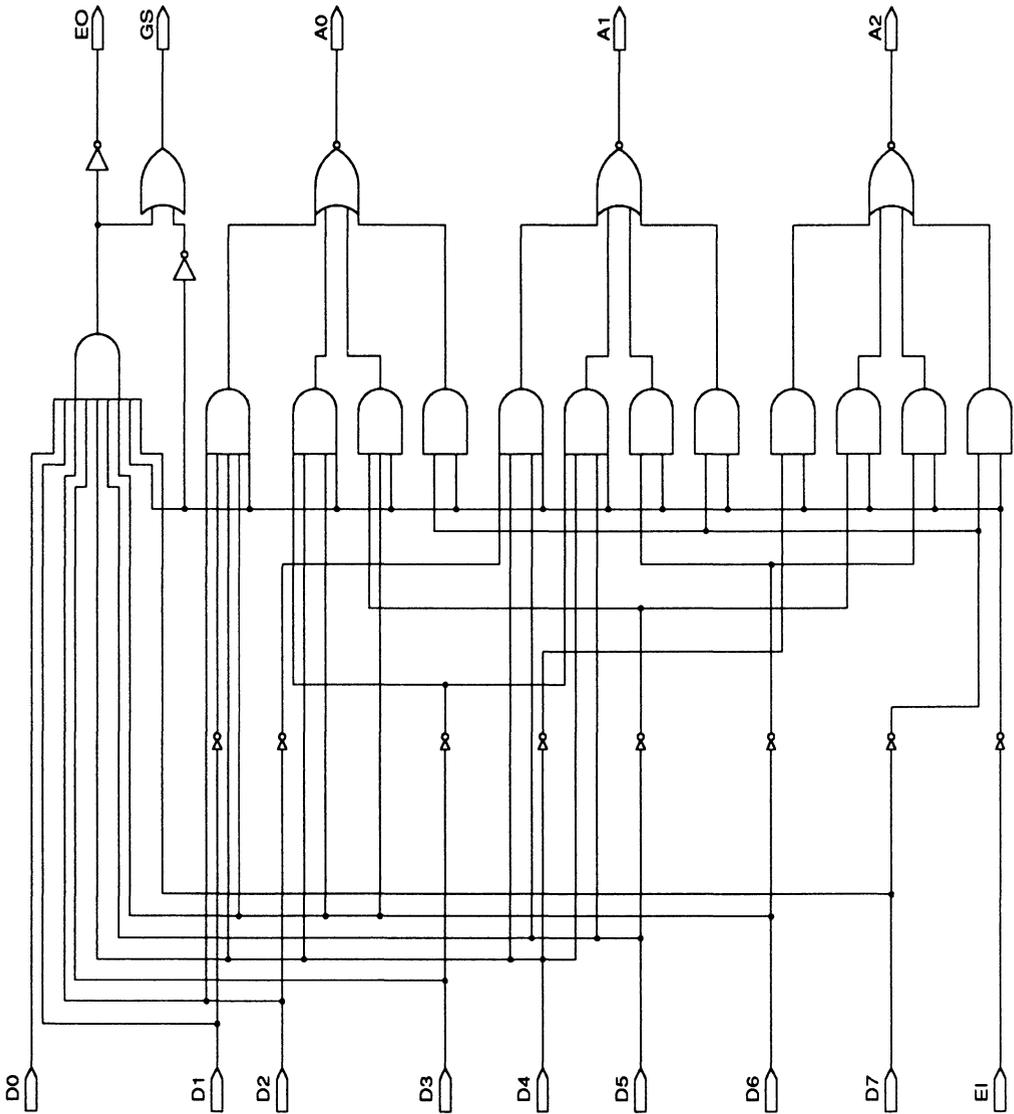
$$A0 = /((/D2 * /D1 * D4 * D6 * /EI) + (/D3 * D4 * D6 * /EI) + (/D5 * D6 * /EI) + (/D7 * /EI))$$

$$A1 = /((/D2 * D4 * D5 * /EI) + (/D3 * D4 * D5 * /EI) + (/D6 * /EI) + (/D7 * /EI))$$

$$A2 = /((/D4 * /EI) + (/D5 * /EI) + (/D6 * /EI) + (/D7 * /EI))$$

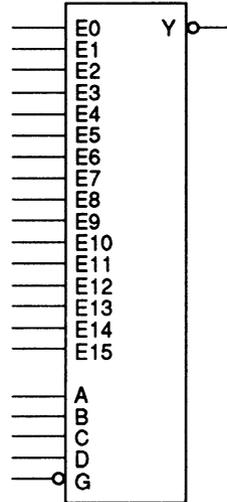
Function Table

Inputs									Outputs				
E1	D0	D1	D2	D3	D4	D5	D6	D7	A2	A1	A0	GS	E1
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H



- Enable input
- Inverted outputs

Logic Symbol



Macrocell count: 3
 Array inputs: 23
 Product terms used: 17
 Product terms allocated: 20

Functional Description

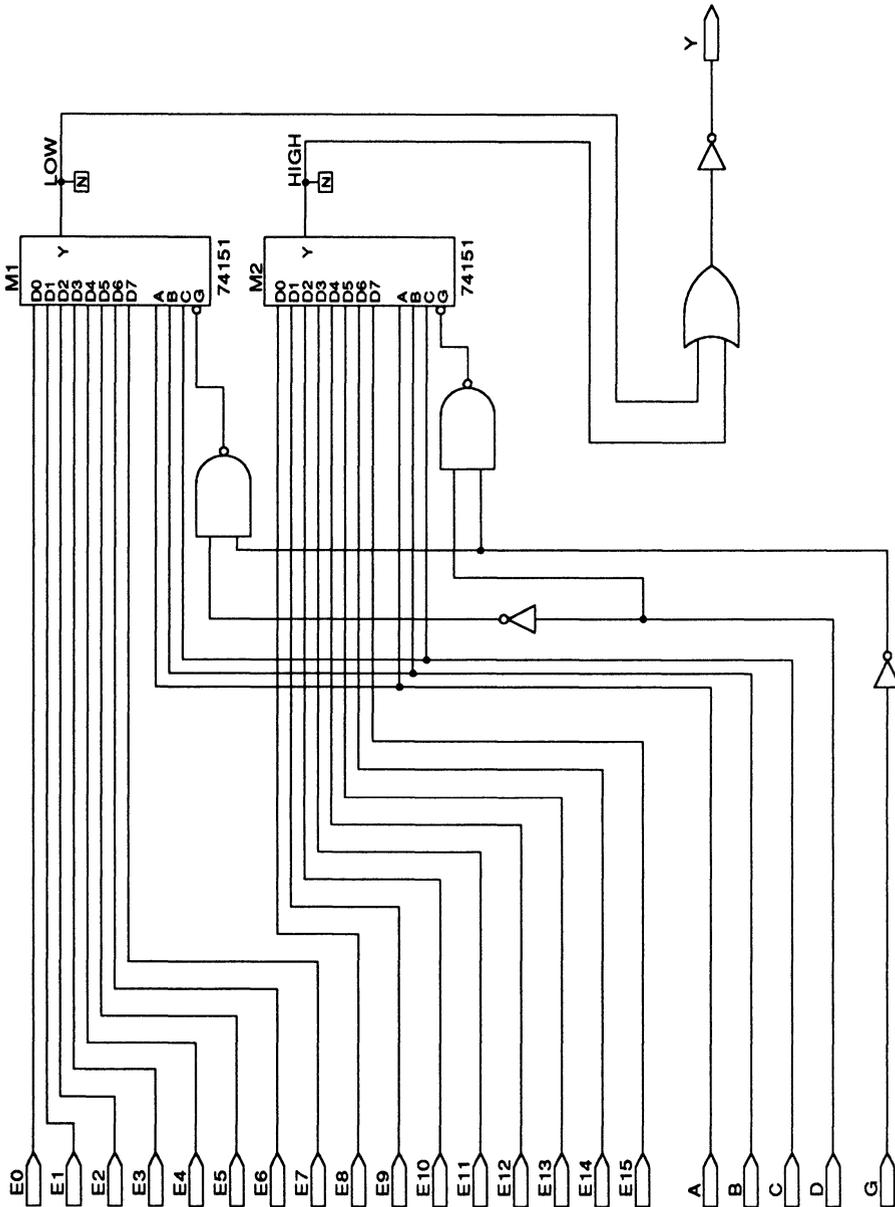
The 74150 macro decodes four data-input lines to select one of 16 data sources. The enable input, G, must be LOW to enable the Y output.

Sample PDS Equivalent

$$\begin{aligned}
 \text{LOW} &= ((E7 * A * B * C * (/D * /G)) + (E6 * /A * B * C * (/D * /G)) + (E5 * A * /B * C * (/D * /G)) + (E4 * /A * /B * C * (/D * /G)) + (E3 * A * B * /C * (/D * /G)) + (E2 * /A * B * /C * (/D * /G)) + (E1 * A * /B * /C * (/D * /G)) + (E0 * /A * /B * /C * (/D * /G))) \\
 \text{HIGH} &= ((E15 * A * B * C * (D * /G)) + (E14 * /A * B * C * (D * /G)) + (E13 * A * /B * C * (D * /G)) + (E12 * /A * /B * C * (D * /G)) + (E11 * A * B * /C * (D * /G)) + (E10 * /A * B * /C * (D * /G)) + (E9 * A * /B * /C * (D * /G)) + (E8 * /A * /B * /C * (D * /G))) \\
 \text{Y} &= (\text{LOW} + \text{HIGH})
 \end{aligned}$$

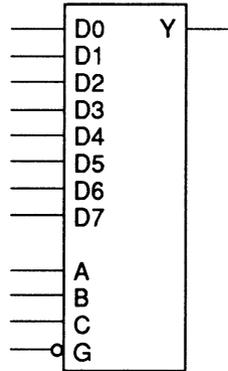
Function Table

Inputs					Outputs
Select				Strobe	
D	C	B	A	G	Y
X	X	X	X	H	H
L	L	L	L	L	E0
L	L	L	H	L	E1
L	L	H	L	L	E2
L	L	H	H	L	E3
L	H	L	L	L	E4
L	H	L	H	L	E5
L	H	H	L	L	E6
L	H	H	H	L	E7
H	L	L	L	L	E8
H	L	L	H	L	E9
H	L	H	L	L	E10
H	L	H	H	L	E11
H	H	L	L	L	E12
H	H	L	H	L	E13
H	H	H	L	L	E14
H	H	H	H	L	E15



- Enable input

Logic Symbol



Macrocell count: 1
 Array inputs: 12
 Product terms used: 8
 Product terms allocated: 8

Functional Description

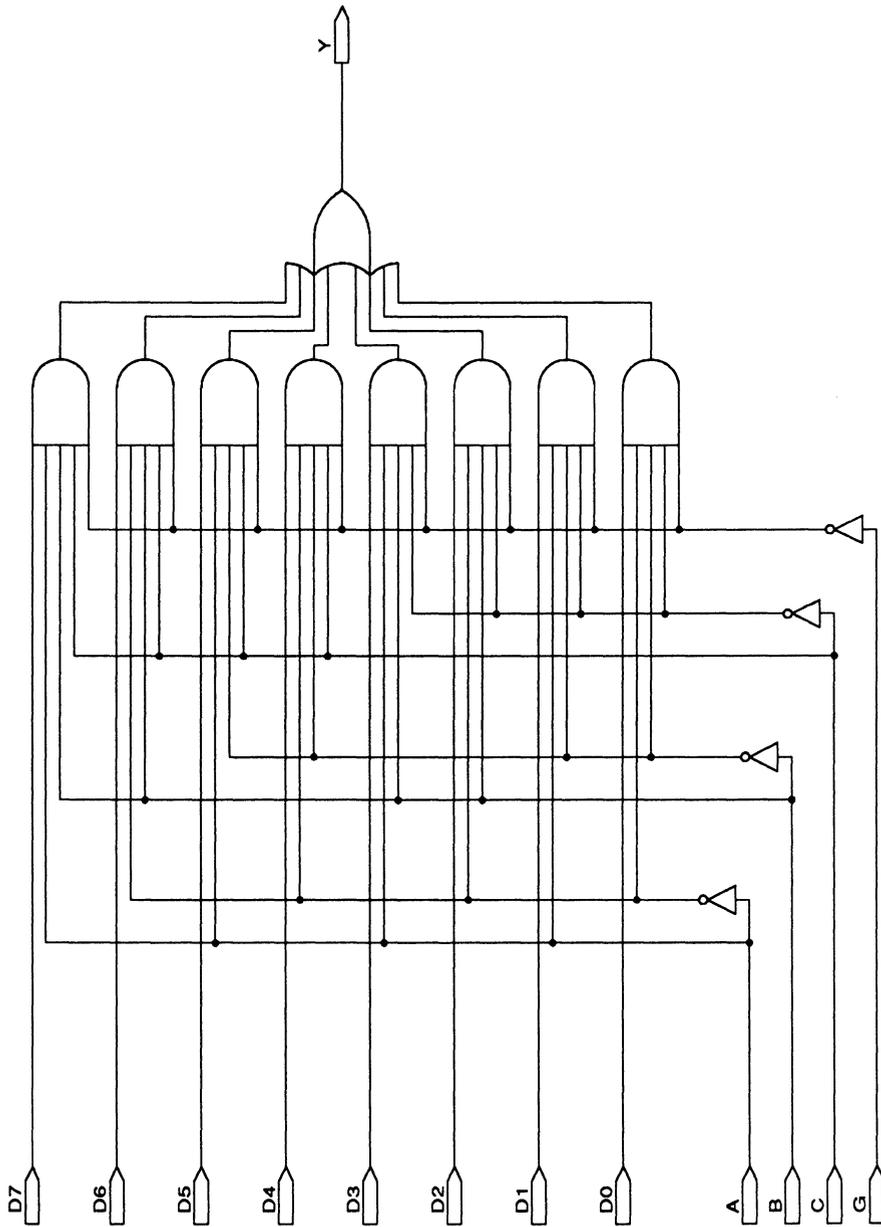
The 74151 macro decodes three data-input lines to select one of eight data sources. The enable input, G, must be LOW to enable the Y output.

Sample PDS Equivalent

$$Y = ((D7 * A * B * C / G) + (D6 * /A * B * C / G) + (D5 * A * /B * C / G) + (D4 * /A * /B * C / G) + (D3 * A * B * /C / G) + (D2 * /A * B * /C / G) + (D1 * A * /B * /C / G) + (D0 * /A * /B * /C / G))$$

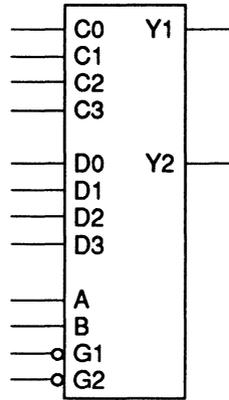
Function Table

Inputs				Outputs
Select			Strobe	
C	B	A	G	Y
X	X	X	H	L
L	L	L	L	D0
L	L	H	L	D1
L	H	L	L	D2
L	H	H	L	D3
H	L	L	L	D4
H	L	H	L	D5
H	H	L	L	D6
H	H	H	L	D7



- Individual enable inputs
- Common data-select inputs

Logic Symbol



Macrocell count: 2
 Array inputs: 12
 Product terms used: 8
 Product terms allocated: 8

Functional Description

The 74153 macro consists of two 4-to-1 multiplexers with common data-select lines. Each 4-to-1 multiplexer has an active LOW strobe input line to enable the output.

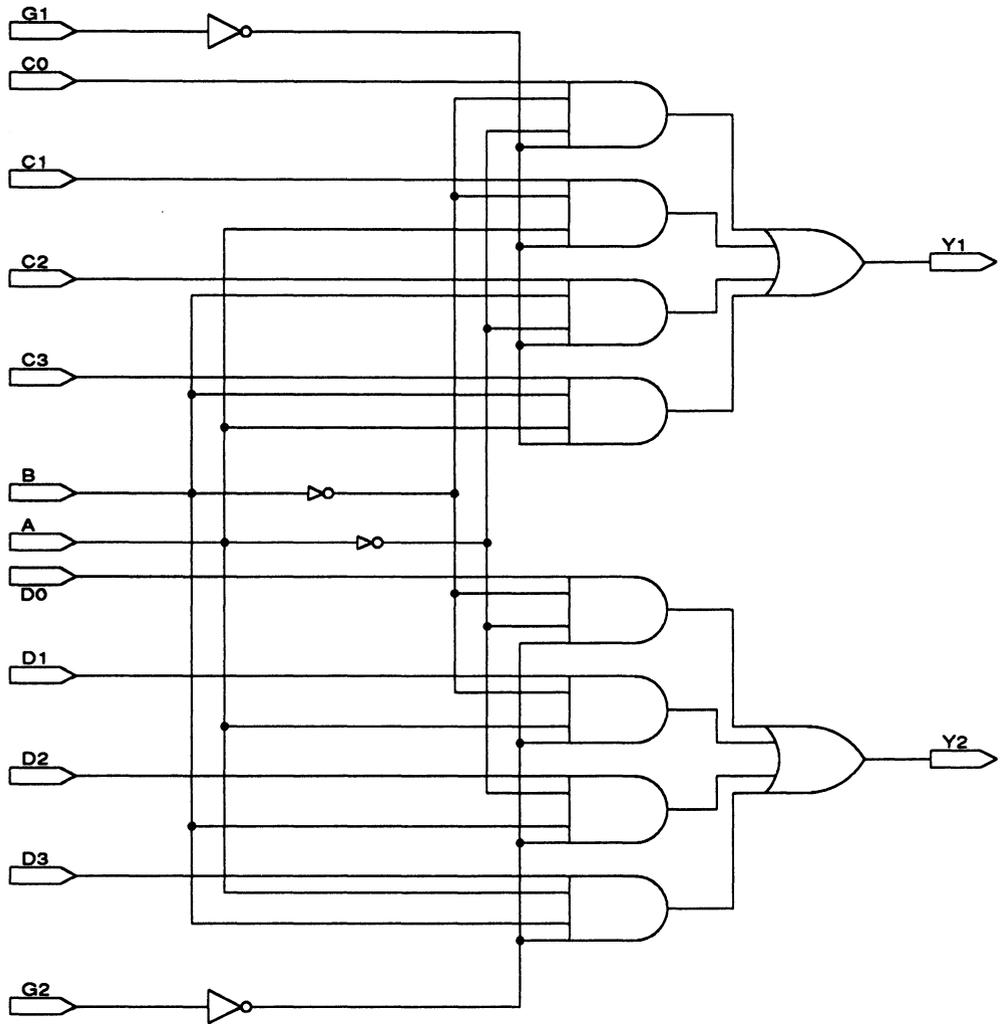
Sample PDS Equivalent

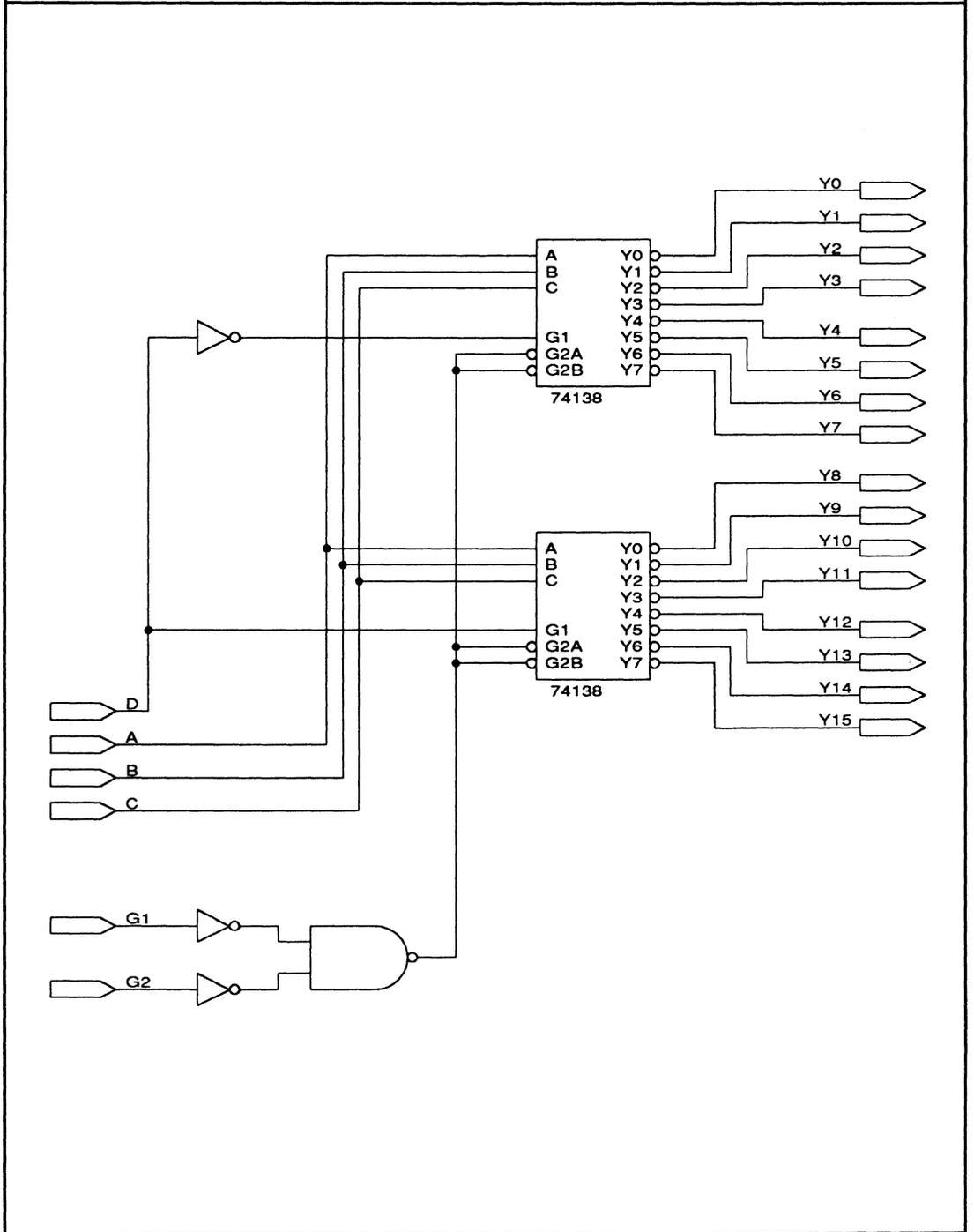
$$Y1 = ((C0 * /B * /A * /G1) + (C1 * /B * A * /G1) + (C2 * B * /A * /G1) + (C3 * B * A * /G1))$$

$$Y2 = ((D0 * /B * /A * /G2) + (D1 * /B * A * /G2) + (D2 * /A * B * /G2) + (D3 * A * B * /G2))$$

Function Table

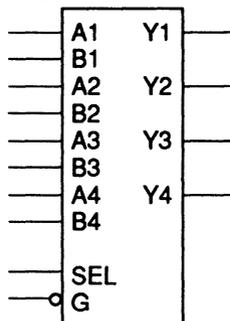
		Inputs					Outputs
Select		Data				Strobe	
B	A	C0	C1	C2	C3	G	Y
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H





- Enable input

Logic Symbol



Macrocell count: 4
 Array inputs: 10
 Product terms used: 8
 Product terms allocated: 16

Functional Description

The 74157 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced LOW regardless of the inputs.

Sample PDS Equivalent

$$Y1 = ((A1 * (/G * /SEL)) + (B1 * (/G * SEL)))$$

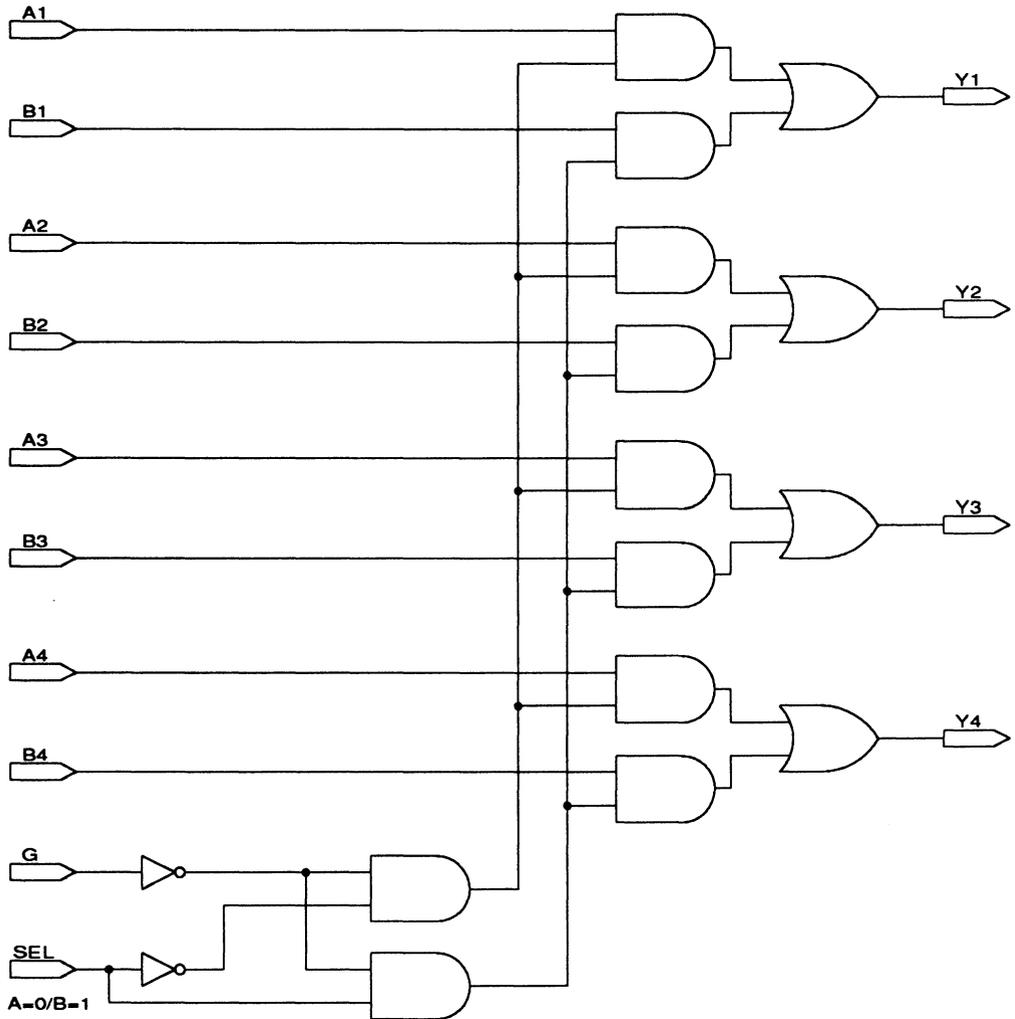
$$Y2 = ((A2 * (/G * /SEL)) + (B2 * (/G * SEL)))$$

$$Y3 = ((A3 * (/G * /SEL)) + (B3 * (/G * SEL)))$$

$$Y4 = ((A4 * (/G * /SEL)) + (B4 * (/G * SEL)))$$

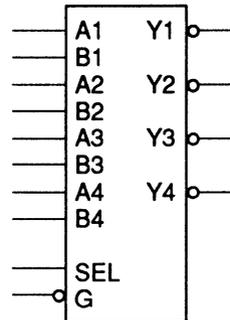
Function Table

Inputs		Outputs			
G	SEL	Y1	Y2	Y3	Y4
H	X	L	L	L	L
L	L	A1	A2	A3	A4
L	H	B1	B2	B3	B4



- Enable input
- Inverted outputs

Logic Symbol



Macrocell count: 4
 Array inputs: 10
 Product terms used: 8
 Product terms allocated: 16

Functional Description

The 74158 macro selects one of two 4-bit words based on the level of the select line, SEL. The enable input, G, must be LOW to enable the output lines. When G is HIGH, all the outputs are forced HIGH regardless of the inputs.

Sample PDS Equivalent

$$Y1 = \overline{((A1 * \overline{/G * /SEL}) + (B1 * (/G * SEL)))}$$

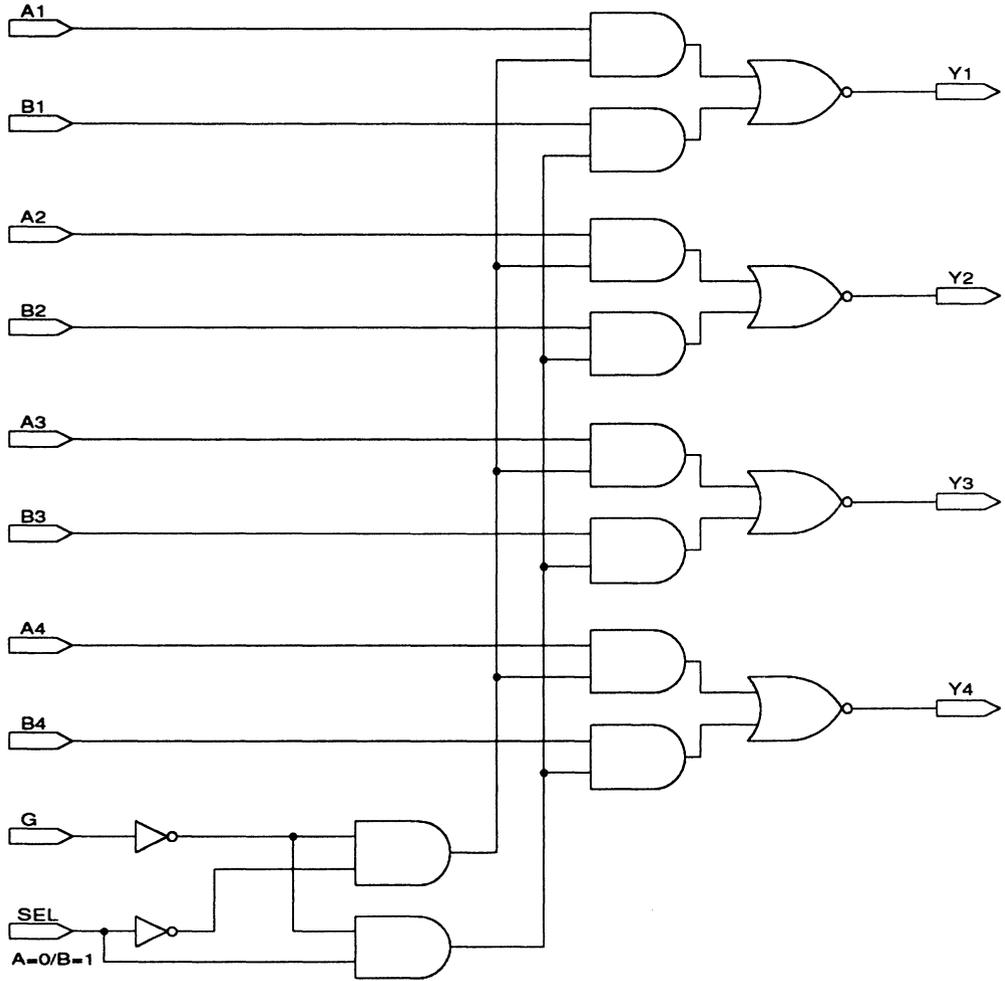
$$Y2 = \overline{((A2 * \overline{/G * /SEL}) + (B2 * (/G * SEL)))}$$

$$Y3 = \overline{((A3 * \overline{/G * /SEL}) + (B3 * (/G * SEL)))}$$

$$Y4 = \overline{((A4 * \overline{/G * /SEL}) + (B4 * (/G * SEL)))}$$

Function Table

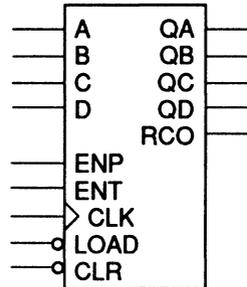
Inputs		Outputs			
G	SEL	Y1	Y2	Y3	Y4
H	X	H	H	H	H
L	L	/A1	/A2	/A3	/A4
L	H	/B1	/B2	/B3	/B4



74162 4-Bit BCD/Decade Counter w/ Synchronous Reset 74162

- Synchronous load
- Synchronous reset
- Carry look-ahead output
- Two enable inputs

Logic Symbol



Macrocell count: 5
 Array inputs: 12
 Product terms used: 18
 Product terms allocated: 24

Functional Description

The 74162 macro is a 4-bit BCD loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count, 9, has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

Sample PDS Equivalent

$$QA.T = ((CLR * QA) + (CLR * /LOAD * /A * QA) + (CLR * /LOAD * /QA * A) + (CLR * ENP * LOAD * ENT))$$

$$QA.clkf = CLK$$

$$QB.T = ((CLR * QB) + ((CLR * ENP * LOAD * ENT) * QA * /QD) + (CLR * /LOAD * /B * QB) + (CLR * /LOAD * /QB * B))$$

$$QB.clkf = CLK$$

$$QC.T = ((CLR * QC) + (CLR * /LOAD * /C * QC) + (CLR * /LOAD * /QC * C) + (CLR * QB * QA * (CLR * ENP * LOAD * ENT)))$$

$$QC.clkf = CLK$$

$$QD.T = ((CLR * QD) + ((CLR * ENP * LOAD * ENT) * QA * QD) + ((CLR * ENP * LOAD * ENT) * QC * QB * QA) + (CLR * /LOAD * /D * QD) + (CLR * /LOAD * /QD * D))$$

$$QD.clkf = CLK$$

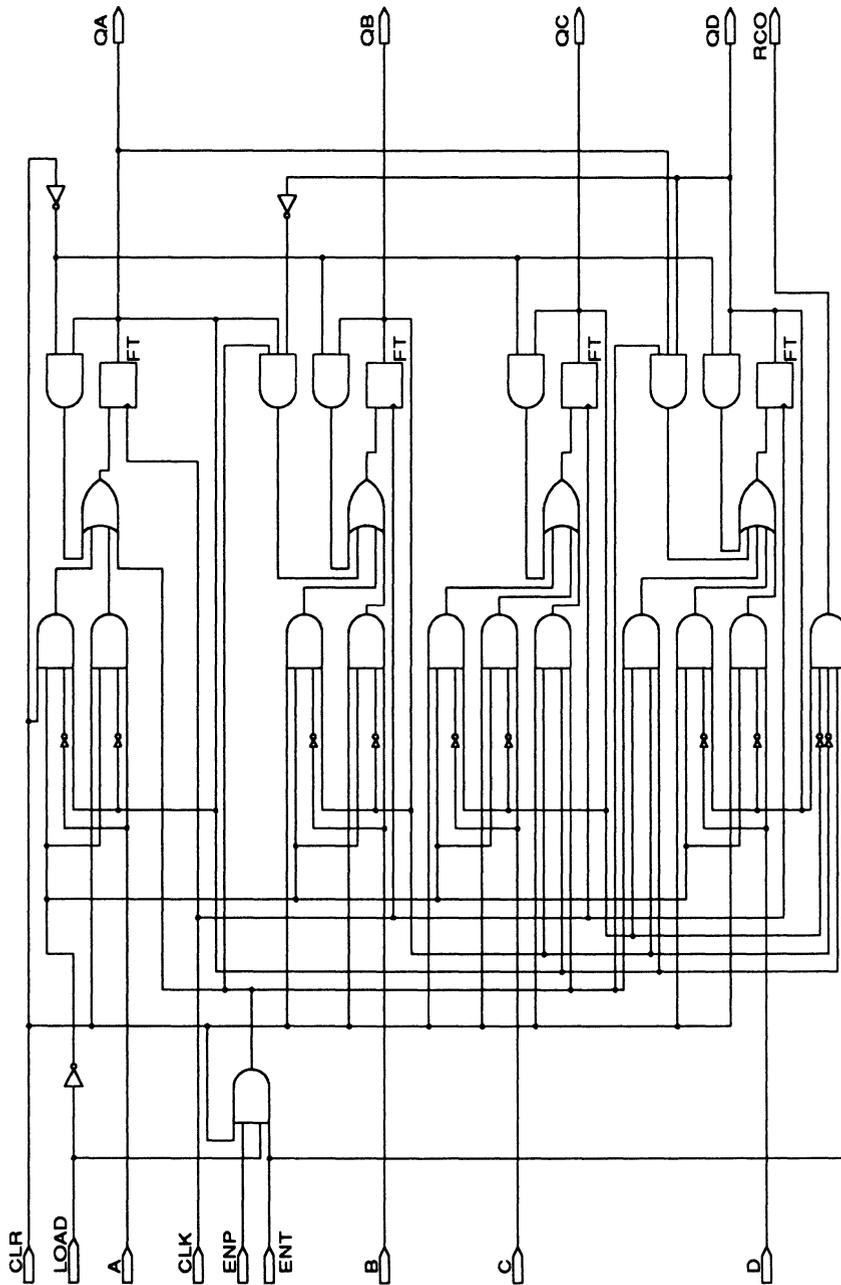
$$RCO = (QD * /QC * /QB * QA * ENT)$$

Function Table

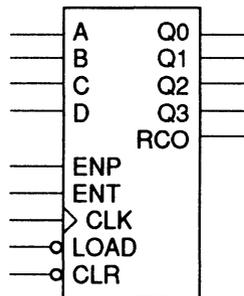
	Inputs					Outputs			
Mode	CLK	CLR	LOAD	ENP	ENT	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 9 and ENT is HIGH. Otherwise, it stays LOW.

74162 4-Bit BCD/Decade Counter w/ Synchronous Reset 74162



- Synchronous 4-bit binary-loadable up counter
- Synchronous reset
- Carry look-ahead output for making wider counters

Logic Symbol

Macrocell count: 5
 Array inputs: 12
 Product terms used: 17
 Product terms allocated: 20

Functional Description

The 74163 macro is a 4-bit binary-loadable up counter with synchronous reset logic. The enable input lines, ENP and ENT, and the ripple carry-out line, RCO, allow for multiple macros to be cascaded. RCO goes HIGH when the maximum count of 15 has been reached and ENT is HIGH. To enable and increment the counter value, you feed the RCO output to the ENP and ENT inputs of the next counter stage. QD is the most significant counter bit.

Sample PDS Equivalent

$$QA.T = ((/CLR * QA) + (CLR * /LOAD * /A * QA) + (CLR * /LOAD * /QA * A) + (CLR * ENP * LOAD * ENT))$$

$$QA.clkf = CLK$$

$$QB.T = ((/CLR * QB) + (CLR * /LOAD * /B * QB) + (CLR * /LOAD * /QB * B) + (CLR * QA * (CLR * ENP * LOAD * ENT)))$$

$$QB.clkf = CLK$$

$$QC.T = ((/CLR * QC) + (CLR * /LOAD * /C * QC) + (CLR * /LOAD * /QC * C) + (CLR * QB * QA * (CLR * ENP * LOAD * ENT)))$$

$$QC.clkf = CLK$$

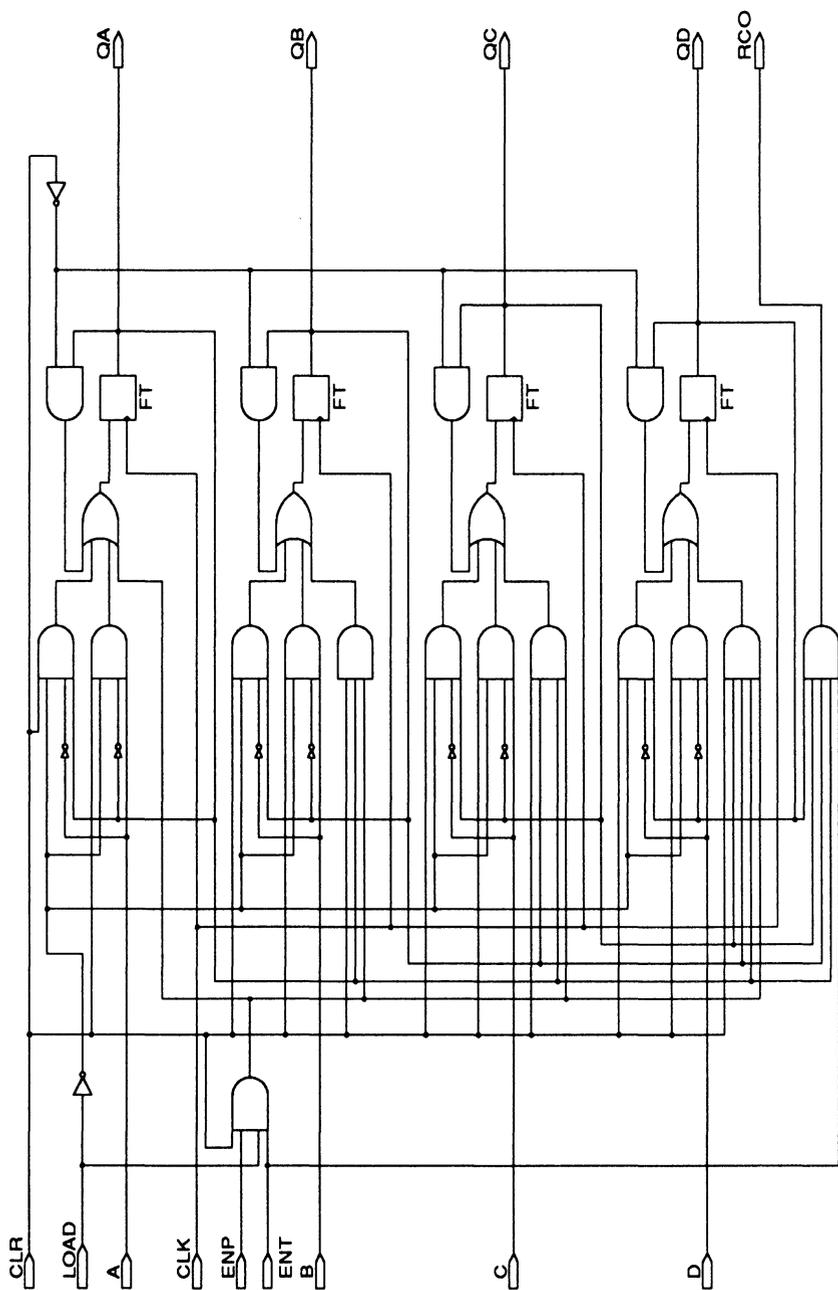
$$QD.T = ((/CLR * QD) + (CLR * /LOAD * /D * QD) + (CLR * /LOAD * /QD * D) + (CLR * QC * QB * QA * (CLR * ENP * LOAD * ENT)))$$

$$QD.clkf = CLK$$

$$RCO = (QD * QC * QB * QA * ENT)$$
Function Table

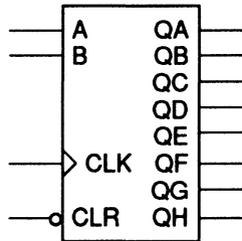
Mode	Inputs					Outputs			
	CLK	CLR	Load	ENP	ENT	QD	QC	QB	QA
Clear	L	L	X	X	X	QD	QC	QB	QA
Clear	↑	L	X	X	X	L	L	L	L
Load	↑	H	L	X	X	D	C	B	A
Count	↑	H	H	H	H	Count Up			
Stop	↑	H	H	L	X	QD	QC	QB	QA
Stop	↑	H	H	X	L	QD	QC	QB	QA

* The RCO is HIGH when the counter output is 15 and ENT is HIGH. Otherwise, it stays LOW.



- Synchronous reset
- ANDed serial inputs

Logic Symbol



Macrocell count: 8
 Array inputs: 10
 Product terms used: 8
 Product terms allocated: 32

Functional Description

The 74164 macro is an 8-bit serial-in parallel-out shift register with synchronous reset. The two serial inputs, A and B, are logically ANDed.

Note:
 The TTL version has asynchronous reset logic.

Sample PDS Equivalent

QA = (B * A * CLR)
 QA.clkf = CLK
 QB = (CLR * QA)
 QB.clkf = CLK
 QC = (CLR * QB)
 QC.clkf = CLK
 QD = (CLR * QC)
 QD.clkf = CLK
 QE = (CLR * QD)
 QE.clkf = CLK
 QF = (CLR * QE)
 QF.clkf = CLK
 QG = (CLR * QF)
 QG.clkf = CLK
 QH = (CLR * QG)
 QH.clkf = CLK

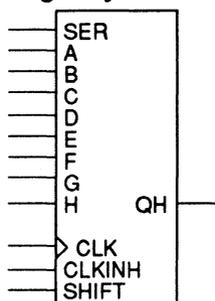
Function Table

Inputs				Outputs							
CLK	CLR	A	B	QA	QB	QC	QD	QE	QF	QG	QH
L	L	X	X	QAo	QBo	QCo	QDo	QEO	QFo	QGo	QHo
↑	L	X	X	L	L	L	L	L	L	L	L
↑	H	H	H	H	QAn	QBn	QCn	QDn	QEn	QFn	QGn
↑	H	L	X	L	QAn	QBn	QCn	QDn	QEn	QFn	QGn
↑	H	X	L	L	QAn	QBn	QCn	QDn	QEn	QFn	QGn

* QAo to QHo = previous state of QA to QH
 QAn to QGn = level of QA to QG before the most recent rising transition of the CLK, and indicates a 1-bit shift.

- Synchronous load
- Shift inhibit

Logic Symbol



Macrocell count: 8
 Array inputs: 19
 Product terms used: 24
 Product terms allocated: 32

Functional Description

The 74165 macro is an 8-bit parallel-in serial-out shift register. To synchronously load the registers, you set the SHIFT input LOW. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values. A parallel-load operation overrides the inhibit function.

Note:

The TTL version has two clock lines and asynchronous load logic.

Function Table

Inputs				Outputs / Internal Registers															
SHIFT	INH	CLK	SER	A	B	C	D	E	F	G	H	QA	QB	QC	QD	QE	QF	QG	QH
L	X	↑	X	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h
L	X	L	X	X	X	X	X	X	X	X	X	QA _o	QB _o	QC _o	QD _o	QE _o	QF _o	QG _o	QH _o
H	L	L	X	X	X	X	X	X	X	X	X	QA _o	QB _o	QC _o	QD _o	QE _o	QF _o	QG _o	QH _o
H	L	↑	L	X	X	X	X	X	X	X	X	L	QA _o	QB _o	QC _o	QD _o	QE _o	QF _o	QG _o
H	L	↑	H	X	X	X	X	X	X	X	X	H	QA _o	QB _o	QC _o	QD _o	QE _o	QF _o	QG _o
H	H	↑	X	X	X	X	X	X	X	X	X	QA _o	QB _o	QC _o	QD _o	QE _o	QF _o	QG _o	QH _o

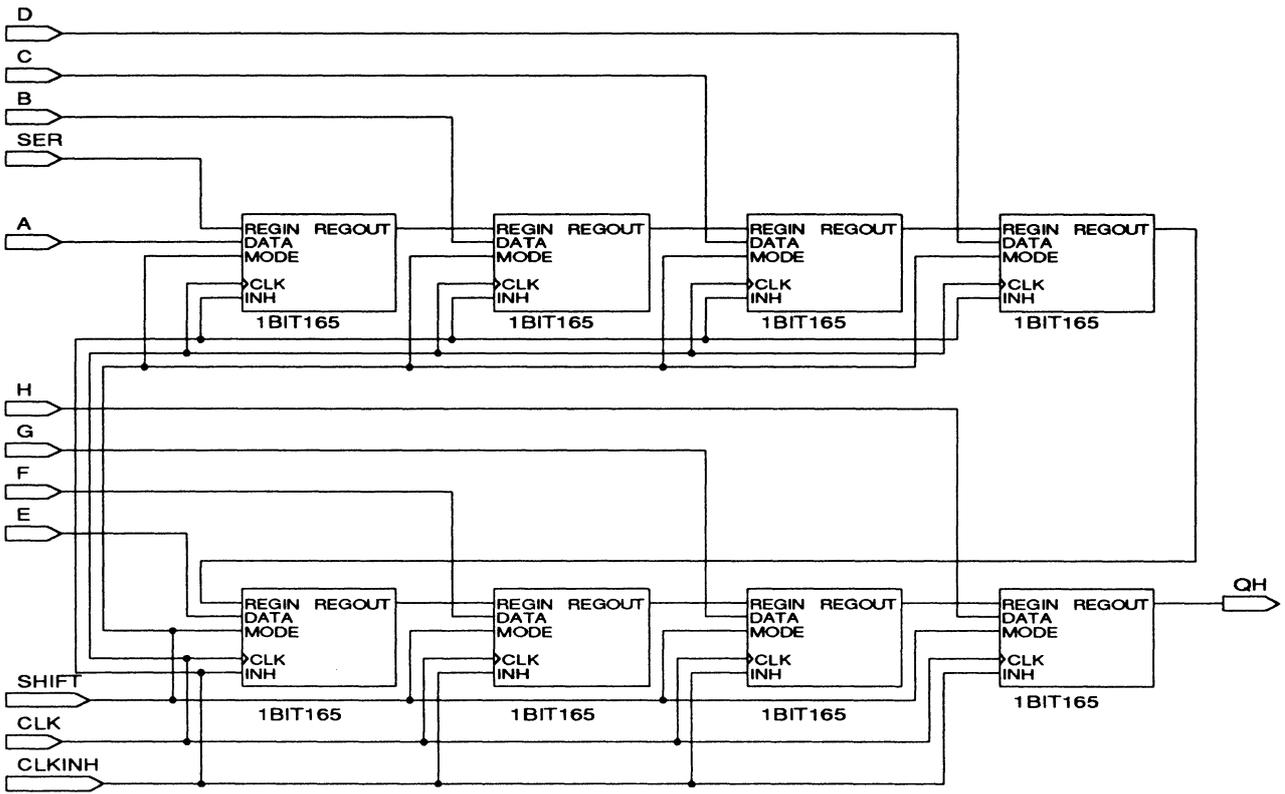
* QA_o to QH_o = previous state of registers QA to QH

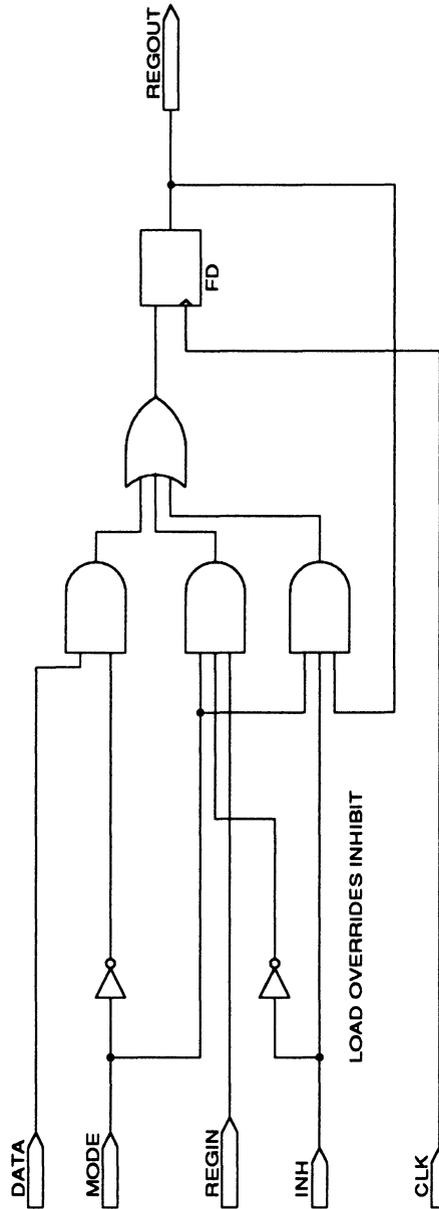
Sample PDS Equivalent

```

M1_REGOUT = ((A */SHIFT) + (SHIFT */CLKINH * SER) + (SHIFT * CLKINH * M1_REGOUT))
M1_REGOUT.clkf = CLK
M2_REGOUT = ((B */SHIFT) + (SHIFT */CLKINH * M1_REGOUT) + (SHIFT * CLKINH * M2_REGOUT))
M2_REGOUT.clkf = CLK
M3_REGOUT = ((C */SHIFT) + (SHIFT */CLKINH * M2_REGOUT) + (SHIFT * CLKINH * M3_REGOUT))
M3_REGOUT.clkf = CLK
M4_REGOUT = ((D */SHIFT) + (SHIFT */CLKINH * M3_REGOUT) + (SHIFT * CLKINH * M4_REGOUT))
M4_REGOUT.clkf = CLK
M8_REGOUT = ((E */SHIFT) + (SHIFT */CLKINH * M4_REGOUT) + (SHIFT * CLKINH * M8_REGOUT))
M8_REGOUT.clkf = CLK
M5_REGOUT = ((F */SHIFT) + (SHIFT */CLKINH * M8_REGOUT) + (SHIFT * CLKINH * M5_REGOUT))
M5_REGOUT.clkf = CLK
M6_REGOUT = ((G */SHIFT) + (SHIFT */CLKINH * M5_REGOUT) + (SHIFT * CLKINH * M6_REGOUT))
M6_REGOUT.clkf = CLK
QH = ((H */SHIFT) + (SHIFT */CLKINH * M6_REGOUT) + (SHIFT * CLKINH * QH))
QH.clkf = CLK

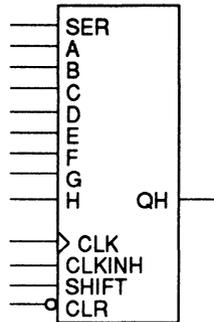
```





- Parallel synchronous load
- Synchronous reset

Logic Symbol



Macrocell count: 8
 Array inputs: 20
 Product terms used: 24
 Product terms allocated: 32

Functional Description

The 74166 macro is an 8-bit parallel-in serial-out shift register with synchronous reset. To load the registers, you set the SHIFT input LOW and apply a rising-edge clock. Setting the inhibit input, CLKINH, HIGH inhibits shifting and the registers retain their current values.

Note:

The TTL version has asynchronous reset logic.

Function Table

Inputs					Internal Outputs								Output									
SHIFT	INH	CLK	CLR	SER	A	B	C	D	E	F	G	H	QAi	QBi	QCi	QDi	QEi	QFi	QGi	QHi	QH	
X	X	L	L	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEO	QFo	QGo	QHo	QHo	
X	X	↑	L	X	X	X	X	X	X	X	X	X	L	L	L	L	L	L	L	L	L	L
X	X	L	X	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEO	QFo	QGo	QHo	QHo	
L	L	↑	H	X	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h	h	
H	L	↑	H	L	X	X	X	X	X	X	X	X	L	QAn	QBn	QCn	QDn	QEn	QFn	QGn	QGn	
H	L	↑	H	H	X	X	X	X	X	X	X	X	H	QAn	QBn	QCn	QDn	QEn	QFn	QGn	QGn	
X	H	↑	H	X	X	X	X	X	X	X	X	X	QAo	QBo	QCo	QDo	QEO	QFo	QGo	QHo	QHo	

* QAo to QHo = previous state of QA to QH

QAn to QGn = level of QA to QG before the most recent rising transition of the CLK, and indicates a 1-bit shift.

Sample PDS Equivalent

$$ND_A = ((CLR * SER * /CLKINH * SHIFT) + (CLR * A * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_A))$$

$$ND_A.clkf = CLK$$

$$ND_B = ((CLR * ND_A * /CLKINH * SHIFT) + (CLR * B * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_B))$$

$$ND_B.clkf = CLK$$

$$ND_C = ((CLR * ND_B * /CLKINH * SHIFT) + (CLR * C * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_C))$$

$$ND_C.clkf = CLK$$

$$ND_D = ((CLR * ND_C * /CLKINH * SHIFT) + (CLR * D * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_D))$$

$$ND_D.clkf = CLK$$

$$ND_E = ((CLR * ND_D * /CLKINH * SHIFT) + (CLR * E * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_E))$$

$$ND_E.clkf = CLK$$

$$ND_F = ((CLR * ND_E * /CLKINH * SHIFT) + (CLR * F * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_F))$$

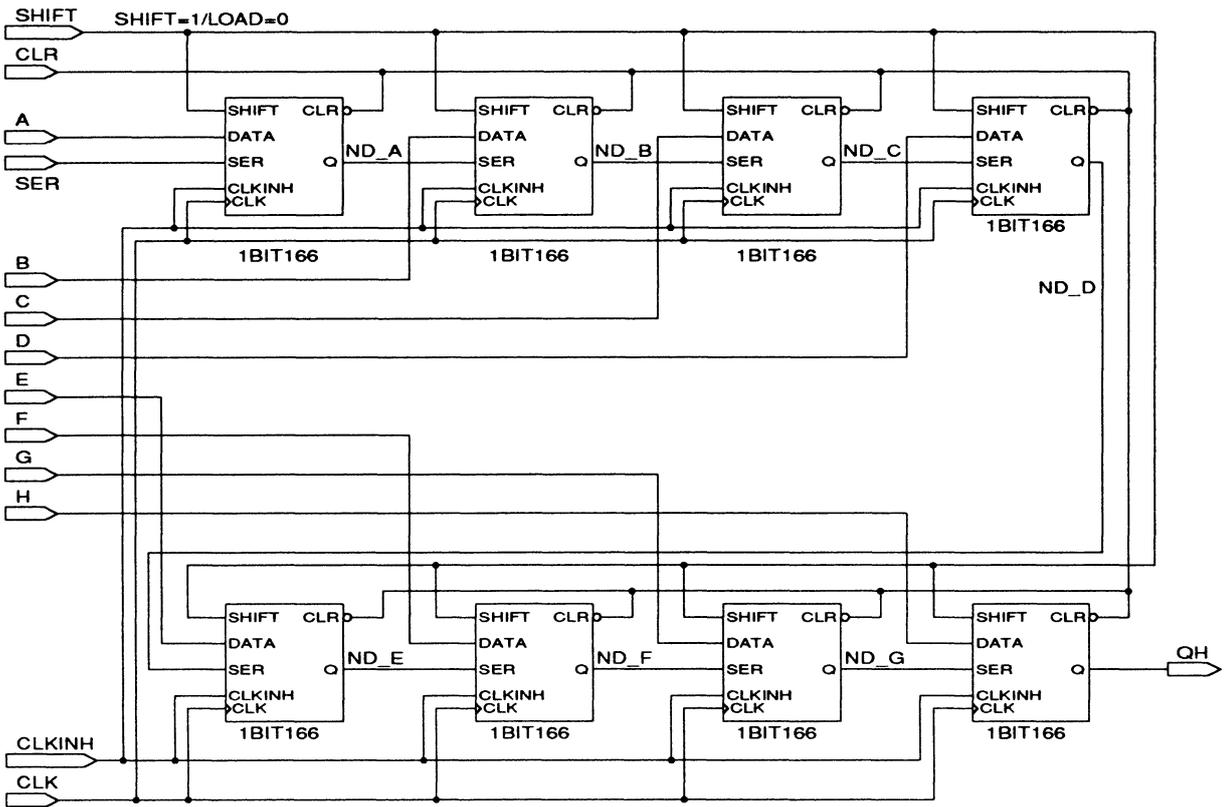
$$ND_F.clkf = CLK$$

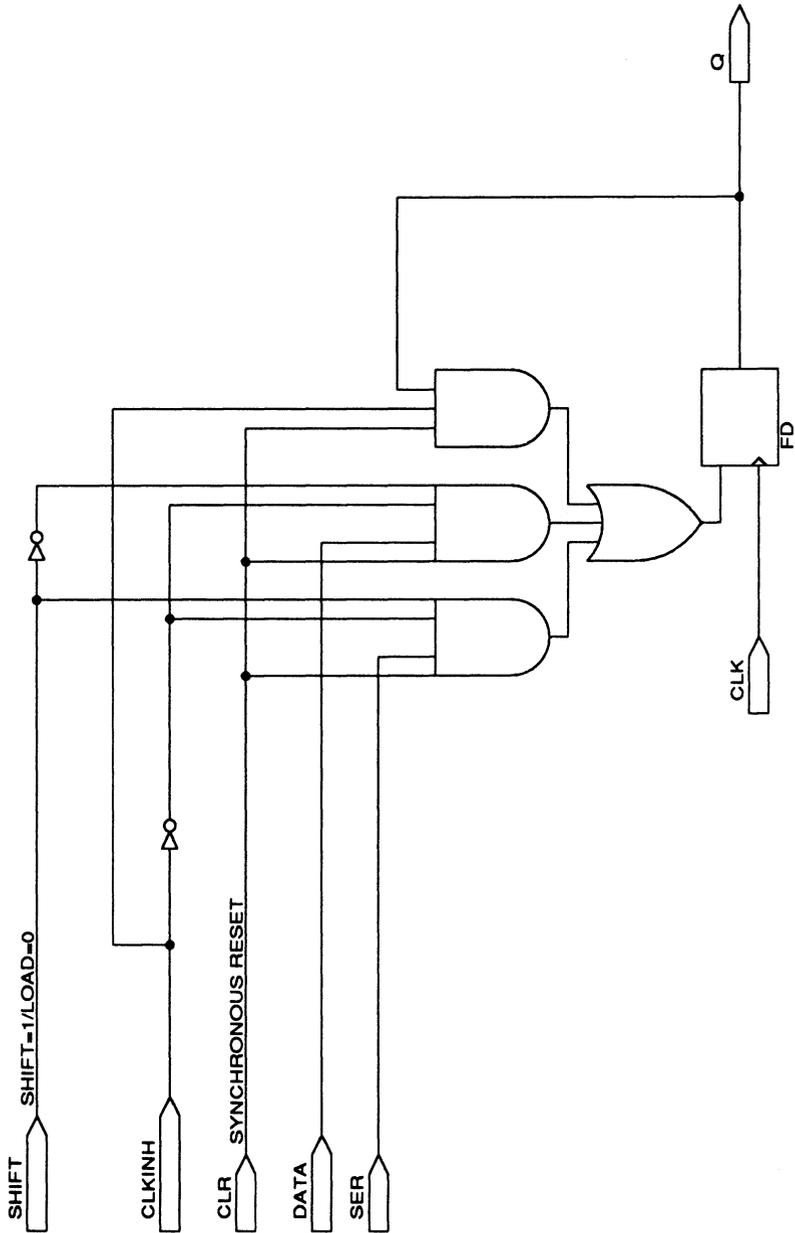
$$ND_G = ((CLR * ND_F * /CLKINH * SHIFT) + (CLR * G * /CLKINH * /SHIFT) + (CLR * CLKINH * ND_G))$$

$$ND_G.clkf = CLK$$

$$QH = ((CLR * ND_G * /CLKINH * SHIFT) + (CLR * H * /CLKINH * /SHIFT) + (CLR * CLKINH * QH))$$

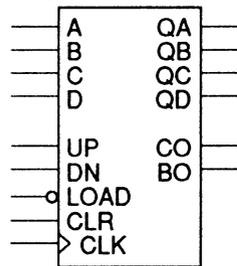
$$QH.clkf = CLK$$





- Synchronous load
- Asynchronous reset
- Carry- and borrow-out signals for expansion

Logic Symbol



Macrocell count: 6
 Array inputs: 12
 Product terms used: 22
 Product terms allocated: 36

Functional Description

The 74192 macro is a 4-bit up/down BCD counter with synchronous parallel load and asynchronous reset logic. You can select an increasing or decreasing count sequence by setting either the UP or DN control input HIGH. An active-LOW borrow signal, BO, is generated when the count is zero and DN is HIGH. An active-LOW carry signal, CO, is generated when the count is 9 and UP is HIGH.

Note:

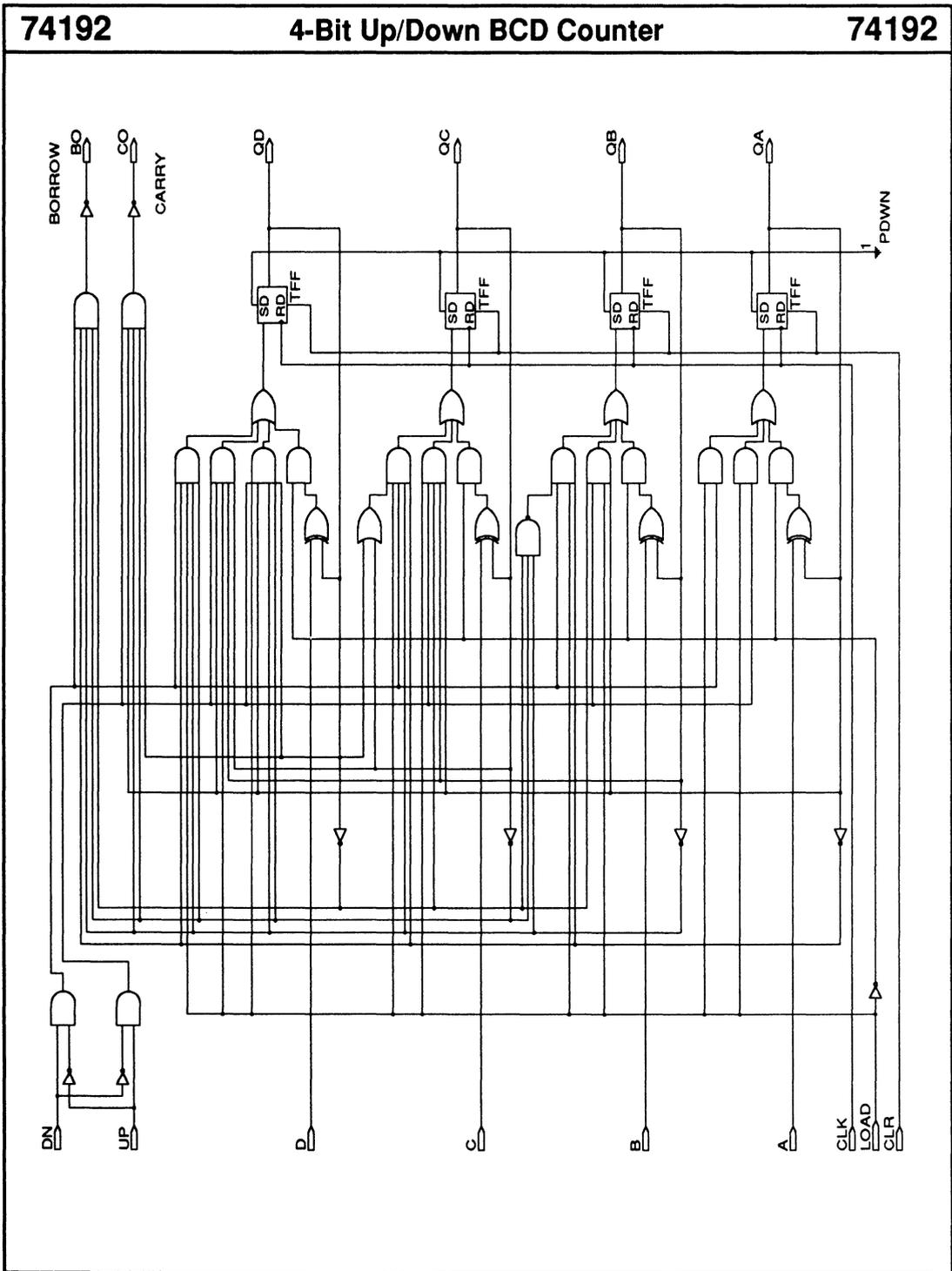
The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

Function Table

Inputs					Outputs									
CLK	CLR	LOAD	UP	DN	A	B	C	D	QA	QB	QC	QD	BO	CO
L	H	X	X	X	X	X	X	X	L	L	L	L	L	L
↑	L	L	X	X	a	b	c	d	a	b	c	d	X	X
↑	L	H	H	L	X	X	X	X	Count Up				H	H
↑	L	H	L	H	X	X	X	X	Count Down				H	H
↑	L	H	H	L	X	X	X	X	H	L	L	H	H	L
↑	L	H	L	H	X	X	X	X	L	L	L	L	L	H
↑	L	H	L	L	X	X	X	X	Hold Count				X	X
↑	L	H	H	H	X	X	X	X	Hold Count				X	X

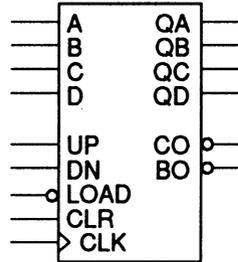
Sample PDS Equivalent

$BO = ((DN \cdot /UP) \cdot /QA \cdot /QB \cdot /QC \cdot /QD)$
 $CO = ((/DN \cdot UP) \cdot QA \cdot /QB \cdot /QC \cdot QD)$
 $QD.T = (((/DN \cdot /UP) \cdot /QA \cdot LOAD \cdot /QB \cdot /QC) + ((/DN \cdot UP) \cdot QA \cdot LOAD \cdot QB \cdot QC) + ((/DN \cdot UP) \cdot LOAD \cdot QA \cdot /QB \cdot /QC \cdot QD) + (/LOAD \cdot (D \cdot : \cdot QD)))$
 $QD.clkf = CLK$
 $QD.rstf = CLR$
 $QC.T = (((QD \cdot QC) \cdot LOAD \cdot (DN \cdot /UP) \cdot /QB \cdot /QA) + (LOAD \cdot (/DN \cdot UP) \cdot /QD \cdot QB \cdot QA) + (/LOAD \cdot (C \cdot : \cdot QC)))$
 $QC.clkf = CLK$
 $QC.rstf = CLR$
 $QB.T = (((/QD \cdot /QC \cdot /QB) \cdot (DN \cdot /UP) \cdot LOAD \cdot /QA) + (/QD \cdot (/DN \cdot UP) \cdot LOAD \cdot QA) + (/LOAD \cdot (B \cdot : \cdot QB)))$
 $QB.clkf = CLK$
 $QB.rstf = CLR$
 $QA.T = ((LOAD \cdot (DN \cdot /UP)) + (LOAD \cdot (/DN \cdot UP)) + (/LOAD \cdot (A \cdot : \cdot QA)))$
 $QA.clkf = CLK$
 $QA.rstf = CLR$



- Synchronous load
- Carry- and borrow-out signals

Logic Symbol



Macrocell count: 6
 Array inputs: 12
 Product terms used: 18
 Product terms allocated: 24

Functional Description

The 74193 macro is a 4-bit up/down binary counter with synchronous load and asynchronous reset logic. You can select an increasing or decreasing count sequence by setting either the UP or DN control input HIGH. An active-LOW borrow signal BO is generated when the count is zero and DN is HIGH. An active-LOW carry signal, CO, is generated when the count is 9 and UP is HIGH. A load operation overrides the count function.

Note:

The TTL version has asynchronous parallel-load logic and uses the UP and DN inputs as two independent clock lines to control the direction of the count sequence.

Function Table

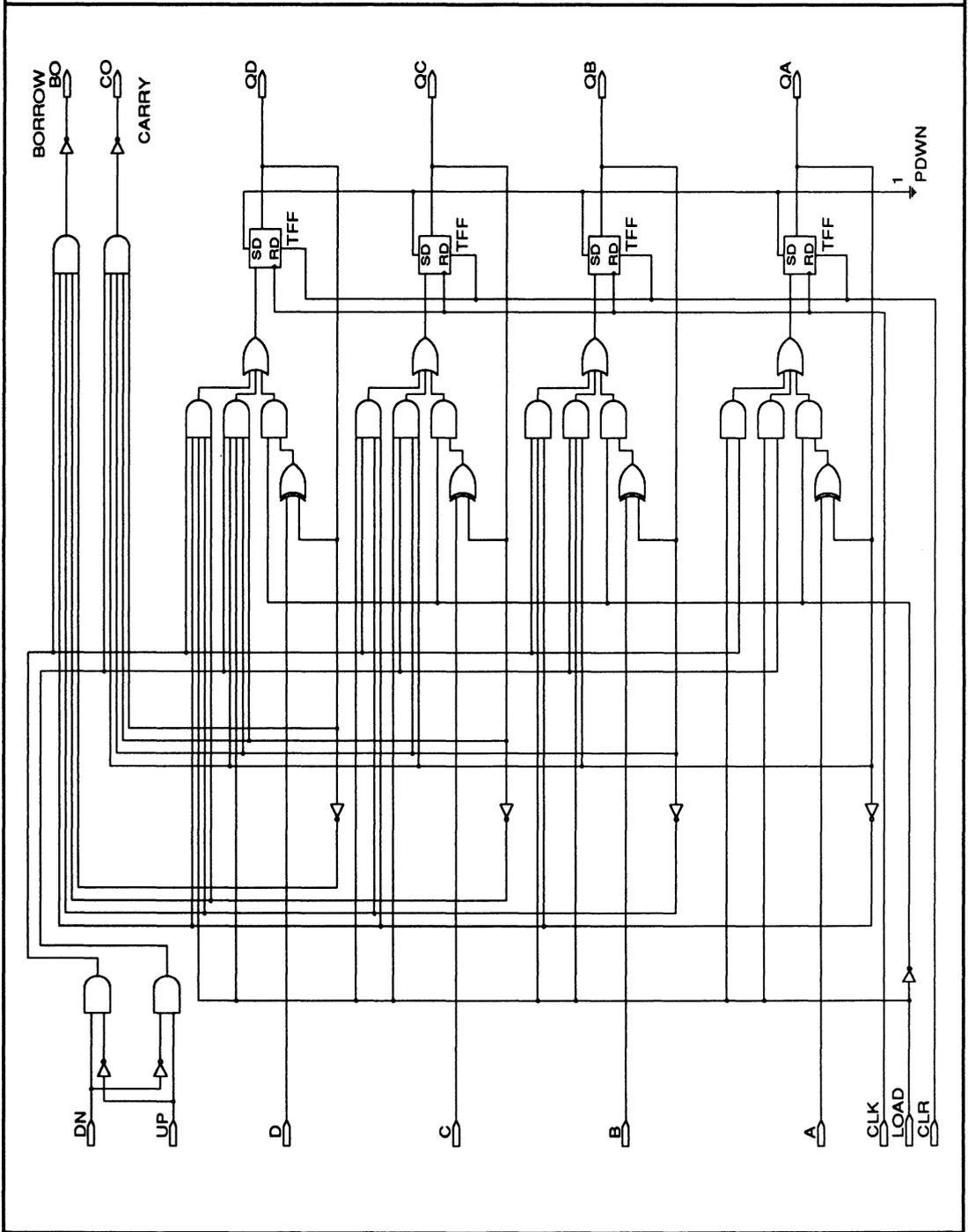
Inputs					Outputs									
CLK	CLR	LOAD	UP	DN	A	B	C	D	QA	QB	QC	QD	BO	CO
L	H	X	X	X	X	X	X	X	L	L	L	L	L	H
↑	L	L	X	X	a	b	c	d	a	b	c	d	X	X
↑	L	H	H	L	X	X	X	X	Count Up				H	H
↑	L	H	L	H	X	X	X	X	Count Down				H	H
↑	L	H	H	L	X	X	X	X	H	H	H	H	H	L
↑	L	H	L	H	X	X	X	X	L	L	L	L	L	H
↑	L	H	L	L	X	X	X	X	Hold Count				X	X
↑	L	H	H	H	X	X	X	X	Hold Count				X	X

Sample PDS Equivalent

```

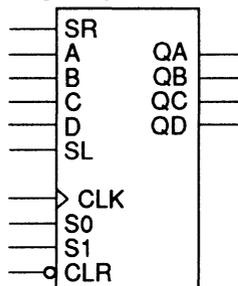
BO = ((DN * /UP) * /QA * /QB * /QC * /QD)
CO = ((/DN * UP) * QA * QB * QC * QD)
QD.T = (((DN * /UP) * /QA * LOAD * /QB * /QC) + ((/DN * UP) * QA * LOAD * QB * QC) + (/LOAD * (D :: QD)))
QD.clkf = CLK
QD.rstf = CLR
QC.T = ((LOAD * (DN * /UP) * /QB * /QA) + (LOAD * (/DN * UP) * QB * QA) + (/LOAD * (C :: QC)))
QC.clkf = CLK
QC.rstf = CLR
QB.T = (((DN * /UP) * LOAD * /QA) + ((/DN * UP) * LOAD * QA) + (/LOAD * (B :: QB)))
QB.clkf = CLK
QB.rstf = CLR
QA.T = ((LOAD * (DN * /UP)) + (LOAD * (/DN * UP)) + (/LOAD * (A :: QA)))
QA.clkf = CLK
QA.rstf = CLR

```



- Parallel-to-serial converter
- Serial-to-parallel converter
- Synchronous reset
- Synchronous loading

Logic Symbol



Macrocell count: 4
 Array inputs: 13
 Product terms used: 16
 Product terms allocated: 16

Functional Description

The 74194 macro is a 4-bit bidirectional universal shift register with synchronous reset logic. Two control lines, S1 and S0, select one of four modes of operation:

- Parallel load of four data inputs
- Right shift (in the direction QA to QD)
- Left shift (in the direction QD to QA)
- Data latch/hold register values

All operations are performed at the rising edge of CLK.

Note:

The TTL version has asynchronous reset logic.

Sample PDS Equivalent

$$QA = ((CLR * SR * /S1 * S0) + (CLR * QB * S1 * /S0) + (CLR * S1 * S0 * A) + (CLR * /S1 * /S0 * QA))$$

$$QA.clkf = CLK$$

$$QB = ((CLR * QA * /S1 * S0) + (CLR * QC * S1 * /S0) + (CLR * S1 * S0 * B) + (CLR * /S1 * /S0 * QB))$$

$$QB.clkf = CLK$$

$$QC = ((CLR * QB * /S1 * S0) + (CLR * QD * S1 * /S0) + (CLR * S1 * S0 * C) + (CLR * /S1 * /S0 * QC))$$

$$QC.clkf = CLK$$

$$QD = ((CLR * QC * /S1 * S0) + (CLR * SL * S1 * /S0) + (CLR * S1 * S0 * D) + (CLR * /S1 * /S0 * QD))$$

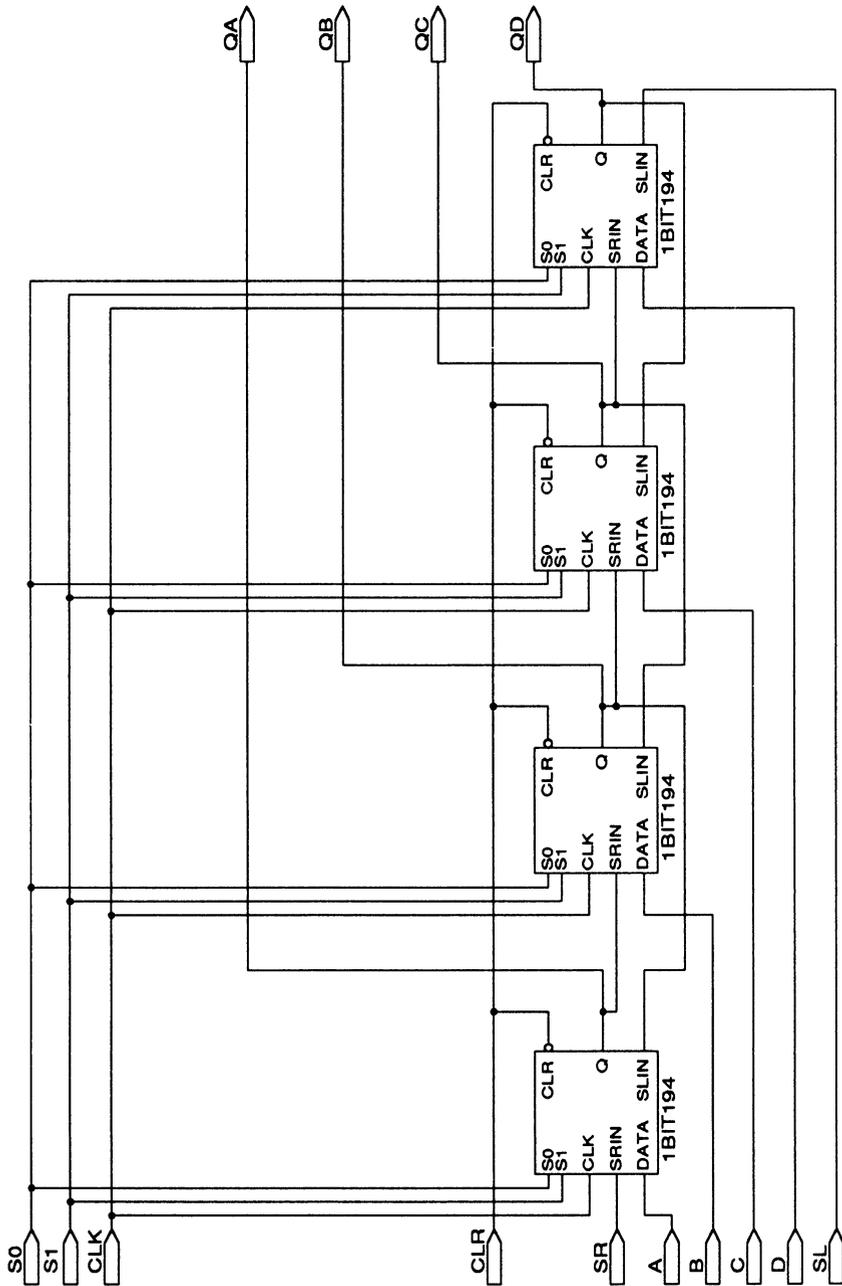
$$QD.clkf = CLK$$

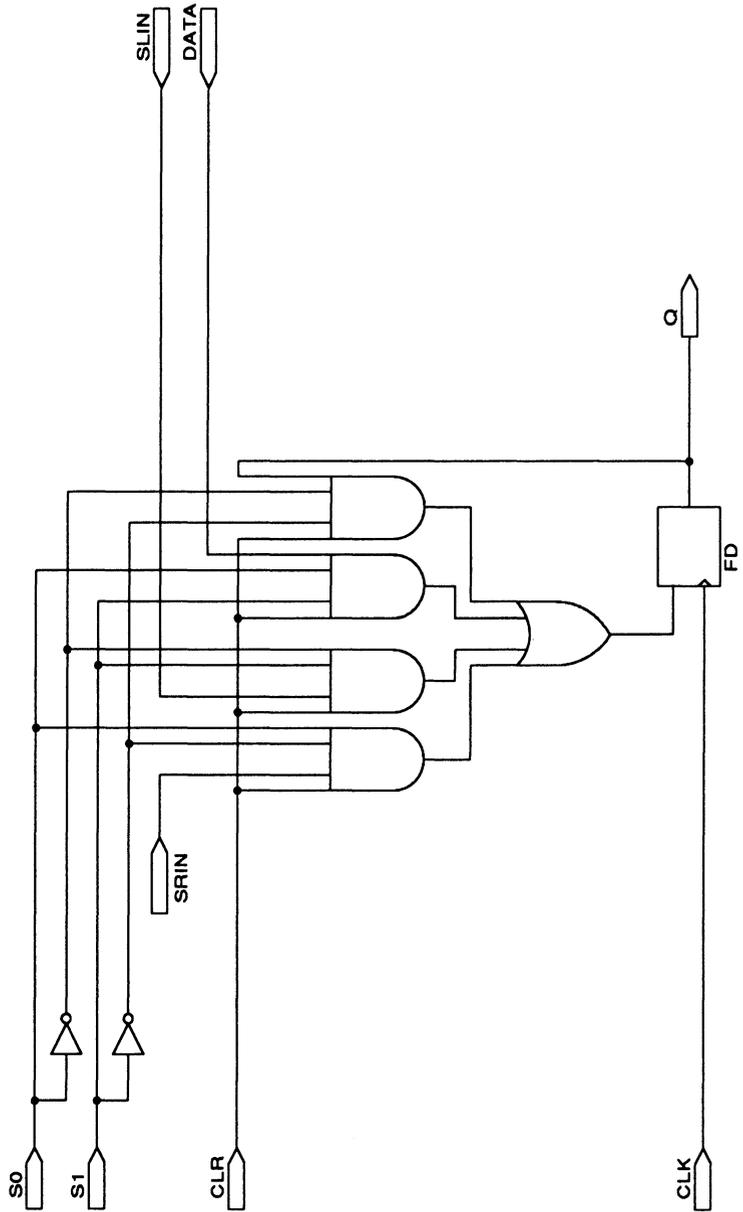
Function Table

		Inputs						Outputs					
Mode		Serial				Parallel							
CLK	CLR	S1	S0	SLSR	A	B	C	D	QA	QB	QC	QD	
L	X	X	X	X	X	X	X	X	QAo	QB0	QC0	QDo	
↑	L	X	X	X	X	X	X	X	L	L	L	L	
X	H	L	L	X	X	X	X	X	QAo	QB0	QC0	QDo	
L	H	L	H	X	L	X	X	X	L	QAn	QBn	QCn	
↑	H	L	H	X	H	X	X	X	H	QAn	QBn	QCn	
↑	H	H	L	L	X	X	X	X	QBn	QCn	QDn	L	
↑	H	H	L	H	X	X	X	X	QBn	QCn	QDn	H	
↑	H	H	H	X	X	a	b	c	d	a	b	c	d

* QAo to QDo = previous state of QA to QD

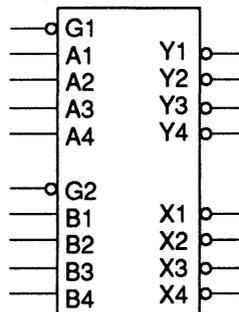
QAn to QDn = level of QA to QD before the most recent rising transition of the CLK, and indicates a 1-bit shift.





- Two enable inputs

Logic Symbol



Macrocell count: 8
 Array inputs: 10
 Product terms used: 8
 Product terms allocated: 32

Functional Description

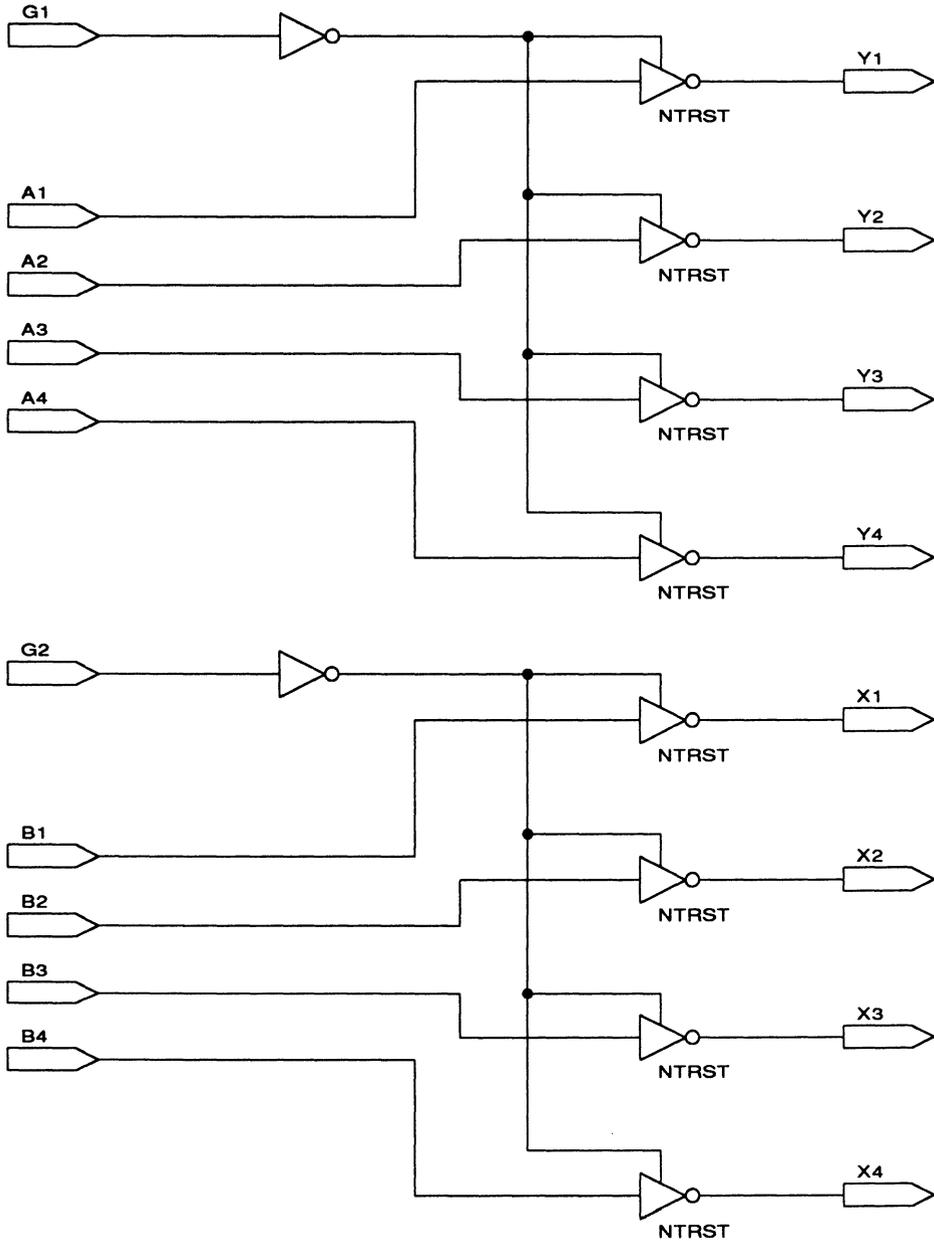
The 74240 macro contains two groups of four inverting buffers. Each group is enabled by an active-LOW input control line.

Sample PDS Equivalent

Y1.trst = /G1
 Y2.trst = /G1
 Y3.trst = /G1
 Y4.trst = /G1
 X1.trst = /G2
 X2.trst = /G2
 X3.trst = /G2
 X4.trst = /G2
 Y1 = A1
 Y2 = A2
 Y3 = A3
 Y4 = A4
 X1 = B1
 X2 = B2
 X3 = B3
 X4 = B4

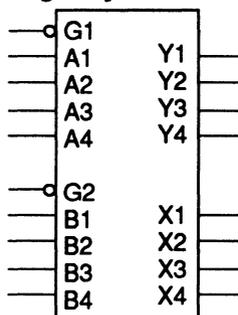
Function Table

Inputs						Outputs											
G1	A1	A2	A3	A4	G2	B1	B2	B3	B4	Y1	Y2	Y3	Y4	X1	X2	X3	X4
L	L	L	L	L	L	L	L	L	L	H	H	H	H	H	H	H	H
H	X	X	X	X	L	H	L	H	H	Z	Z	Z	Z	L	H	L	L
L	L	H	H	L	H	L	H	H	H	H	L	L	H	Z	Z	Z	Z



- Two enable inputs
- 3-state outputs

Logic Symbol



Macrocell count: 8
 Array inputs: 10
 Product terms used: 8
 Product terms allocated: 32

Functional Description

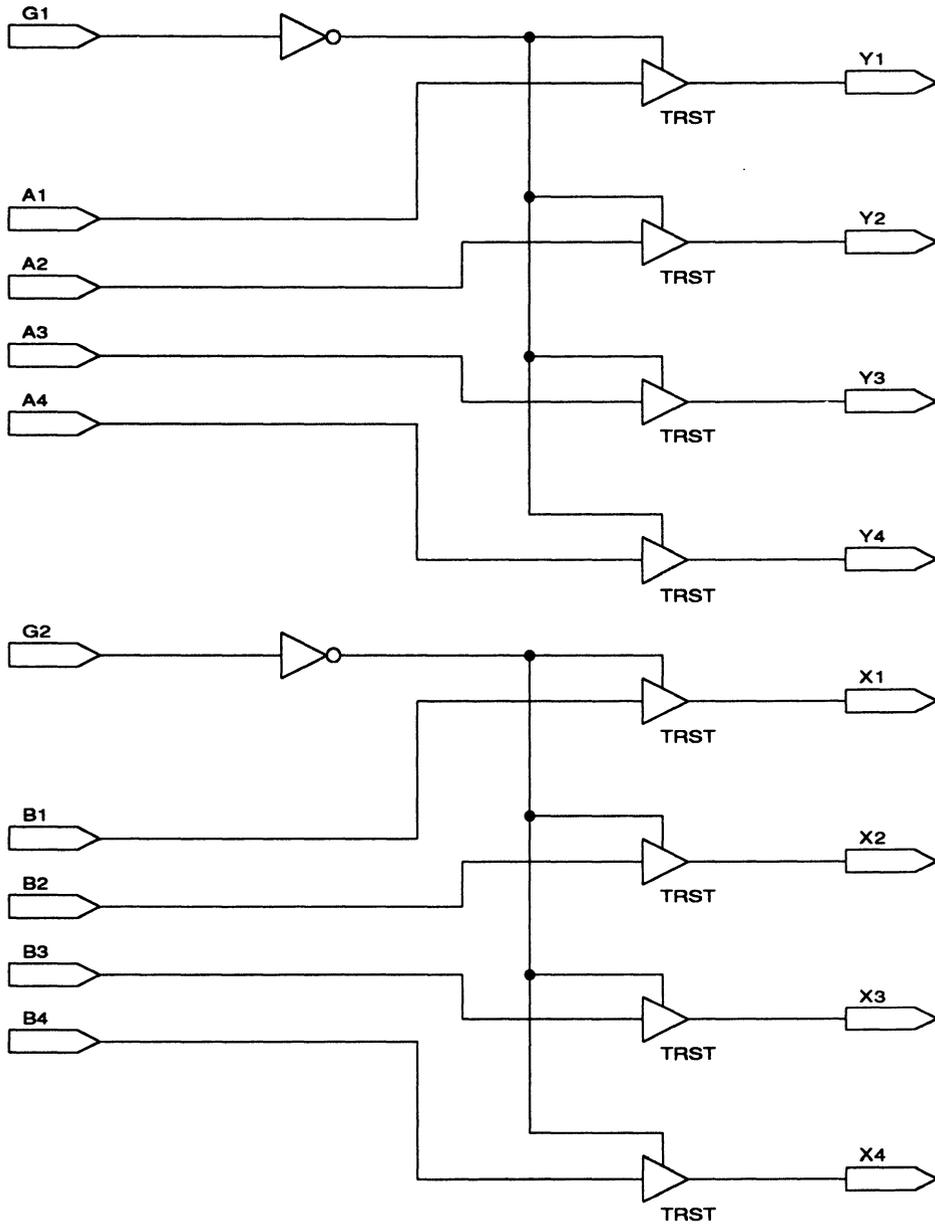
The 74244 macro contains two groups of four non-inverting buffers. Each group is enabled by an active-LOW input control line.

Sample PDS Equivalent

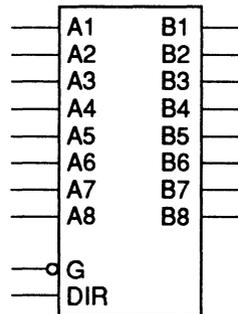
Y1.trst = /G1
 Y2.trst = /G1
 Y3.trst = /G1
 Y4.trst = /G1
 X1.trst = /G2
 X2.trst = /G2
 X3.trst = /G2
 X4.trst = /G2
 Y1 = A1
 Y2 = A2
 Y3 = A3
 Y4 = A4
 X1 = B1
 X2 = B2
 X3 = B3
 X4 = B4

Function Table

Inputs						Outputs											
G1	A1	A2	A3	A4	G2	B1	B2	B3	B4	Y1	Y2	Y3	Y4	X1	X2	X3	X4
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
H	X	X	X	X	L	H	L	H	H	Z	Z	Z	Z	H	L	H	H
L	L	H	H	L	H	L	H	H	H	L	H	H	L	Z	Z	Z	Z



- Enable input

Logic Symbol

Macrocell count: 16
 Array inputs: 18
 Product terms used: 16
 Product terms allocated: 64

Functional Description

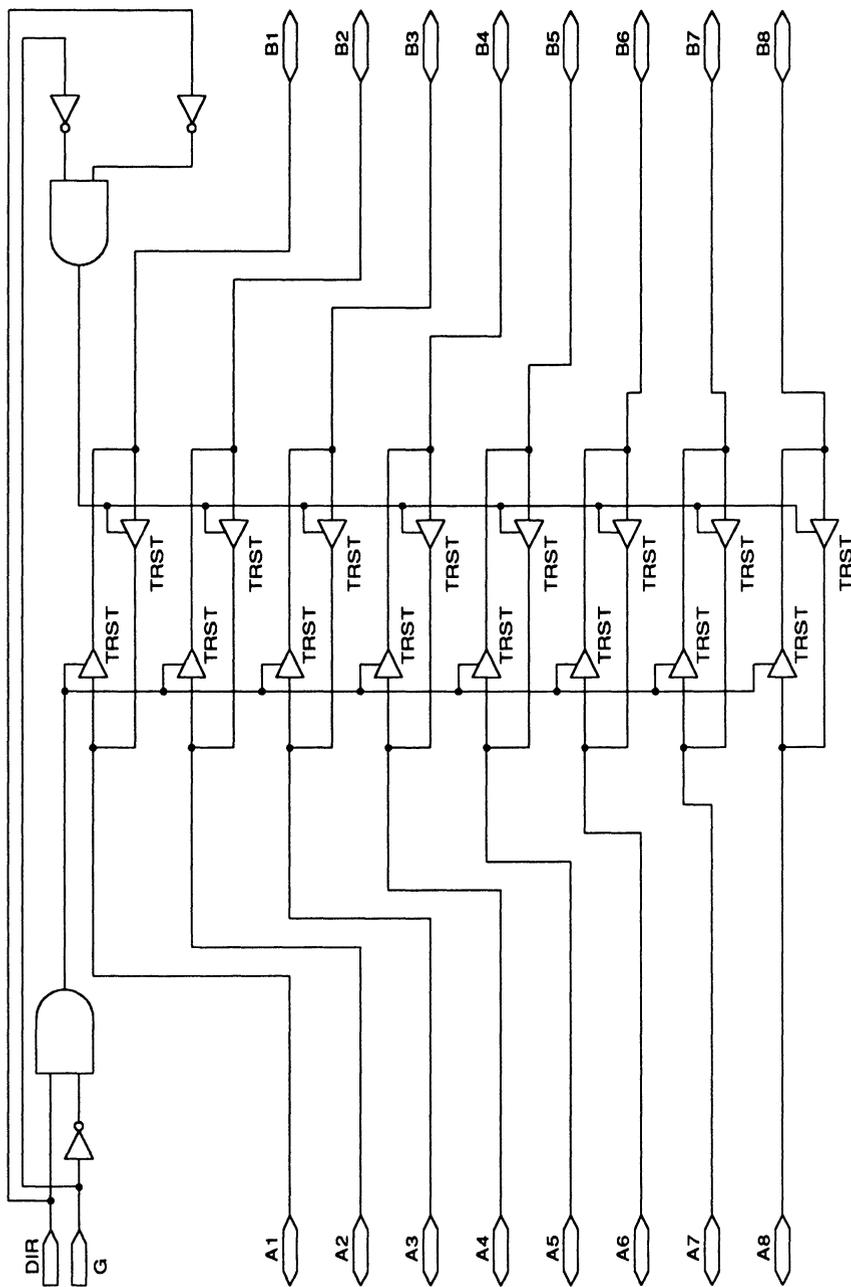
The 74245 macro implements an 8-bit bus transceiver. You can transmit data from bus A to bus B or from bus B to bus A. The data-transfer direction is controlled by the DIR control line. If the enable input, G, is set HIGH, then the buses are disabled and isolated.

Sample PDS Equivalent

$B1.trst = (DIR * /G)$	$B1 = A1$
$A1.trst = (/G * /DIR)$	$A1 = B1$
$B2.trst = (DIR * /G)$	$B2 = A2$
$A2.trst = (/G * /DIR)$	$A2 = B2$
$B3.trst = (DIR * /G)$	$B3 = A3$
$A3.trst = (/G * /DIR)$	$A3 = B3$
$B4.trst = (DIR * /G)$	$B4 = A4$
$A4.trst = (/G * /DIR)$	$A4 = B4$
$B5.trst = (DIR * /G)$	$B5 = A5$
$A5.trst = (/G * /DIR)$	$A5 = B5$
$B6.trst = (DIR * /G)$	$B6 = A6$
$A6.trst = (/G * /DIR)$	$A6 = B6$
$B7.trst = (DIR * /G)$	$B7 = A7$
$A7.trst = (/G * /DIR)$	$A7 = B7$
$B8.trst = (DIR * /G)$	$B8 = A8$
$A8.trst = (/G * /DIR)$	$A8 = B8$

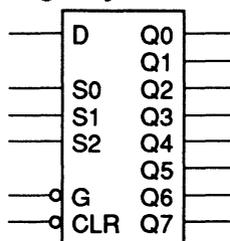
Function Table

Inputs		Operation
G	DIR	
L	L	Bus B Data to Bus A
L	H	Bus A Data to Bus B
H	X	Buses Isolated



- Four modes of operation
- Asynchronous reset

Logic Symbol



Macrocell count: 16
 Array inputs: 14
 Product terms used: 16
 Product terms allocated: 64

Functional Description

The 74259 macro is an 8-bit addressable latch with asynchronous reset logic. The following four modes of operation are selectable via the CLR and G inputs.

- **Addressable latch**
Data on the D input line is written to the latch addressed by the three select lines: S2, S1, and S0. The other latches retain their values.
- **Memory**
The latch outputs do not change.
- **Active-HIGH 3-to-8 demultiplexer**
The addressed latch output follows the data input while the other latch outputs are held LOW.
- **Reset**
All latch outputs are set LOW regardless of the value on the select and data-input lines.

Sample PDS Equivalent

$$Q7 = (((S2 * S1 * S0) * /G * D) + (/S2 * S1 * S0) * CLR * Q7) + (CLR * Q7 * G) + (Q7 * CLR * D)$$

$$Q6 = (((S2 * S1 * /S0) * /G * D) + (/S2 * S1 * /S0) * CLR * Q6) + (CLR * Q6 * G) + (Q6 * CLR * D)$$

$$Q5 = (((S2 * /S1 * S0) * /G * D) + (/S2 * /S1 * S0) * CLR * Q5) + (CLR * Q5 * G) + (Q5 * CLR * D)$$

$$Q4 = (((S2 * /S1 * /S0) * /G * D) + (/S2 * /S1 * /S0) * CLR * Q4) + (CLR * Q4 * G) + (Q4 * CLR * D)$$

$$Q3 = (((/S2 * S1 * S0) * /G * D) + (/S2 * S1 * S0) * CLR * Q3) + (CLR * Q3 * G) + (Q3 * CLR * D)$$

$$Q2 = (((/S2 * S1 * /S0) * /G * D) + (/S2 * S1 * /S0) * CLR * Q2) + (CLR * Q2 * G) + (Q2 * CLR * D)$$

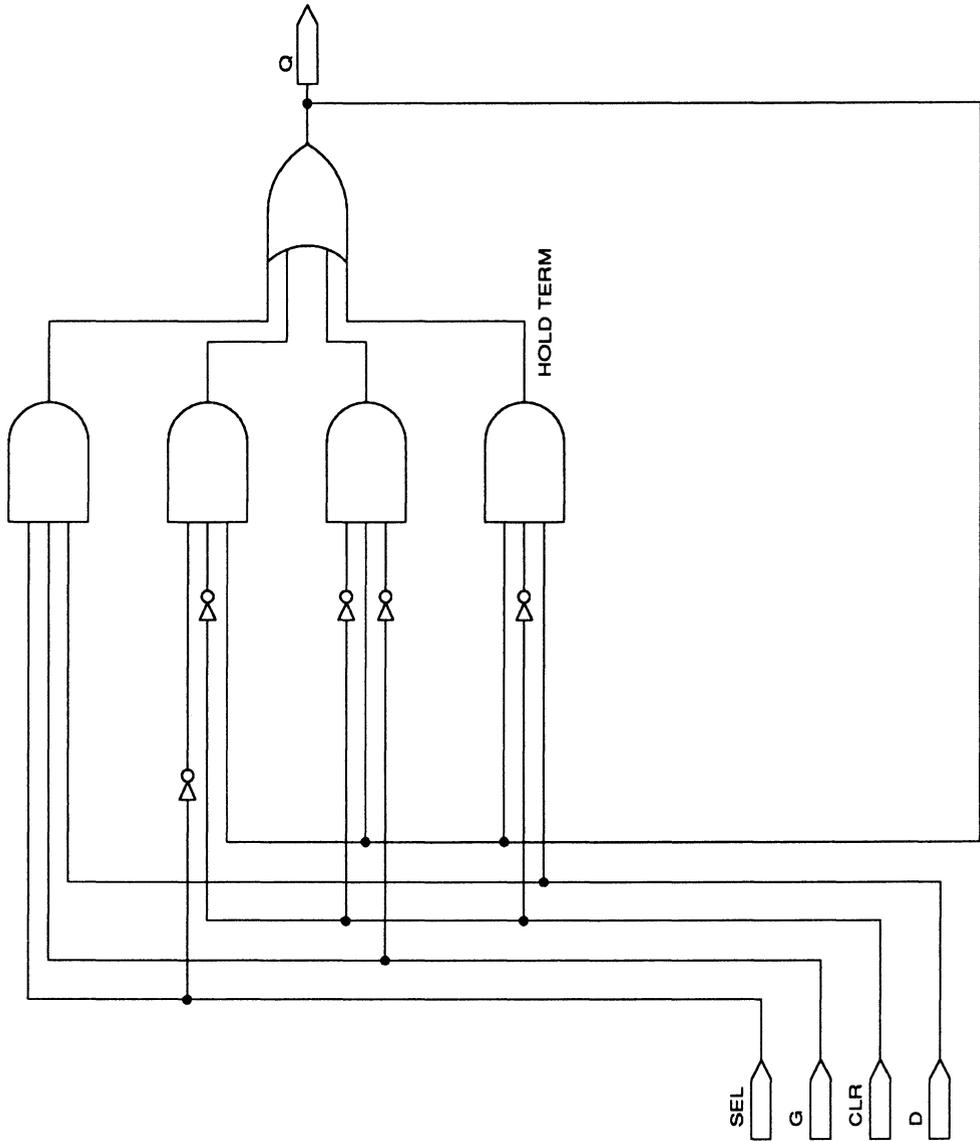
$$Q1 = (((/S2 * /S1 * S0) * /G * D) + (/S2 * /S1 * S0) * CLR * Q1) + (CLR * Q1 * G) + (Q1 * CLR * D)$$

$$Q0 = (((/S2 * /S1 * /S0) * /G * D) + (/S2 * /S1 * /S0) * CLR * Q0) + (CLR * Q0 * G) + (Q0 * CLR * D)$$

Function Table

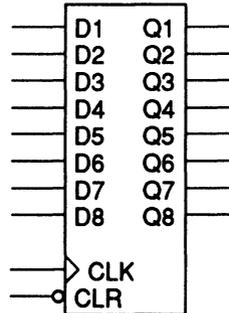
Inputs		Outputs		Functions
CLR	G	Addressed Latch	Other Latches	
H	L	D	Qo	Addressable Latch Memory 8-line demultiplexer Reset
H	H	Qo	Qo	
L	L	D	L	
L	H	L	L	

* Qo = previous state of latch output Q
 D = data input D



- Asynchronous reset

Logic Symbol



Macrocell count: 8
 Array inputs: 9
 Product terms used: 8
 Product terms allocated: 32

Functional Description

The 74273 macro is an octal D-FF bank with asynchronous reset logic.

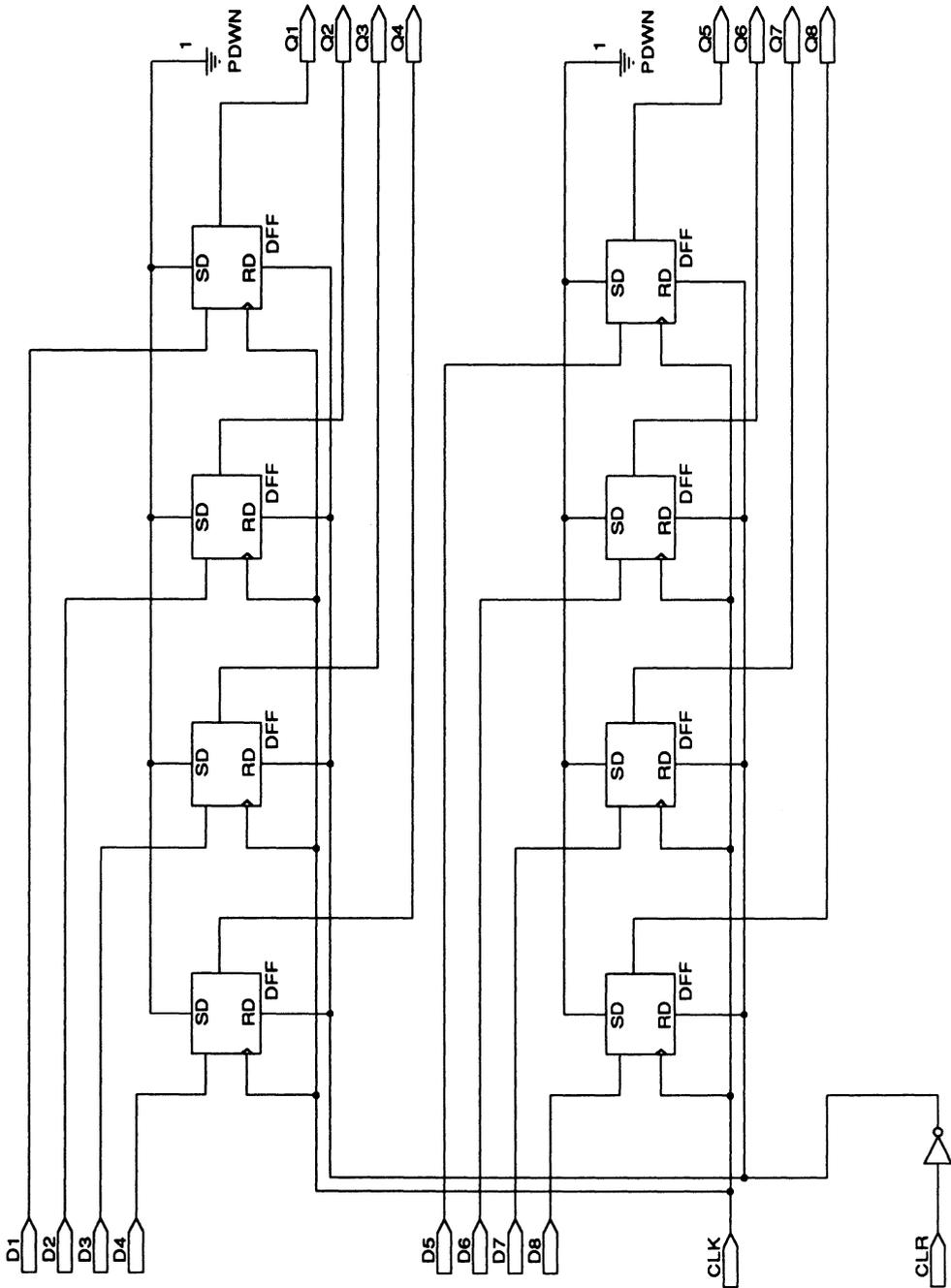
Sample PDS Equivalent

Q1 = D1
 Q1.clkf = CLK
 Q1.rstf = /CLR
 Q2 = D2
 Q2.clkf = CLK
 Q2.rstf = /CLR
 Q3 = D3
 Q3.clkf = CLK
 Q3.rstf = /CLR
 Q4 = D4
 Q4.clkf = CLK
 Q4.rstf = /CLR
 Q5 = D5
 Q5.clkf = CLK
 Q5.rstf = /CLR
 Q6 = D6
 Q6.clkf = CLK
 Q6.rstf = /CLR
 Q7 = D7
 Q7.clkf = CLK
 Q7.rstf = /CLR
 Q8 = D8
 Q8.clkf = CLK
 Q8.rstf = /CLR

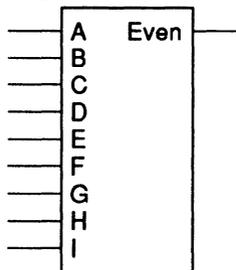
Function Table

Inputs			Outputs
CLR	CLK	D	Q
L	X	X	L
H	↑	L	L
H	↑	H	H
H	L	X	Q _o

* Q_o = previous state of Q



- Active-HIGH output

Logic Symbol

Macrocell count: 4
 Array inputs: 12
 Product terms used: 16
 Product terms allocated: 16

Functional Description

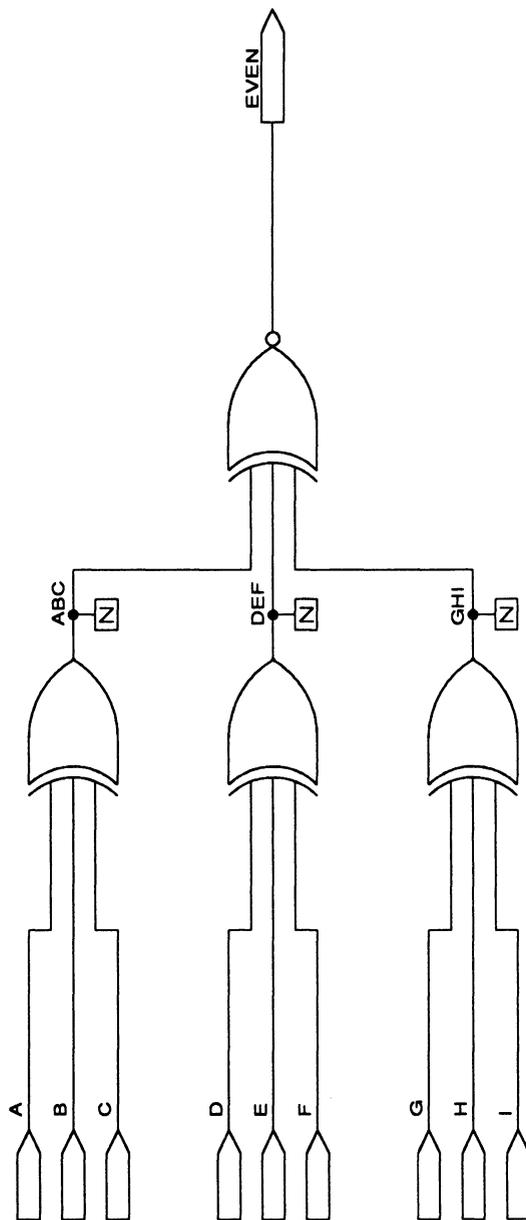
The 74280 macro is a combinatorial circuit that generates or checks for even parity on nine data lines. Odd parity is obtained by taking the inversion of the EVEN parity output.

Sample PDS Equivalent

ABC = (A :+: B :+: C)
 DEF = (D :+: E :+: F)
 EVEN = /(ABC :+: DEF :+: GHI)
 GHI = (G :+: H :+: I)

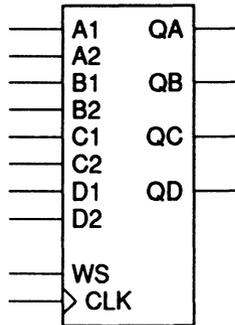
Function Table

Inputs	Outputs
Number of Inputs A - I that are High	Even
0, 2, 4, 6, 8, 1, 3, 5, 7, 9	H L



- Synchronous storage

Logic Symbol



Macrocell count: 8
 Array inputs: 21
 Product terms used: 32
 Product terms allocated: 32

Functional Description

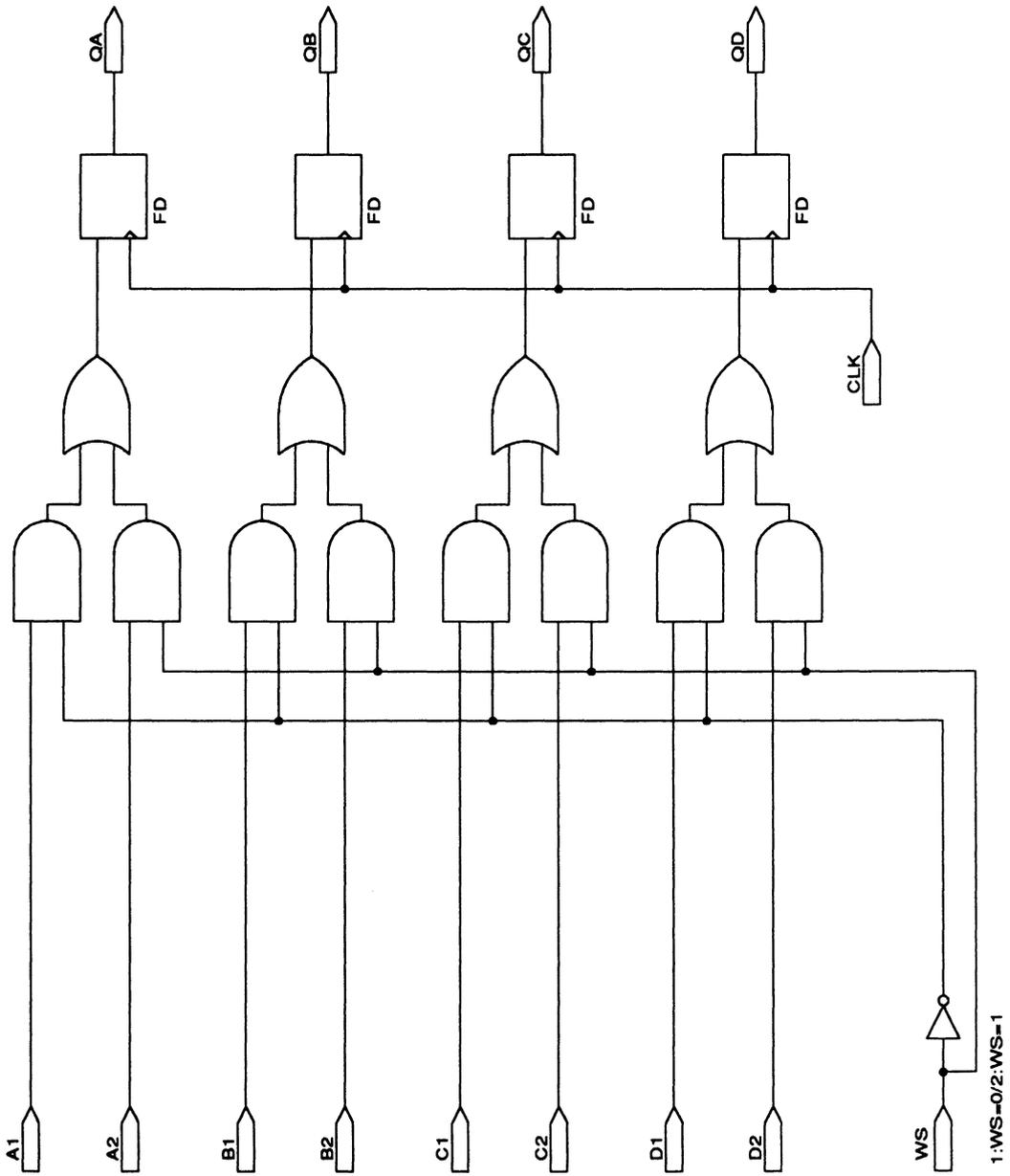
The 74298 macro selects one of two 4-bit words and stores each bit in a register on the rising edge of CLK.

Sample PDS Equivalent

$QA = ((A1 * /WS) + (A2 * WS))$
 QA.clkf = CLK
 $QB = ((B1 * /WS) + (B2 * WS))$
 QB.clkf = CLK
 $QC = ((C1 * /WS) + (C2 * WS))$
 QC.clkf = CLK
 $QD = ((D1 * /WS) + (D2 * WS))$
 QD.clkf = CLK

Function Table

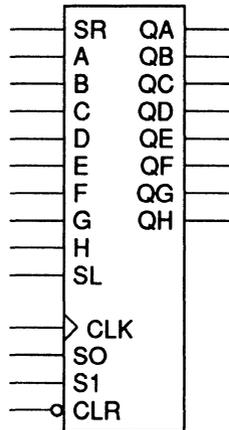
Inputs		Outputs			
CLK	WS	QA	QB	QC	QD
L	X	QA ₀	QB ₀	QC ₀	QD ₀
↑	L	A1	B1	C1	D1
↑	H	A2	B2	C2	D2



1:WS=0/2:WS=1

- Parallel-to-serial converter
- Serial-to-parallel converter
- Synchronous reset
- Synchronous loading

Logic Symbol



Macrocell count: 8
 Array inputs: 21
 Product terms used: 32
 Product terms allocated: 32

Functional Description

The 74299X macro is composed of two 74194s connected to form an 8-bit bidirectional universal shift register with synchronous reset logic.

Note:

The TTL version has three-state bidirectional I/Os that serve as the parallel-load inputs as well as the Q outputs.

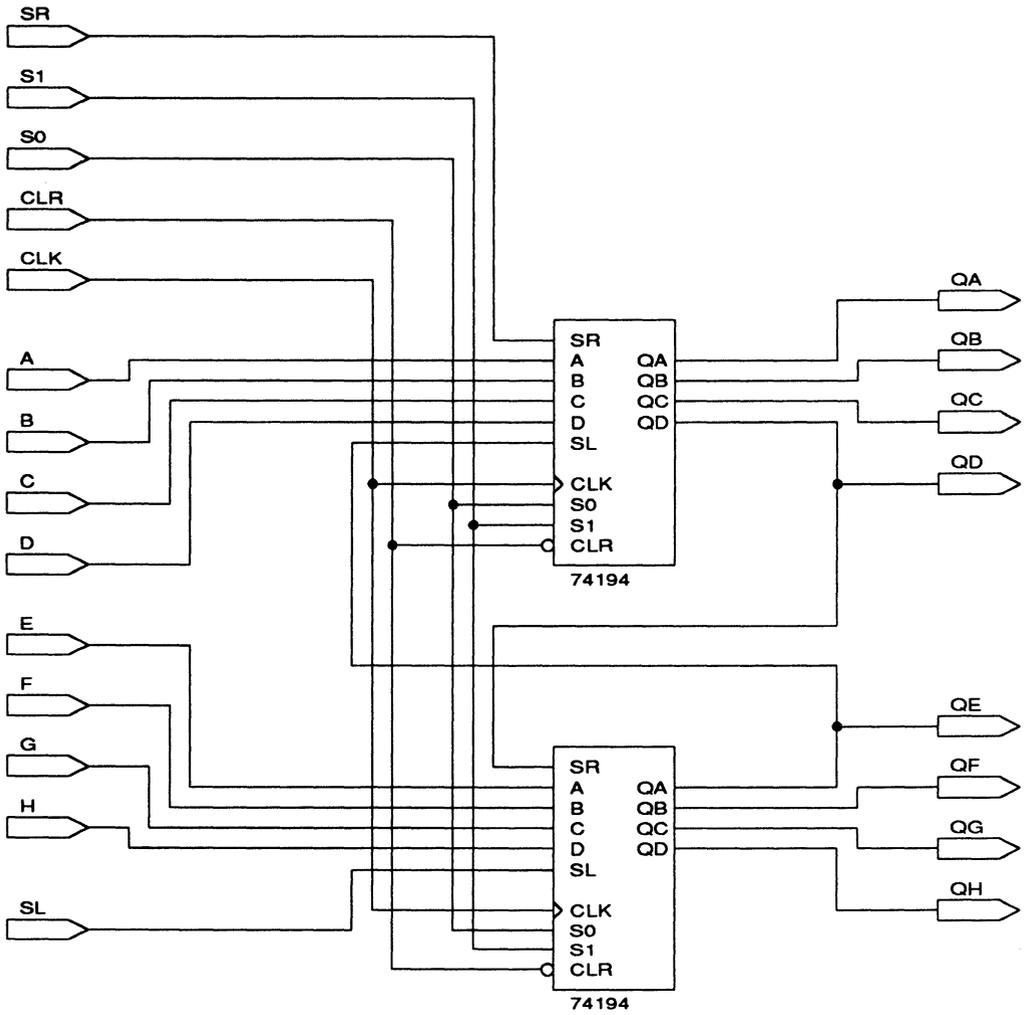
Sample PDS Equivalent

$QA = ((CLR * SR * /S1 * S0) + (CLR * QB * S1 * /S0) + (CLR * S1 * S0 * A) + (CLR * /S1 * /S0 * QA))$
 QA.clkf = CLK
 $QB = ((CLR * QA * /S1 * S0) + (CLR * QC * S1 * /S0) + (CLR * S1 * S0 * B) + (CLR * /S1 * /S0 * QB))$
 QB.clkf = CLK
 $QC = ((CLR * QB * /S1 * S0) + (CLR * QD * S1 * /S0) + (CLR * S1 * S0 * C) + (CLR * /S1 * /S0 * QC))$
 QC.clkf = CLK
 $QD = ((CLR * QC * /S1 * S0) + (CLR * QE * S1 * /S0) + (CLR * S1 * S0 * D) + (CLR * /S1 * /S0 * QD))$
 QD.clkf = CLK
 $QE = ((CLR * QD * /S1 * S0) + (CLR * QF * S1 * /S0) + (CLR * S1 * S0 * E) + (CLR * /S1 * /S0 * QE))$
 QE.clkf = CLK
 $QF = ((CLR * QE * /S1 * S0) + (CLR * QG * S1 * /S0) + (CLR * S1 * S0 * F) + (CLR * /S1 * /S0 * QF))$
 QF.clkf = CLK
 $QG = ((CLR * QF * /S1 * S0) + (CLR * QH * S1 * /S0) + (CLR * S1 * S0 * G) + (CLR * /S1 * /S0 * QG))$
 QG.clkf = CLK
 $QH = ((CLR * QG * /S1 * S0) + (CLR * SL * S1 * /S0) + (CLR * S1 * S0 * H) + (CLR * /S1 * /S0 * QH))$
 QH.clkf = CLK

Function Table

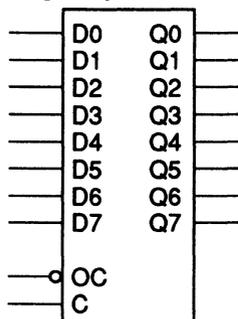
		Inputs				Outputs							
Mode		Serial				Parallel							
CLK	CLR	S1	S0	SL	SR	A	B	C	D	QA	QB	QC	QD
L	X	X	X	X	X	X	X	X	X	QAo	QB0	QC0	QDo
↑	L	X	X	X	X	X	X	X	X	L	L	L	L
X	H	L	L	X	X	X	X	X	X	QAo	QB0	QC0	QDo
L	H	L	H	X	L	X	X	X	X	L	QAn	QBn	QCn
↑	H	L	H	X	H	X	X	X	X	H	QAn	QBn	QCn
↑	H	H	L	L	X	X	X	X	X	QBn	QCn	QDn	L
↑	H	H	L	H	X	X	X	X	X	QBn	QCn	QDn	H
↑	H	H	H	X	X	a	b	c	d	a	b	c	d

* QAo to QDo = previous state of QA to QD
 QAn to QDn = level of QA to QD before the most recent rising transition of the CLK, and indicates a 1-bit shift.



- Enable input

Logic Symbol



Macrocell count: 8
 Array inputs: 18
 Product terms used: 16
 Product terms allocated: 32

Functional Description

The 74373 macro is an octal D latch with an active-LOW enable input.

Sample PDS Equivalent

$$Q1 = ((Q1 * VCC * /C) + (VCC * GND) + (VCC * C * D1) + (D1 * VCC * Q1))$$

$$Q1.trst = /OC$$

$$Q2 = ((Q2 * VCC * /C) + (VCC * GND) + (VCC * C * D2) + (D2 * VCC * Q2))$$

$$Q2.trst = /OC$$

$$Q3 = ((Q3 * VCC * /C) + (VCC * GND) + (VCC * C * D3) + (D3 * VCC * Q3))$$

$$Q3.trst = /OC$$

$$Q4 = ((Q4 * VCC * /C) + (VCC * GND) + (VCC * C * D4) + (D4 * VCC * Q4))$$

$$Q4.trst = /OC$$

$$Q5 = ((Q5 * VCC * /C) + (VCC * GND) + (VCC * C * D5) + (D5 * VCC * Q5))$$

$$Q5.trst = /OC$$

$$Q6 = ((Q6 * VCC * /C) + (VCC * GND) + (VCC * C * D6) + (D6 * VCC * Q6))$$

$$Q6.trst = /OC$$

$$Q7 = ((Q7 * VCC * /C) + (VCC * GND) + (VCC * C * D7) + (D7 * VCC * Q7))$$

$$Q7.trst = /OC$$

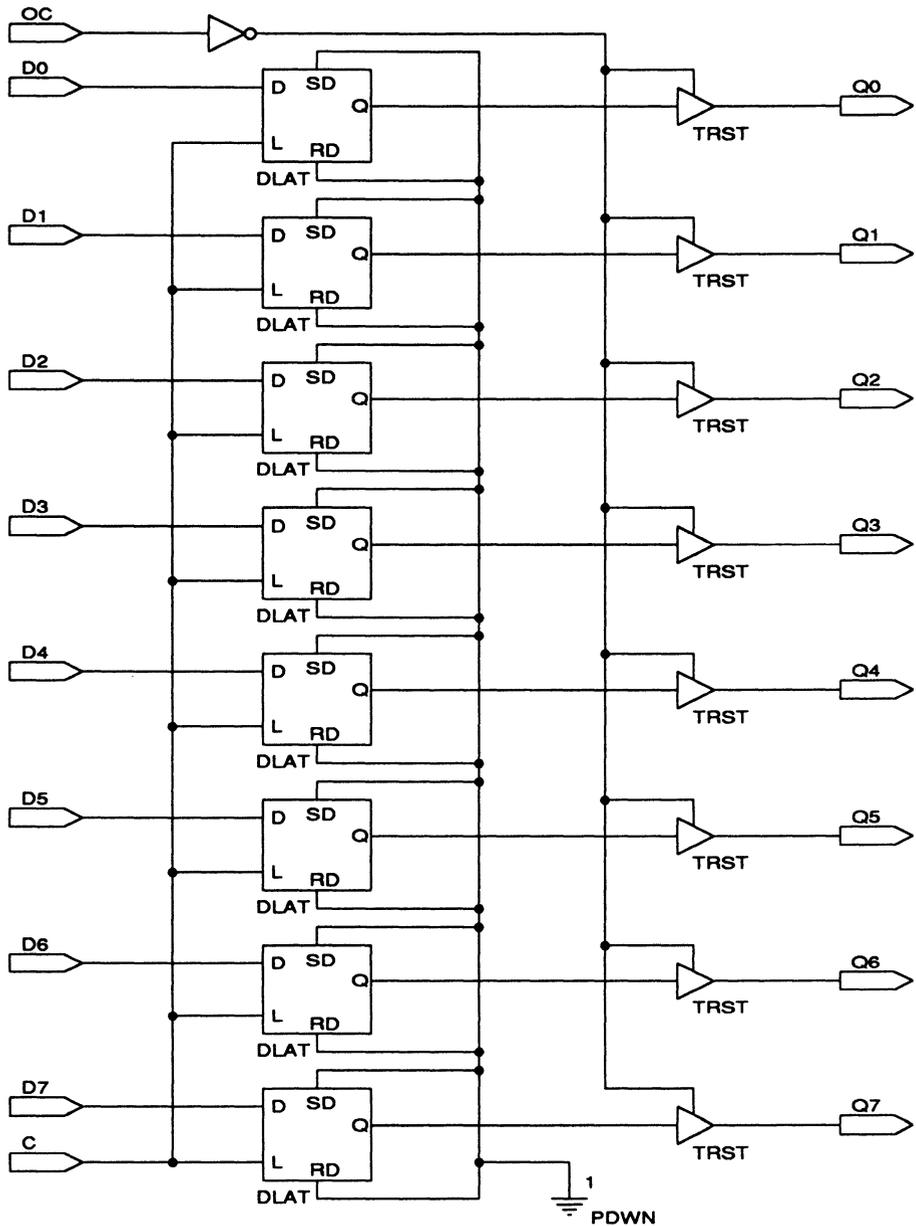
$$Q8 = ((Q8 * VCC * /C) + (VCC * GND) + (VCC * C * D8) + (D8 * VCC * Q8))$$

$$Q8.trst = /OC$$

Function Table (for each D latch)

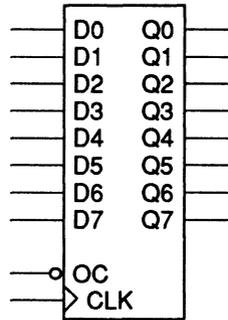
Inputs			Outputs
OC	C	D	Q
H	X	X	Z
L	H	H	H
L	H	L	L
L	L	X	Q _o

* Q_o = previous state of Q



- Enable input
- 3-state outputs

Logic Symbol



Macrocell count: 8
 Array inputs: 9
 Product terms used: 8
 Product terms allocated: 32

Functional Description

The 74374 macro is an octal D-type register with an active-LOW enable input.

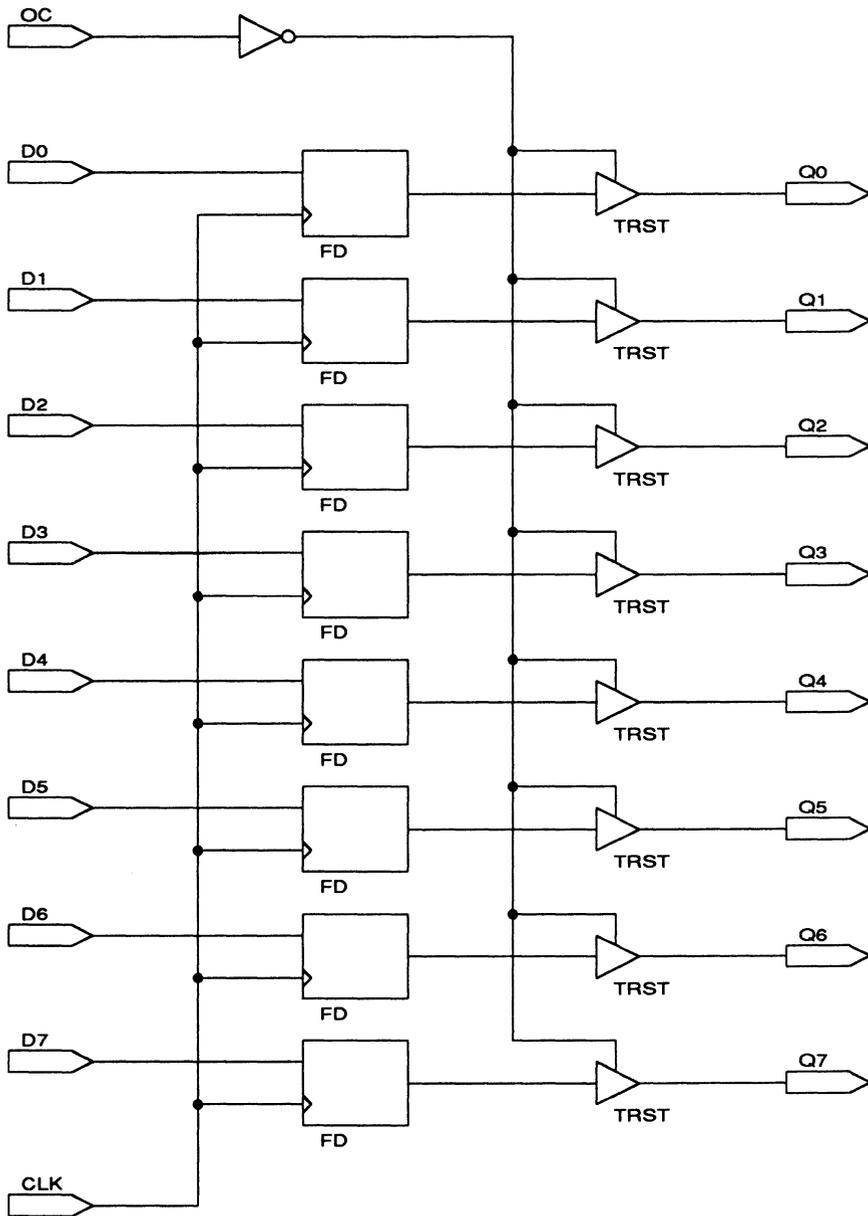
Sample PDS Equivalent

Q1.trst = /OC	Q5.trst = /OC
Q1 = D1	Q5 = D5
Q1.clkf = CLK	Q5.clkf = CLK
Q2.trst = /OC	Q6.trst = /OC
Q2 = D2	Q6 = D6
Q2.clkf = CLK	Q6.clkf = CLK
Q3.trst = /OC	Q7.trst = /OC
Q3 = D3	Q7 = D7
Q3.clkf = CLK	Q7.clkf = CLK
Q4.trst = /OC	Q8.trst = /OC
Q4 = D4	Q8 = D8
Q4.clkf = CLK	Q8.clkf = CLK

Function Table (for each D flip-flop)

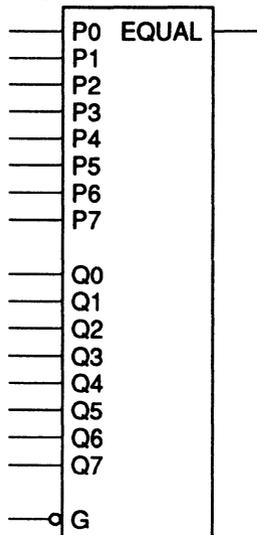
Inputs			Outputs
OC	CLK	D	Q
H	X	X	Z
L	↑	H	H
L	↑	L	L
L	L	X	Q _o

* Q_o = previous state of Q



- Enable input
- Active-HIGH output

Logic Symbol



Macrocell count: 5
 Array inputs: 21
 Product terms used: 17
 Product terms allocated: 20

Functional Description

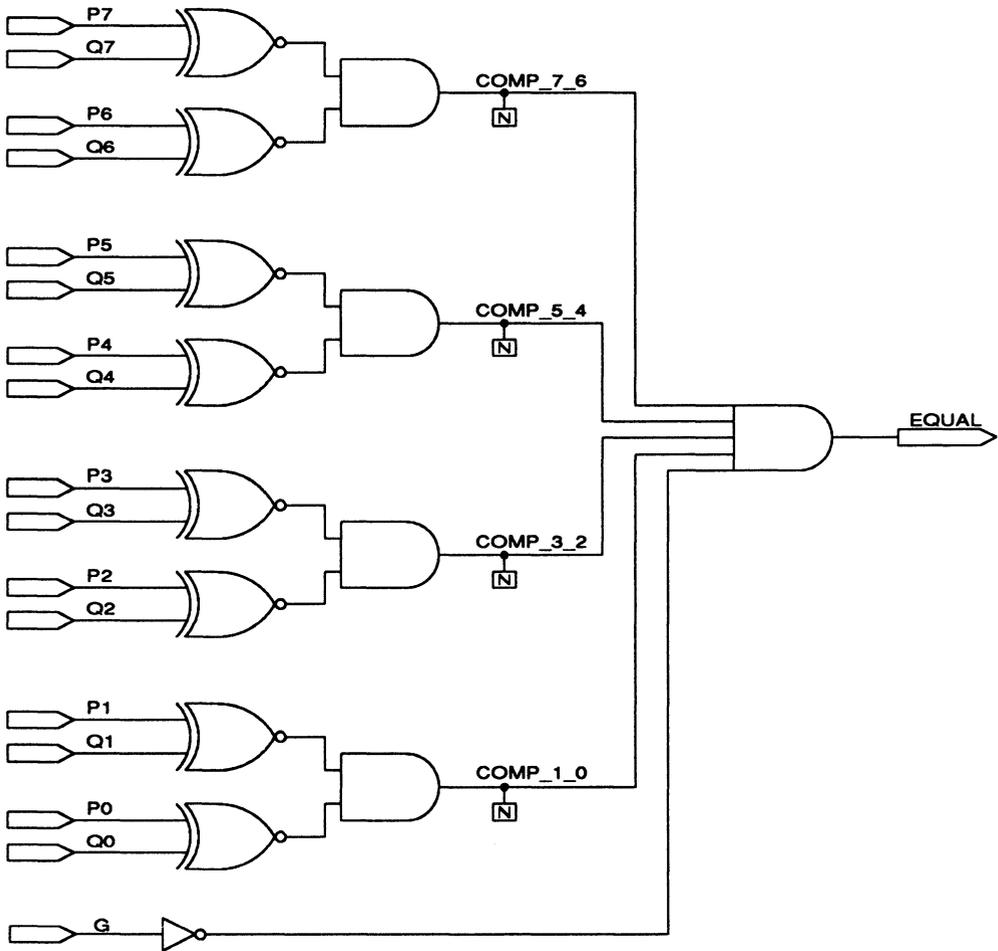
The 74518 macro compares two 8-bit numbers and sets the EQUAL output HIGH if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

Sample PDS Equivalent

```
COMP_7_6 = ((P7 :+ Q7) * !(P6 :+ Q6))
COMP_5_4 = ((P5 :+ Q5) * !(P4 :+ Q4))
EQUAL = (COMP_7_6 * COMP_5_4 * COMP_3_2
  * COMP_1_0 * /G)
COMP_3_2 = ((P3 :+ Q3) * !(P2 :+ Q2))
COMP_1_0 = ((P1 :+ Q1) * !(P0 :+ Q0))
```

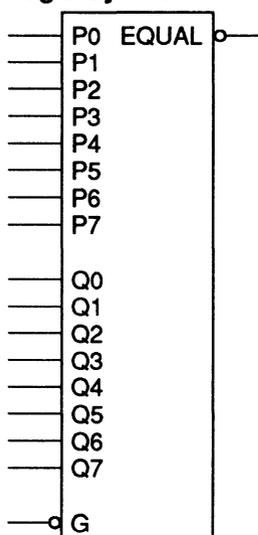
Function Table

Inputs		Outputs	
DATA P, Q	G	EQUAL	
P(7:0) = Q(7:0)	L	H	
P(7:0) > Q(7:0)	L	L	
P(7:0) < Q(7:0)	L	L	
X	H	L	



- Enable input
- Active-LOW output

Logic Symbol



Macrocell count: 5
 Array inputs: 21
 Product terms used: 17
 Product terms allocated: 20

Functional Description

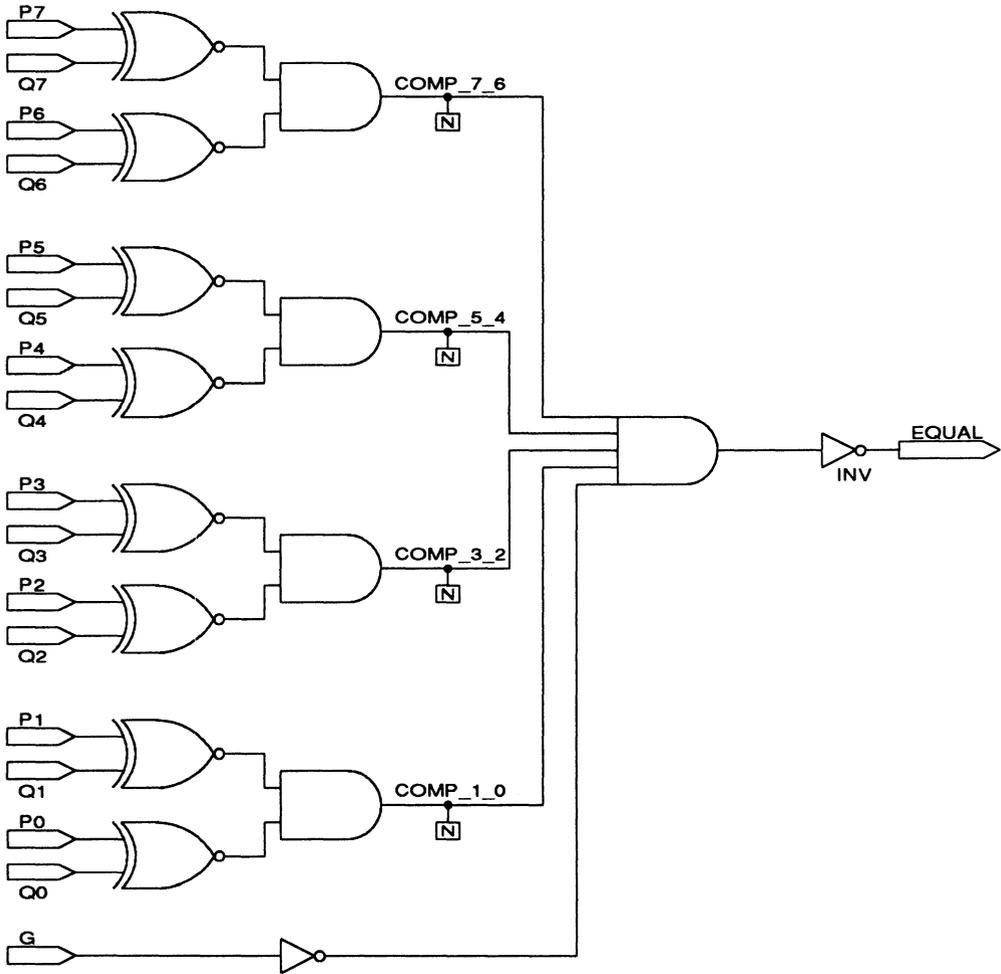
The 74521 macro compares two 8-bit numbers and sets the EQUAL output LOW if the two numbers are equal. The enable input, G, must be held LOW to enable the EQUAL output.

Sample PDS Equivalent

```
COMP_7_6 = ((P7 :+ Q7) * !(P6 :+ Q6))
COMP_5_4 = ((P5 :+ Q5) * !(P4 :+ Q4))
EQUAL = (COMP_7_6 * COMP_5_4 * COMP_3_2
  * COMP_1_0 * /G)
COMP_3_2 = ((P3 :+ Q3) * !(P2 :+ Q2))
COMP_1_0 = ((P1 :+ Q1) * !(P0 :+ Q0))
```

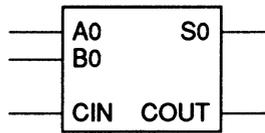
Function Table

Inputs		Outputs	
DATA P, Q	G	EQUAL	
P(7:0) = Q(7:0)	L	L	
P(7:0) > Q(7:0)	L	H	
P(7:0) < Q(7:0)	L	H	
X	H	H	



- Carry input
- Carry output

Logic Symbol



Macrocell count: 2
 Array inputs: 3
 Product terms used: 7
 Product terms allocated: 8

Functional Description

The ADD1 macro adds two 1-bit numbers. You can use the carry-out and carry-in signals to cascade multiple adders.

Sample PDS Equivalent

$$S0 = (A0 \oplus B0 \oplus CIN)$$

$$COUT = ((A0 * B0) + (CIN * (A0 \oplus B0)))$$

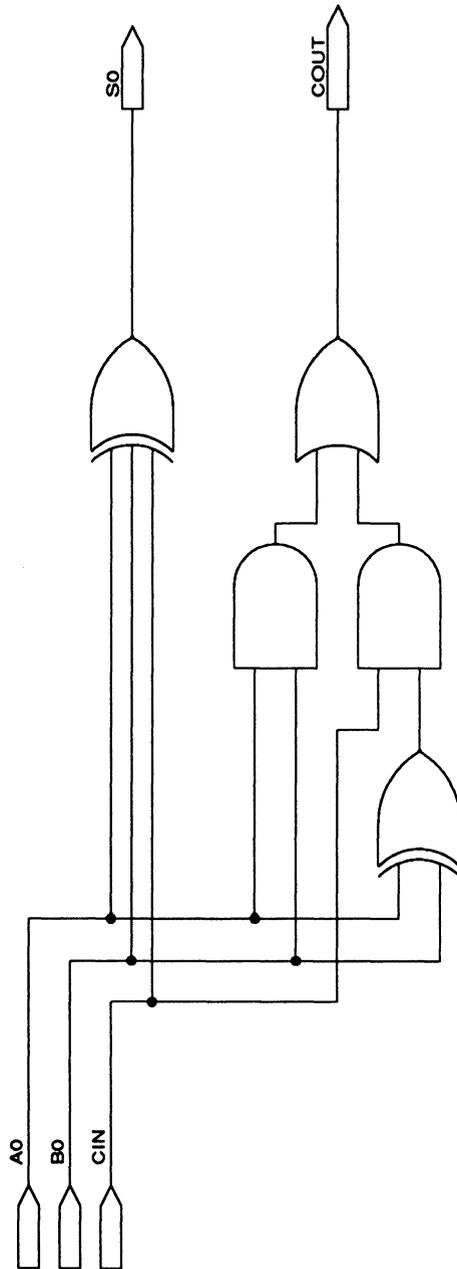
Function Table

Inputs			Outputs	
A0	B0	CIN	COUT	S0
L	L	L	L	L
L	L	H	L	H
L	H	L	L	H
L	H	H	H	L
H	L	L	L	H
H	L	H	H	L
H	H	L	H	L
H	H	H	H	H

ADD1

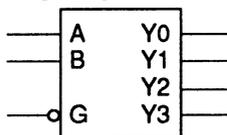
1-Bit Full Adder

ADD1



- Enable input

Logic Symbol



Macrocell count: 4
 Array inputs: 3
 Product terms used: 4
 Product terms allocated: 16

Functional Description

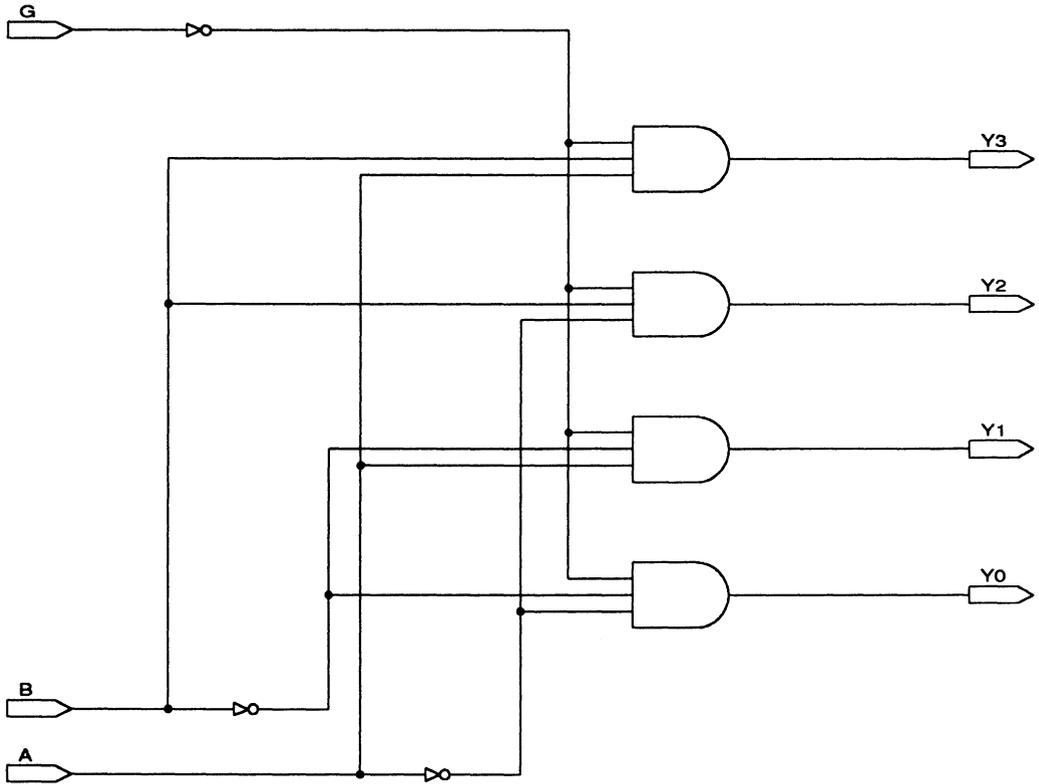
The DECODE4 macro decodes one of four active-HIGH output lines depending on the 2-bit data inputs. The enable input, G, must be LOW to activate the decoder. You can use the enable inputs to cascade multiple decoders.

Sample PDS Equivalent

$Y3 = (/G * B * A)$
 $Y2 = (/G * B * /A)$
 $Y1 = (/G * /B * A)$
 $Y0 = (/G * /B * /A)$

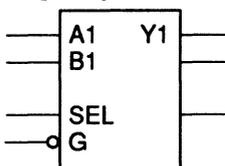
Function Table

Inputs			Outputs			
B	A	G	Y0	Y1	Y2	Y3
H	L	H	L	L	L	L
L	L	L	H	L	L	L
L	H	L	L	H	L	L
H	L	L	L	L	H	L
H	H	L	L	L	L	H



- Enable input

Logic Symbol



Macrocell count: 1
 Array inputs: 4
 Product terms used: 2
 Product terms allocated: 4

Functional Description

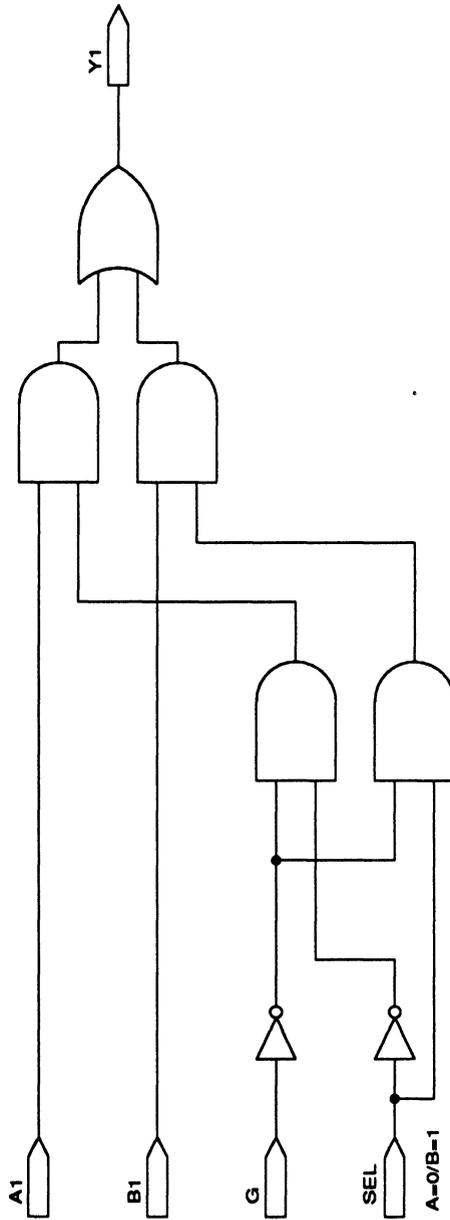
The MUX2 macro decodes one data-input line to select one of two data sources. The enable input, G, must be LOW to enable the Y1 output.

Sample PDS Equivalent

$$Y1 = ((A1 * (/G * /SEL)) + (B1 * (/G * SEL)))$$

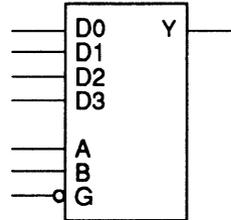
Function Table

Inputs		Outputs
Select	Strobe	
Sel	G	Y1
X	H	L
L	L	A1
H	L	B1



- Enable input

Logic Symbol



Macrocell count: 1
 Array inputs: 7
 Product terms used: 4
 Product terms allocated: 4

Functional Description

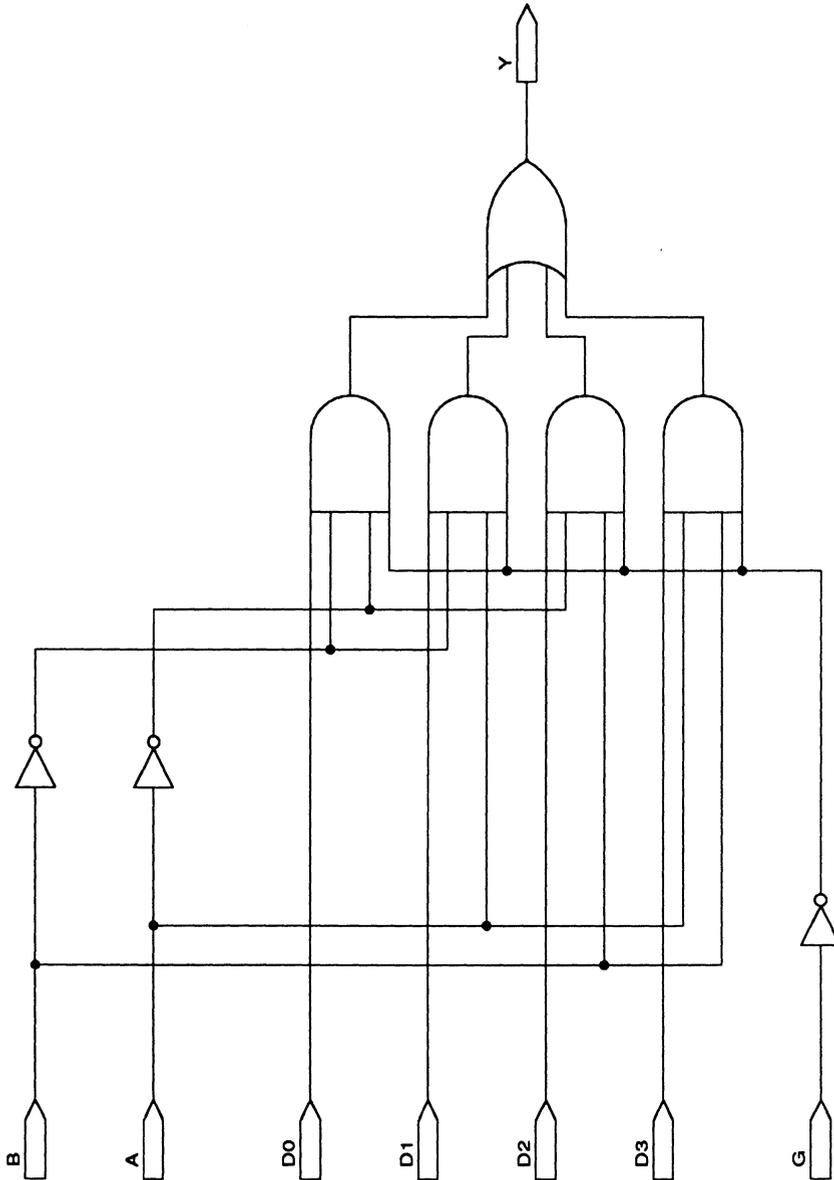
The MUX4 macro decodes two data-input lines to select one of four data sources. The enable input, G, must be LOW to enable the Y output.

Sample PDS Equivalent

$$Y = ((D0 * /B * /A * /G) + (D1 * /B * A * /G) + (D2 * /A * B * /G) + (D3 * A * B * /G))$$

Function Table

Inputs			Outputs
Select	Strobe		
B	A	G	Y
X	X	H	L
L	L	L	D0
L	H	L	D1
H	L	L	D2
H	H	L	D3



SECTION IV

SOFTWARE REFERENCE

Chapter 9: Menus and Commands

Chapter 10: Language Reference

Chapter 11: Device Programming Reference

CHAPTER 9

MENUS AND COMMANDS

CONTENTS

MENUS AND COMMANDS	1
9.1 OVERVIEW	2
9.1.1 FEATURES	3
9.1.2 CONVENTIONS.....	5
9.2 COMMANDS AND OPTIONS	7
9.2.1 FILE MENU.....	8
9.2.1.1 Begin New Design	8
Text-Based Design Form	10
Schematic-Based Design Form.....	12
9.2.1.2 Retrieve an Existing Design.....	15
9.2.1.3 Merge Design Files	16
Files Menu	18
Get Next Input File	18
Merge Files	19
List Combined Files	19
Save	20
Abandon Input	20
Quit	20
Editor Menu	21
Edit a File	21
View the Output Buffer	22
Resolution Menu.....	22
Resolve Detectable Conflicts	22
Bind Pins/Nodes	26
Edit Pin/Node List	28
Setup Menu	30
Options	30
Set Renaming Strategy	31

9.2.1.4	Change Directory	32
9.2.1.5	Delete Specified Files	33
9.2.1.6	Set Up	34
	Working Environment	34
	Compilation Options	37
	Simulation Options	43
	Logic Synthesis Options	44
9.2.1.7	Go To System	48
9.2.1.8	Quit	48
9.2.2	EDIT MENU	49
9.2.2.1	Text File	49
9.2.2.2	Schematic File	50
9.2.2.3	Control File for Schematic Design	50
9.2.2.4	Auxiliary Simulation File.....	51
9.2.2.5	Other File	51
9.2.3	RUN MENU	52
9.2.3.1	Compile	53
9.2.3.2	Simulation	54
9.2.3.3	Both.....	55
9.2.3.4	Other Operations	55
	Convert Schematic to Text	55
	Back Annotate Signals	56
	Disassemble From.....	56
	Recalculate JEDEC Checksum.....	58
	Translate from PLPL.....	59
	Execute.....	59
9.2.4	VIEW MENU	60
9.2.4.1	Execution Log File	61
9.2.4.2	Design File.....	61
9.2.4.3	Reports	62
	Fuse Map	62
	Mach Report	62
9.2.4.4	JEDEC Data	63
	Fuse Data Only.....	63
	Vectors + Fuse Data.....	63
9.2.4.5	Simulation Data.....	64
	History.....	64
	Trace.....	65

9.2.4.6	Waveform Display	65
	History.....	66
	Trace.....	66
9.2.4.7	Current Disassembled File.....	66
9.2.4.8	Pinout.....	67
9.2.4.9	Netlist Report	67
9.2.4.10	Other File	68
9.2.5	DOWNLOAD MENU.....	68
9.2.6	DOCUMENTATION MENU.....	69
9.2.6.1	Index of Topics	69
9.2.6.2	Language Reference	70
	Overview.....	71
	Syntax.....	71
	Definitions.....	71
	Use	71
	Related Topics	71
	Previous Menu	71
9.2.6.3	Help On Errors.....	71
9.2.7	[F1] FOR HELP	72

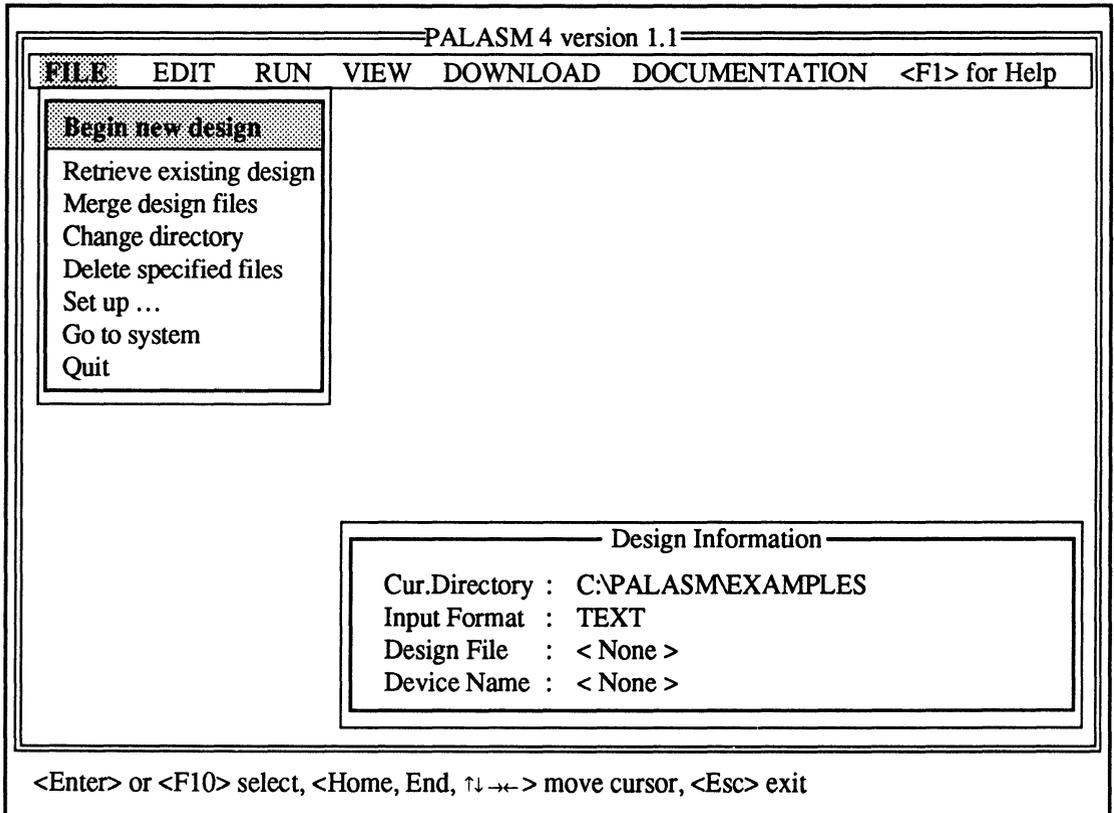
The *PALASM 4* software provides a unique environment that furnishes all commands required to develop a PLD or MACH-device design. This chapter is divided into two major topics.

- The overview, 9.1, introduces the features and conventions of the software.
- The commands and options discussion, 9.2, provides definitions and operational details for commands on the File, Edit, Run, View, Download, and Documentation menus.

9.1 OVERVIEW

Menus allow you to view and quickly select any command you need to produce and debug a PLD or MACH-device design. Using commands provided on menus, you can create or retrieve a design, process it, view and print reports, and download the JEDEC file to a device programmer.

The top-level screen is shown next. Various areas are identified and described after the figure.



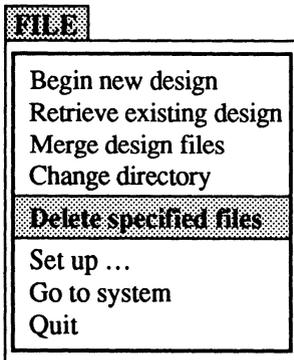
The bar across the top of the screen contains all **menu names**. A prompt on the right identifies how to access online Help. Each name reflects the kinds of commands on that menu.

- The **File** menu provides the file management, working environment, and system commands, as discussed under 9.2.1.

- The **Edit** menu provides commands to work on a particular kind of file in **either** the text editor **or** schematic editor, as described under 9.2.2.
- The **Run** menu lists all the commands you need to process a design file, as discussed under 9.2.3.
- The **View** menu includes commands to display files generated during each process, as described under 9.2.4.
- The **Download** menu provides access to the device programmer via a programmer-communication utility, as detailed under 9.2.5.
- The **Documentation** menu allows access to online reference material, as described under 9.2.6.

Depending on the working environment setup, which you define using the Set up command on the File menu, current design information appears in the lower-right corner of the screen. The status line at the bottom of the screen provides messages and prompts that change as needed.

9.1.1 FEATURES



Whether you're familiar with the environment or not, the menus and commands are easy to work with. The following features are standard.

- Drop-down-style menus like the one shown here
- Dialog-box-style forms
- Pop-up-style lists
- Keyboard invocation of commands

Commands, such as Set up, followed by ellipses, display a submenu of additional commands. When you select a command, one of three things may happen.

- A process may be initiated, in which case, a window usually opens.

- B. A submenu may appear listing additional commands for you to choose.
- C. A form may appear where you supply additional information.
- D. An option list appears only when you select an option field on a form.

A sample form is shown below. All forms presented in this chapter show the default options as they appear after first installing the software.

===== DELETE SPECIFIED FILES =====					
This Utility deletes all files in the current directory with the following extensions when 'Y' is selected.					
SHEMATIC PROCESS FILES	TEXT PROCESS FILES		OUTPUT FILES		
-----	-----		-----		
JNL	Y	BAK	Y	PL2	N
JXR	Y	TRE	Y	XPT	N
JNF	Y	TMP	Y	JED	N
FLS	Y	LIS	Y	HST	N
FLP	Y	@??	Y	TRF	N
SRF	Y	LOG	Y	JDC	N
CRF	Y			XRF	N
OXR	Y			BLC	N
Others	*.				

Each form provides one or more fields that typically contain information you can accept or change; the highlighted field is active. Most fields are composed of a field name and a corresponding specification. Three kinds of fields are provided: text, option, and status.

- You can type information, such as a file name, directly into the highlighted (active) **text** field.
- You press [F2] to display a list of choices for the active **option** field.

When you select an option, the list is dismissed and the specification on the form is updated.

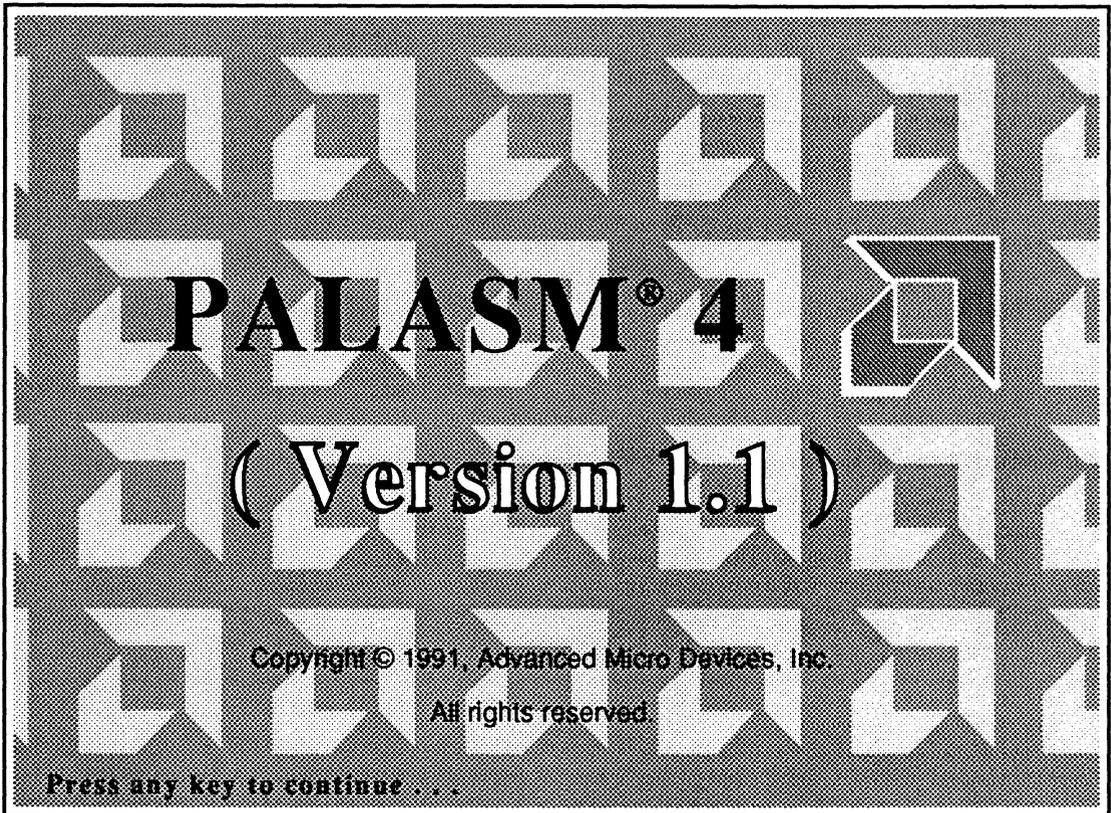
Note: An error is reported if you attempt to type into an option field.

- You cannot edit or change data in a **status** field. It's provided for information only.

9.1.2 CONVENTIONS

To use the software, you type the command shown in bold at left from the operating system. The copyright screen appears, as shown below.

PALASM [Enter]



[Enter]

After you press a key to dismiss the copyright screen, the PALASM menus become available. At this point, you use the keyboard to select commands and options and to activate fields in forms.

Table 1 describes how you select commands from menus, submenus, and lists and how to fill in a form.

Table 1: Select a Command and Fill in a Form

TASK	KEYBOARD
Open / display a menu (select menu name)	Press arrow keys to highlight menu name.
Select a command from open menu, submenu, or list	Type the first letter of the command, which is capitalized, or press arrow keys to highlight command, then press [Enter].
Select a field / move to next or previous field	Press arrow keys to highlight field.
Display options	Press [F2].
Select option	Press arrow keys to highlight item, then press [Enter] to select item.
Enter text	Type new text.
Edit text	Move cursor and backspace or retype.
Cancel form or list / return to previous menu bar	Press [Esc].
Confirm specifications in a form	Press either [Enter] or [F10].

- When you enter a form, the first field is active unless it's a status field. You can enter data, change data, or select another field.
- When you leave a form, you're returned to the previous form, submenu, or menu. You can select another command or exit.
- When you return to a menu or submenu, the command associated with the form remains highlighted.

9.2 COMMANDS AND OPTIONS

Discussions are divided according to menu name or function, starting with the File menu at the left and moving across the menu bar to the right.

- 9.2.1, File Menu
- 9.2.2, Edit Menu
- 9.2.3, Run Menu
- 9.2.4, View Menu
- 9.2.5, Download Menu
- 9.2.6, Documentation Menu
- 9.2.7, [F1] for Help

Within each discussion, commands are explained in logical order starting at the top of the menu and working through to the end.

- Any submenu or form that appears when you select a command is explained under the corresponding command discussion.
- Definitions for each field in a form are discussed in order, beginning at the top of the form and working through to the end.
- Choices for each field are discussed in order.

9.2.1 FILE MENU

FILE

Begin new design
Retrieve existing design
Merge design files
Change directory
Delete specified files
Set up ...
Go to system
Quit

The File menu appears automatically when you enter the software environment. As shown on the left, this menu provides two kinds of commands.

- **File management commands**
 - Begin new design
 - Retrieve existing design
 - Merge design files
- **Software-environment commands**
 - Change directory
 - Delete specified files
 - Set up ...
 - Go to system
 - Quit

Depending on the working environment you define using the Set up command, current design information may appear in the lower-right corner of the screen.

9.2.1.1 Begin New Design

FILE

Begin new design
Retrieve existing design
Merge design files
Change directory
Delete specified files
Set up ...
Go to system
Quit

This command is automatically highlighted each time you enter the software environment. Each new file you create is stored in the current working directory.¹

When you select the Begin new design command, a form appears so you can specify the file type and name.

Input format: TEXT
New file name:

¹ Refer to discussion 9.2.1.4, in this chapter, for details about changing the current working directory.

Input format: Text

This option field specifies the type of design you'll produce. Text refers to a text-based PDS file, which is the default.

To produce an OrCAD/SDT III schematic-based design, you must press [F2] to display the options and select Schematic.

Note: Schematic-base designs are supported only for MACH devices.

New file name:

You type the name, which must adhere to standard DOS naming conventions, in the new file name text field.

- Use any combination of upper- and/or lowercase letters, numbers, the underscore, `_`, and dollar sign, `$`, characters.
- Use up to eight, 8, characters and an optional extension: either `.PDS` for Boolean or state-machine designs or `.SCH` for schematic designs.

When you confirm your specifications, the name you specified is compared with existing file names.

- If the name corresponds to an existing file of the same type, you're asked if you want to overwrite the existing file.

In this case, you respond by typing the letter Y to overwrite the old file or the letter N.

- If the name is unique, one of two forms appears, depending on the kind of file you specified.

Each form is described next.

Text-Based Design Form

After confirming a text-based format and design name, you're immediately transferred to the form shown below. The fields on this form assist you in completing the declaration segment of the PDS file.

PDS Declaration Segment

Title Pattern Revision Author Company Date					
	08/15/90				
CHIP	ChipName = <input style="width: 80px;" type="text" value="cntr"/>	Device = <input style="width: 100px;" type="text"/>			
P/N	Number	Name	Paired with PIN	Storage	;comment

Enter Header Data. [Press <ESC>=abort, F1=help, F10=save & exit]

The Title field is active when the form appears. You can **either** type a title **or** select a different field. The text and option fields on this form are described below.

- Title, Pattern, Revision, Author, and Company fields can contain up to fifty-nine, 59, characters, including any combination of alphanumeric characters or symbols.
- Date provides today's date automatically, as specified by the operating system, which you can change if you use the ##/##/## format.

- ChipName currently displays the design file name without the extension.

You can specify a new chip name of up to eight, 8, alphanumeric characters.

- Device refers to the type of device for the design.

Options include all PLD and MACH device types. You must specify a device type before saving information and leaving the form.

- P/N identifies the statement as either a pin or node statement.

Options include a blank and two types: pin and node.

- Number requires a pin or node location, which can be **either** a whole number that fixes the location on the device **or**, for MACH devices only, a question mark, ?, that defines a floating location.
- Name requires a pin or node name.
- Paired with pin is an optional node attribute; if used, you must enter the number of the pin to which the node will be paired.²
- Storage refers to the optional storage type.

Options include a blank and three types: Combinatorial, Latched, or Registered. Combinatorial is the default, which is used if this field is left blank.

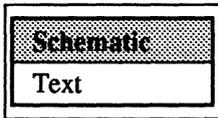
² Refer to Chapter 10, in this section, for details about the following topics: using the question mark to float pin and node locations, naming syntax in pin and node statements, and pairing a node with a pin.

- Comment adds an optional comment to the statement, which is preceded by a semicolon in the PDS file but not on the form.

Options include Input, Output, IO, Clock, and Enable.

After you create and confirm specifications, you're transferred to the text editor and the PDS file is displayed.

Schematic-Based Design Form



The image shows a rectangular form with a double border. The top half of the form has a stippled background and contains the word "Schematic" in a bold, sans-serif font. The bottom half of the form has a plain white background and contains the word "Text" in a bold, sans-serif font.

Press [F2] and then press [Enter] to select a schematic-based format and design name. Two files are automatically created.

- An empty schematic worksheet file is created using the name you specified.
- An empty control file is created using the design name with a .CTL extension, design.CTL; then you're immediately transferred to the control-file form shown next.

Schematic CTL File Information

Title
Pattern
Revision
Author
Company
Date

08/15/90

CHIP ChipName = Device =

Enter Header Data. [Press <ESC>=abort, F1=help, F10=save & exit]

When the schematic data is converted to a PDS format, the information in this form provides the declaration segment of the PDS file.

The Title field is active when the form appears. You can **either** type a title **or** select a different field. The text and option fields on this form are described next.

- Title, Pattern, Revision, Author, and Company fields can contain up to fifty-nine, 59, characters, including any combination of alphanumeric characters or symbols.
- Date provides today's date automatically, as specified by the operating system, which you can change if you use the ##/##/## format.
- ChipName currently displays the design file name without the extension.

You can specify a new chip name of up to eight, 8, alphanumeric characters.

Important: Chip is a reserved word and cannot appear in any field, unless embedded in another word, such as ChipDate.

- Device, is provided where you specify the MACH device type for the design.³

You must specify a device type before saving information and leaving the form.

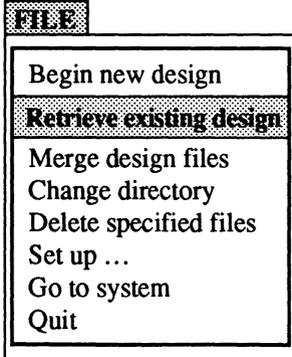
After you create and confirm specifications, you're automatically transferred to OrCAD/SDT III.⁴ A blank worksheet with the name you specified earlier is available along with the AMD-supplied MACH library. You can begin placing symbols and wires to produce the schematic file.

Important: You must enter OrCAD/SDT III in this manner to use the AMD-supplied library for MACH-device designs.

³ Refer to the *PALASM 4* online release notes for a listing of devices with JEDEC support.

⁴ Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about using the schematic editor.

9.2.1.2 Retrieve an Existing Design



Input format: Text

File name:

You select this command and complete the form below to identify an existing design in the current working directory you want to edit or process.⁵

The form that appears is similar to the one you complete to create a new design file.

A rectangular form with a double border. It contains two lines of text: 'Input format: TEXT' and 'File name: *.*'.

This option field specifies the type of design you'll produce. Text refers to a text-based PDS file and is the default.

To edit or process an OrCAD/SDT III schematic-based design, you press [F2] to display the options and select Schematic.

You type the design name in this intelligent text field.

Note: Initially, the name field may be blank or may include *.* , however, once you create or retrieve a file, the form includes the name of the current design.

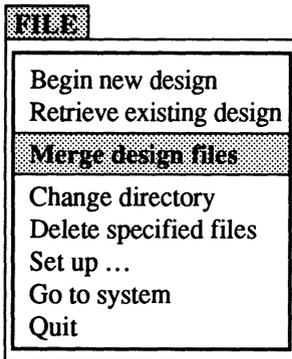
- If the field is blank, you can type a name.
- If the field contains *.* , a list of all file names appears when you press [Enter].

You can enter *.PDS or *.SCH to display a list of specific files to select.

⁵ Refer to discussion 9.2.1.4, in this chapter, for details about changing the current working directory.

After you confirm your specifications, you can choose any command to specify the operation you want to perform. Depending on your working-environment setup, current design information may appear in the lower-right corner of the screen.

9.2.1.3 Merge Design Files



You select the Merge design files command and complete the form below to initiate a process where you can combine design files.⁶ The form that appears is similar to the one you complete to create a new design.

A rectangular form with a double border. It contains two fields: "Input format: TEXT" and "Text output file name: *.*".

Input format: TEXT

You can combine only PDS files. Therefore, the Input format field on this form is a status field that you cannot activate or change.

Text output file name:

You type the name of the output file that will include all combined data in this field; the name must adhere to standard DOS conventions. The output file is stored in the current working directory.

Important: After you confirm the output file name, the merge process is initiated which includes compiling the design file and then the merge screen appears.

Four menus, Files, Editor, Resolution, and Setup, provide all commands for the merge process.

⁶ Refer to Section II, Chapter 4, for guidelines to use when merging design files.

Status fields across the center of the screen identify the output file name, current input file name, and the number of files combined during this session.

Initially, the output file name is specified. However, the input file name is not listed because you have not yet retrieved the first input file.

The detectable-conflicts and pin-summary tables reflect the status of a comparison that's made after you retrieve an input file or resolve conflicts. Messages and prompts appear at the bottom of the screen as usual. The next figure shows the merge-process screen.

MERGE DESIGN FILES

FILES EDITOR RESOLUTION SETUP

Get next input file
Merge files
List combined files
Save
Abandon input
Quit

Output File CNTR.PDS Input File Files Combined 0

Detectable Conflicts	Pin Summary	OUTPUT	INPUT
State 0	Pins 0	0	0
Pins/Nodes 0	Nodes 0	0	0
Strings 0	Floating 0	0	0
Vectors 0	Unreferenced 0	0	0
Conditions 0			
Architecture 0			

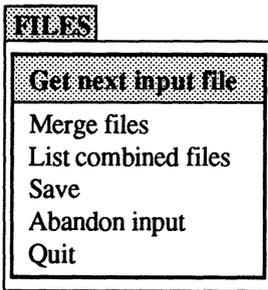
Specify file name for next input file. [<Enter> Select <Esc> Exit]

Initially, the input buffer is empty; the output buffer contains only empty declaration and equation segments.

Important: Following discussions define each merge-process command and all related forms and options. Menus are discussed in order from left to right; commands are discussed in order from first to last.

Files Menu

Get Next Input File



The merge process stores files temporarily in the input and output memory buffers. All commands on this menu, except Quit, operate on files in the memory buffers.

This command is highlighted when you begin the merge process. When you select this command, a form appears with the intelligent text field shown below.

A screenshot of a form with a double-line border. Inside, there is a single-line input field containing the text "*.pds".

You can **either** type a file name **or** display a list of files and select a name from the list.

- If the form contains *.pds; press [F2] or [Enter] to display a list of all PDS files.
- If the form contains *.*; press [F2] or [Enter] to display a list of all files; however, you can only select a PDS file.

In any case, after you confirm the name, the following process is completed.

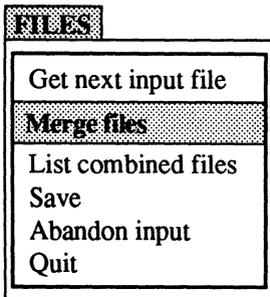
- Design data is loaded into the input buffer; the status line in the center of the screen reflects the name of the input file.
- The design is parsed, expanded, and minimized.

If errors are detected, the input file is abandoned and the input buffer is cleared automatically.

- Data in the input buffer is compared with data in the output buffer.

The pin-summary table reflects the status of the design in the input buffer. If design data is in the output buffer, the detectable conflicts table reflects the number of signal name or pin location conflicts between the two buffers.

Merge Files



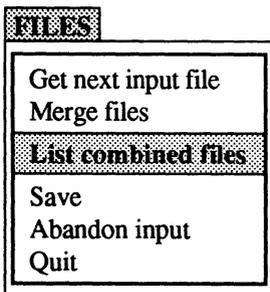
You select this command, after resolving conflicts, to move the input file into the output buffer. Data in the two buffers are combined into a single design in the output buffer; the input buffer is cleared.

The status field in the center of the screen, which identifies the number of files combined during this session, increments by one.

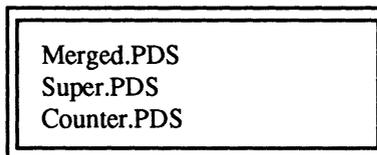
Important: If you initiate the Quit command before merging data in the input buffer with data in the output buffer, a warning appears and asks if you are sure you want to quit. In this case,

- Y confirms you want to quit without merging data.
- N cancels the quit command so you can merge and save the data.

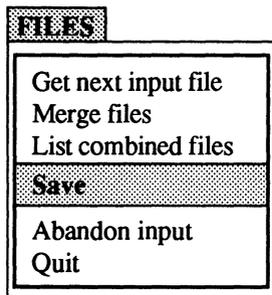
List Combined Files



This command lists the names of all files you've combined during this session. You cannot select or edit any name in the list.



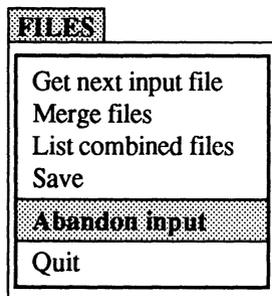
Save



The Save command writes all data in the output buffer to the specified output file. Until you select this command, data resides only in a memory buffer.

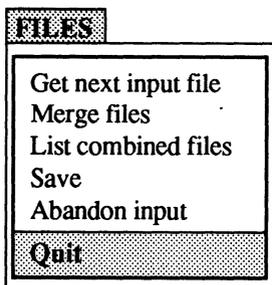
Important: You must merge files to move data from the input buffer into the output buffer. Then save data in the output buffer to write it to the output file.

Abandon Input

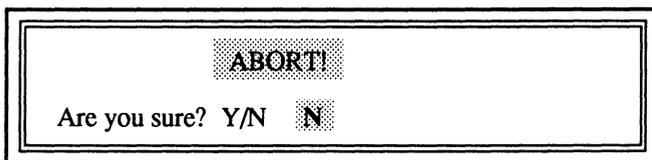


You use this command to clear the input buffer if you find the file is not appropriate to merge with data in the output buffer. The input-file status field in the center of the screen identifies the name of the input file; however, the field is cleared automatically either when the file is abandoned or after merging.

Quit



You use the Quit command to leave the merge process and return to the PALASM environment. When you select this command, you are asked to confirm ending the session.



- Y returns you to the PALASM environment.
- N cancels the Quit command.

Important: If you initiate the Quit command before merging data in the input buffer with data in the output buffer, a warning appears and asks if you are sure you want to quit. In this case,

- Y confirms you want to quit without merging data.
- N cancels the quit command so you can merge and save the data.

Also, if you initiate the Quit command before saving data in the output buffer, a warning states the design has changed since the last save and asks if you are sure you want to quit. In this case,

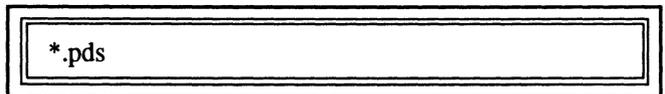
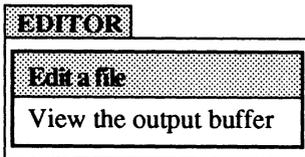
- Y confirms you want to quit without saving changes.
- N cancels the quit command so you can save the data.

Editor Menu

This menu provides two editor commands for the merge process. You cannot edit information in either the input or output buffer. However, you can edit any file and you can view information in the output buffer. The Resolution menu offers a command to edit the pin/node list in the output buffer.

Edit a File

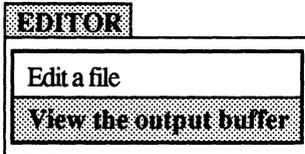
You select this command to correct design errors discovered when you retrieved the input file or to edit header data, device type, or pin locations in a combined design. When you select this command, a form appears containing an intelligent text field, as shown.



You can **either** type a file name **or** display a list of files and select a name from the list. In either case, after you confirm the name, the text editor becomes

available and the designated file is displayed on the screen.⁷ To return to the merge process, you must quit from the text editor as usual.

View the Output Buffer



When you select this command, a view of the combined design in the output buffer appears on the screen. However, you cannot edit in view mode.

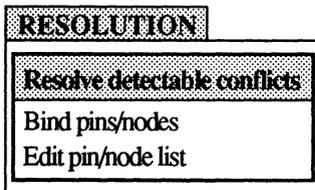
To return to the merge process from view mode, just press [Esc].

Resolution Menu

This menu provides commands to resolve conflicts between designs.

Recommendation: It's important to resolve conflicts before you merge the design in the input buffer with the design in the output buffer.

Resolve Detectable Conflicts



You use this command to display the conflict resolution form. Detectable conflicts occur when signals in the input and output buffer have the same name or pin number. These conflicts can be resolved from the conflict resolution form, shown next.

⁷ Refer to Section V, Appendix A, for command definitions for the AMD-supplied text editor.

CONFLICT RESOLUTION

Output File	Input File	Action	Substitute
GT1	Super.PDS		
CLOCK	CLOCK	RENAME INPUT	CLOCK_001
RESET	RESET	RENAME INPUT	RESET_001

Output File: Pin ? CLOCK COMB
 Input File: Pin ? CLOCK COMB
 RESOLUTION: RENAME INPUT -- In input file change
 'PIN ? CLOCK COMB' to 'PIN ? CLOCK_001 COMB'

This form includes two columns with option fields and two columns with status fields.

- Status fields: Output File and Input File
- Option fields: Action and Substitute

Output File:

This column heading identifies the name under which combined data in the output buffer will be saved.

Each status field in this column identifies a signal name that conflicts with a signal in the input buffer. Only conflicting signals are listed. However, you cannot activate or edit status fields in this column.

Input File:

This column heading identifies the file in the input buffer.

Each status field in this column lists a signal name that conflicts with a signal in the output buffer. Again, you cannot activate or edit fields in this column.

Action

This column identifies the recovery for each signal conflict. The first item in this column is active when the form appears. Possible recovery actions include the following.

- Rename input
- Bind

The default action is to **rename** the signal in the input buffer; this option fills each row when the form appears. The name that will be used appears in the Substitute column.

Important: When you intend to use separate signals, you must rename one.

If you intend to use the same signal, you must bind them together using a common signal name.

To **bind** signals, you

- Press [Tab] to highlight the action field that corresponds to the pertinent conflict, then display the options as usual.
- Select Bind from the list.

In this case, Bind appears in the action field and the name in the output buffer becomes the common name. You can change the common name as explained under Substitute.

Wildcard appears as an action in a field **when** you specify no floating input pins as a setup option **and** two pins are assigned to the same pin location on the device. In this case, a question mark is automatically assigned to the pin location in the input buffer. To restore the pin location specified in the input buffer, you must edit the pin/node list after combining the files.

Important: Wildcard is **not** available on the list of options.

Substitute

The fields in this column identify the name that will replace every instance of the signal name in the input buffer.

- If the action is to **rename** the input, the substitute is based on the naming strategy you specified using the Set renaming strategy command on the Setup menu.
- If the action is to **bind** signals together, the substitute name is taken from the design in the output buffer.

When the action is set to rename, you can change the substitute name by selecting this field and typing a new name.

Status information

Information at the bottom of the form identifies the exact pin or node statements in conflict and how the statement in the input buffer will change. For example, when you **rename** a signal the corresponding message reads as follows.

```
RESOLUTION:  RENAME INPUT -- In input file change
              'PIN ? CLOCK COMB' to 'PIN ? CLOCK_001
              COMB'
```

When you **bind** signals together, the corresponding message reads as shown below.

```
RESOLUTION:  BIND -- pin definitions are identical
```

In either case, the status at the bottom of the form reflects the automatic change. If you alter the action or substitute name, the status won't reflect this until you confirm, leave the field, and return to it.

Bind Pins/Nodes

RESOLUTION

Resolve detectable conflicts

Bind pins/nodes

Edit pin/node list

You use this command to display the Bind form, shown next. You can use this form to bind signals of different names to a common signal name. Initially, this form includes only those signals you bound together using the Bind action on the conflict-resolution form.

BIND			
Output File	Input File	Action	Substitute
GT1 CLOCK	Super.PDS CLK	BIND	CLOCK
Output File: Pin ? CLOCK COMB Input File: Pin ? CLOCK COMB RESOLUTION: BIND — In input file change 'PIN ? CLK COMB' to 'PIN ? CLOCK COMB'			

This form is similar to the conflict resolution form; however the field types differ as follows.

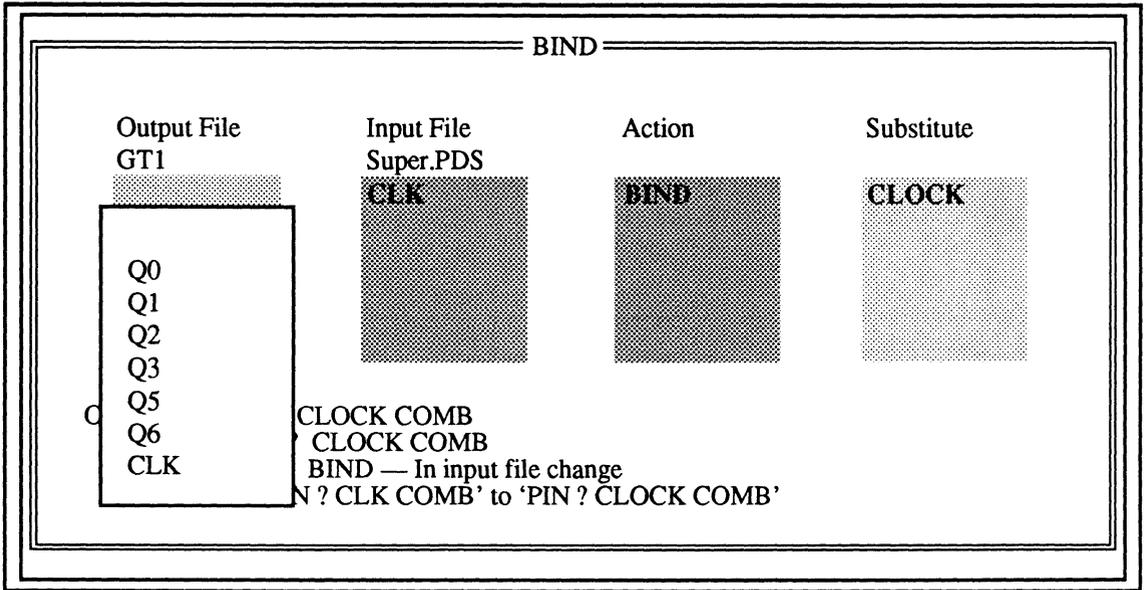
- Option fields: Output File and Input File
- Status fields: Action and Substitute

Output File

This column heading identifies the name under which combined data will be saved.

Each option field in this column identifies a signal in the output buffer that is bound by a common name to a signal in the input buffer.

Blank fields are provided so you can bind signals with different names to a common name. When a blank field is active, you press [F2] to display a list of all signals in the file, then use arrow keys and [Enter] to select a name to fill in the field. An example follows.



You repeat this process with the Input File column. The substitute name is taken from the output buffer.

Input File

This column heading identifies the name of the file in the input buffer.

Each option field in this column identifies a signal in the input buffer that is bound by a common name to a signal in the combined design in the output buffer.

Blank fields are provided so you can specify binding signals with different names to a common name, as described under Output File.

Action

The option field in this column lists Bind when the form first appears. Options for this field include the following.

- Bind
- No action

Bind is the default action. You can select the action field, display a list of options, and select No action to cancel the bind operation for associated signals.

Substitute

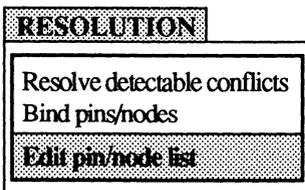
This status field lists the common name that will replace every instance of the original signal name in the input buffer. The common name is taken from the pin in the output buffer. The default substitute name is the one that's used in the output. You cannot edit the substitute name field in this form. However, you can edit the combined design later to change the common name.

Status information

Information at the bottom of the form identifies the exact pin or node statements and how the statement in the input buffer will change. For example,

```
RESOLUTION: BIND -- In input file change  
              'PIN ? CLOCK COMB' to 'PIN ? CLK COMB'
```

Edit Pin/Node List



The image shows a dialog box titled "RESOLUTION". It contains three buttons: "Resolve detectable conflicts", "Bind pins/nodes", and "Edit pin/node list". The "Edit pin/node list" button is highlighted with a shaded background.

This command displays a form that includes the header, device type, and pin statements in the output buffer. This form looks and operates like the new PDS file declaration-segment form discussed under 9.2.1.1 and shown opposite.

You can use this form to **edit** information in the **output buffer**.⁸

For example, the Bind form allows you to treat pins in the input and output buffers as the same pin; however, the common name is taken from the pin in the output buffer.

⁸ Refer to discussion 9.2.1.1 for details about using the text-based design form.

You can edit the combined data in the output buffer after you merge, to choose a new name for a pin. In addition, you can edit header information or the device type.

The header information is taken from the first input file. Headers in all other input files are disregarded.

Important: You can use a question mark, ?, in the number field to specify a floating pin or node location. The storage type and comment fields are optional.

Also: If you enter new pin/node statements and run out of empty fields, just press [F10] to save the current changes and return to the merge process, then select Edit pin/node list again to display the form. Each time you enter this form, 20 empty pin/node fields become available.

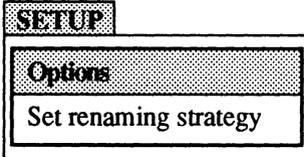
PIN/NODE				
Title Pattern Revision Author Company Date	16 Bit Counter EXCNT2 A Gail ADVANCED MICRO DEVICES 09/02/90			
CHIP	ChipName =	EXCNT2	Device =	MACH110
P/N	Number	Name	Type	Comment
Pin	?	CARRY	REGISTERED	
Pin	?	CLK	REGISTERED	
Pin	?	COUNT	REGISTERED	
Pin	?	Q1	REGISTERED	
Pin	?	Q2	REGISTERED	
Pin	?	Q3	REGISTERED	
Pin	?	Q4	REGISTERED	

Setup Menu

This menu provides commands you use to set up the merge environment.

Options

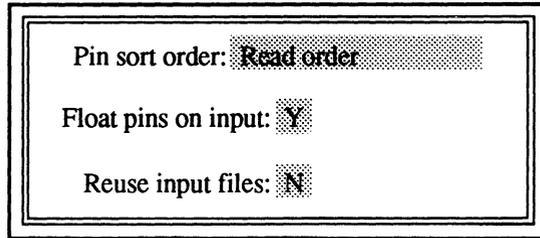
When you select this command the form below appears listing the options you can set.



SETUP

Options

Set renaming strategy



Pin sort order: Read order

Float pins on input: Y

Reuse input files: N

Pin sort order

The pin-sort order field is an option field that determines how signals are listed in pin/node statements when you want to edit the pin/node list. Available options are listed below.

OPTIONS	DEFINITIONS
Read order	List names in the order in which they are read.
Last pin first	List names in reverse read order.
Pin number, name	Sort the list by pin number, then name.
Name	Sort the list alphabetically by name.

Float pins on input

This is a Yes/No text field. Other entries are not accepted.

- Y, the default, specifies floating all pins on input.

In this case, pin numbers specified in the design file are changed to a question mark, ?, to indicate floating.

- N specifies using the pin numbers assigned in the design file.

Recommendation: It is best to float pins on input to eliminate pin location conflicts. However, if you do not float pins on input and there are two pins assigned to the same location, the location in the input file will be floated automatically. You must then edit the pin/node list in the combined file after merging to restore the original pin location.

Reuse input files

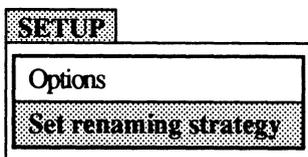
This is a Yes/No text field field. Other entries are not accepted.

- Y lists the names of all files when you use the Get next input file command.

In this case, when you type either *.* or *.PDS into the file name form, followed by [Enter], the resulting list contains the names of all files.

- N, the default, ensures the names of files merged during this session do not appear in the file name list that appears when you get the next input file.

Set Renaming Strategy



This command allows you to redefine the strategy for default substitute signal names. When you select this command, the form below appears.

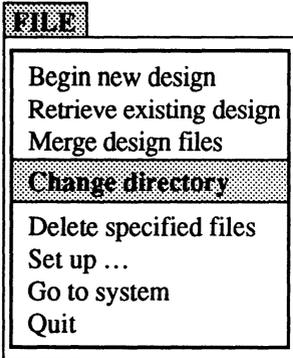
A rectangular form with a double border. Inside the form, the text '\$_#' is displayed.

The field contains the default specification, which means that substitute signal names will be composed of all or part of the existing name, an underscore character, and a three-digit number: name_001.

- \$ is replaced with the original name. When necessary, the name is truncated so the entire resulting name does not exceed 14 characters.

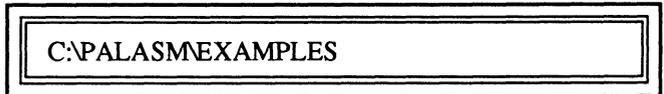
- # ensures that unique names are produced and should be included in any naming strategy. If existing signal names include numbers, these numbers are automatically skipped when substitute names are produced.
- _ allows you to quickly spot substitute names and the numbers assigned.

9.2.1.4 Change Directory



All files are stored in, and retrieved from, the current working directory; all commands operate on the files in the current working directory.

When you select this command to change the current working directory, a form appears with a text field that identifies the path to the current directory.



You can replace all or part of the existing path name with a new one. The new path name must include a valid drive, directory, and subdirectory.

After you confirm the new path, the specified directory becomes the current working directory. Depending on the setup you've defined using the Set up and Working environment commands, the new path may appear in the lower-right corner of the screen.

9.2.1.5 Delete Specified Files

FILE

Begin new design
 Retrieve existing design
 Merge design files
 Change directory

Delete specified files

Set up ...
 Go to system
 Quit

This command initiates a process to delete specified files from the current working directory. When you select this command, a form appears listing all process-related files. Design files are not listed.

- Schematic-process files are created when you convert schematic data to a PDS file.
- Text-process files are created when you compile or simulate a PDS file.
- Output files show various process results you may be interested in viewing.
- Others *, in the left column, is an option field where you can type a specific file extension that's not listed.

DELETE SPECIFIED FILES

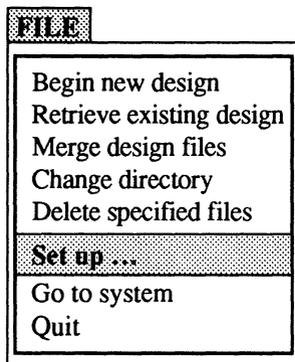
This Utility deletes all files in the current directory with the following extensions when 'Y' is selected.

SHEMATIC PROCESS FILES		TEXT PROCESS FILES		OUTPUT FILES	
-----		-----		-----	
JNL	Y	BAK	Y	PL2	N
JXR	Y	TRE	Y	XPT	N
JNF	Y	TMP	Y	JED	N
FLS	Y	LIS	Y	HST	N
FLP	Y	@??	Y	TRF	N
SRF	Y	LOG	Y	JDC	N
CRF	Y			XRF	N
OXR	Y			BLC	N
Others	*				

Names are identified by file **extension**. The text field beside each extension contains the letter Y, Yes, or N, No; all files marked with a Y will be deleted. The default is to delete all files **except** those listed under Output files, which includes results you may be interested in viewing.

When you confirm the information in this form, the designated files are deleted.

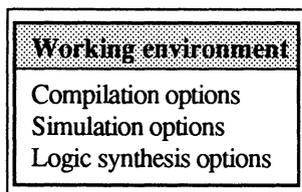
9.2.1.6 Set Up



This command allows you to identify software-environment and process preferences. For example, you can suppress certain forms that might otherwise appear each time you begin compilation or simulation. In addition, you can identify a preferred editor and communication program over those supplied by AMD.

The submenu that appears when you select this command offers additional choices as explained below.

Working Environment



This command is used to specify preferences for your working environment. When you select this command, the form below appears providing text fields that display the specifications currently in effect.

Editor program:	C:\PALASMEXE\ED.EXE
RS-232 communication program:	C:\PALASMEXE\PC2.EXE
Provide compile options on each run:	Y
Provide simulation options on each run:	Y
Display design information window:	Y
Turn system bell on:	N
Generate netlist report:	Y

Editor program:

This field specifies the path name to the text editor you use to create and edit PDS, simulation, and other text files. The default path name identifies the location of the AMD-supplied text editor.⁹

- If you change the path, a preferred editor will be available for viewing and editing files.
- If the path you supply is incomplete or incorrect, the editor will not be found.

RS-232 communication ...

This field provides the path name to the software that's required to communicate with the device programmer when you download the JEDEC file. The default path name identifies the location of the AMD-supplied communication program.

- If you change the path, a preferred program will be used during the download process.
- If the path you supply is incomplete or incorrect, the program will not be found.

Provide compile options ...

This field specifies when to display the form that defines compilation options.

- Y displays the form each time you select **either** the Compile **or** Both command from the Run menu.
- N displays the form only when you select the Set up command from the File menu followed by the Compilation options command from the submenu.

Provide simulation opt ...

This field specifies when to display the form that identifies the simulation-file option.

- Y displays the form each time you select **either** the Simulation **or** Both command from the Run menu.

⁹ Refer to Section V, Appendix A, for a summary of the AMD-supplied text editor commands and operations.

- N displays the form only when you select the Set up command from the File menu followed by the Simulation options command from the submenu.

Display design informa ...

Current design information includes the working directory, input format, design file name, and device type.

- Y displays current information in the lower-right corner of the screen.
- N suppresses the information.

Turn system bell on:

A bell tone can warn you of syntax errors and illegal actions while working with the software.

- Y sounds the tone.
- N suppresses the tone.

Generate netlist report:

A netlist report is generated when schematic data is converted to a PDS file.

- Y generates the report.
- N suppresses the report.

Upon confirmation, you're returned to the Setup submenu. Specifications take effect as soon as you confirm them, though it may not be obvious until you take a particular action.

Compilation Options

Working environment
Compilation options
Simulation options
Logic synthesis options

You select this command to display the form that defines compilation options for the current design, as shown below.¹⁰

COMPILATION OPTIONS			
Log file name:	PALASM.LOG		
Run mode:	AUTO		
Process from	Format: SCHEMATIC File: ORCADDMA.SCH		
Check syntax:	N	Merge mixed mode:	N
Expand Boolean:	N	Minimize Boolean:	Y
Expand state:	N	Assemble:	N

The form includes status, option, and text fields.

- Two **status** fields in the center of the form identify the input format and file name.
- One **option** field, Run mode, allows you to specify either automatic or manual compilation.
- **Text** fields are provided so you can confirm or cancel options that will be used when you specify manual run mode.

Log file name:

All error, warning, and status messages that scroll by during software processes are stored in the execution-log file named in this text field. The information stored in the log is replaced each time you run a new process.

The default file name is PALASM.LOG. To retain additional versions, you can assign a different name using standard DOS naming conventions. To view any but the most recent log, you must use the Other command on the View menu.

¹⁰ Depending on the working environment setup you've specified, the compilation form may appear automatically when you select either the Compile or Both commands from the Run menu.

Run mode:

Auto
Manual

The list associated with this option field provides two choices: Auto and Manual.

Automatic mode performs all functions to process a design and ignores specifications in the lower part of the Compilation Options form.

Manual mode performs only those functions specified on the lower part of this form, though it may result in a less than optimal process and result.

Important: The specifications in the following text fields apply **only** when **manual** run mode is specified.

Check Syntax:

This field specifies whether or not a syntax check is made on the PDS file. Any errors discovered during this check must be corrected before compilation can be completed.

Expand Boolean:

This field specifies expanding Boolean equations in the PDS file. Expansion means all equation definitions are expanded into individual equations.

Expand State:

A compiled PDS file contains only Boolean equations. This field specifies whether or not state-machine descriptions are expanded to Boolean equations.

Merge Mixed Mode:

This field specifies whether or not to merge a design that contains both Boolean and state-machine descriptions.

Minimize Boolean:

Minimization reduces a set of Boolean equations to a sum-of-products form that usually involves fewer product terms or literals. This field specifies minimizing Boolean equations in the PDS file.

Assemble:

Assembly translates information in a .TRE file and produces a JEDEC fuse map file for all PAL and PLS device designs and a MACH report for all MACH-device designs. This option field specifies whether or not assembly is performed.

If you're working on a MACH-device design, a MACH Fitting Options form appears after you confirm options on the Compilation Options form.

MACH FITTING OPTIONS	
OUTPUT:	
Report level	Detailed
SIGNAL PLACEMENT:	
Force all signals to float?	Y
Use placement data from	Design file
Save last successful placement	<F3>
Press <F9> to edit file containing	Last successful placement
FITTING OPTIONS:	
When compiling	Run all until first success

The MACH Fitting Options form specifies options unique to fitting MACH-device designs. The form includes status, option, and text fields.

- **Option** fields allow you to specify preferences for output reports, signal placement, and fitting options.
- One **text** field, Force all signals to float, allows you to specify either yes or no.
- A **status** field in the center of the form indicates you want to save the last successful placement.

Report level

This option field provides two report choices for MACH-device designs; the default is Detailed.

OPTIONS	DEFINITIONS
Brief	Suppresses information
Detailed	Provides all available data on the fitting process

Force all signals to float?

This text field identifies whether the pin and node locations specified in the design file are used or ignored. If you type a Y in this field, the design-file placement is ignored and all pin and node locations are left floating; the software chooses locations automatically.

Use placement data from

This option field allows you to specify the source of the signal-placement data to be used during the next fitting process.

OPTIONS	DEFINITIONS
Design file	Use the pin/node statements in the PDS file.
Last successful placement	Use data in the .PLC file, from the last successful placement.
Saved placement	Use data in the .BLC file saved by pressing [F3] after an earlier successful fitting process.

Note: You can override any of these placement options by typing the letter Y in the Force all signals to float field.

Save last successful placement

Data generated during the last successful fitting process is automatically stored in a file named after the design with a .PLC extension: design.PLC. The PLC file is overwritten during each successful fitting process.

This status field indicates you can permanently store the last successful placement in a file, named after the design with a .BLC extension. Press [F3] after a successful fitting process to create this file. This field cannot be selected.

Press [F9] to edit file containing

You can edit the results of a successful placement to use during the next fitting process. For example, you can edit a pin placement to suit specific design constraints. This option field specifies which results are displayed in the text editor when you press [F9].

OPTIONS	DEFINITIONS
Last Successful	Edit the PLC file, which contains the results of the last successful placement.
Saved Placement	Edit the BLC file, which contains the results of an earlier successful placement saved by pressing [F3].

The default is to display results in the PLC file from the last successful placement. This form must be visible when you press [F9].

When compiling

This option field allows you to specify one of four fitting strategies; the default is Run until 1st success: STD.

OPTIONS	DEFINITIONS
Use all fitting options	Run all possible combinations; do not stop on first success.
Run until 1st success: STD	Run all possible combinations; stop at first success. Does not execute extra macrocell iterations.
Run until 1st success: EXTRA	Run all possible combinations, including extra macrocell iterations. Stops at first success.
Select one combination.	Choose a placement or resource specification from a new form that appears.

See the **PALASM 4 Release Notes** that accompany this software for more information on the Run until 1st success option.

After you choose the Select one combination option, a form appears with additional specifications. All fields on

this form are text fields where you can enter Y, Yes, or N, No. The default in each case is Y, which enables the corresponding item.

Maximize packing of logic blocks?	Y
Expand small PT spacing?	Y
Expand all PT spacing?	Y

Maximize packing of logic blocks?

This field specifies packing as many macrocells as possible into each logic block versus holding some macrocells in reserve.

- Y places as many macrocells as possible into each block.
- N holds some macrocells in reserve.

Expand small PT spacing

This field allows more flexibility in choosing macrocell placements and switch-matrix paths for feedback signals.

- Y leaves adjacent macrocells empty when functions with less than four product terms are placed.
- N disables the option to leave adjacent macrocells empty.

Expand all PT spacing

This specification provides extra switch-matrix resources for intrablock routing. Additional resources are needed for designs that contain many inputs that feed multiple blocks. These resources can be reserved during signal placement by leaving adjacent macrocells empty.

- Y skips one macrocell between each placed signal.
- N disables the option to skip one macrocell between each signal.¹¹

¹¹ Refer to Section II, Chapter 5, for further details regarding these three fitting strategies.

Simulation Options

Working environment
Compilation options
Simulation options
Logic synthesis options

When you select this command for non-MACH-device designs, a form appears that allows you to define where simulation commands are stored.

Use auxiliary simulation file: N

When you select this command for MACH-device designs, the form that appears has an additional option field. This additional field allows you to specify the source of the signal placement data to be used during simulation and test vector generation.

Use auxiliary simulation file: N
Use placement data from: Design file

Use auxiliary simulation file:

Simulation commands can be stored in a separate auxiliary file, named after the design with a .SIM extension. Though you can store commands separately anytime, it is particularly important to do so in the following instances.

- When you merge multiple PDS files into a single MACH-device design, the simulation segments are automatically removed.

You can use existing simulation commands within each design as a guide to produce a separate auxiliary simulation file.

- When creating schematic-based designs for MACH devices, no simulation commands appear in the resulting PDS file.

Subsequent debugging and compilation overwrites the resulting PDS file so it's best to store simulation commands in a separate file.

Use placement data from:

This option field allows you to identify the source of the signal placement data needed to generate test vectors during simulation. For MACH-device designs, test vectors will not be generated if signal placement data is not available. You can choose from three options.

OPTIONS	DEFINITIONS
Design file	Use the pin/node statements in the PDS file.
Last successful placement	Use data in the .PLC file, from the last successful placement.
Saved placement	Use data in the .BLC file saved by pressing [F3] after an earlier successful fitting process.

When selecting one of the above options, the following considerations apply.

- If the design file specifies any pins as floating, use this option field to select the placement data in either the .PLC or the .BLC files.
- If the design file specifies any pins as floating and you select the Design file option, test vectors will not be generated during simulation unless you first back annotate signal placement data from either the .PLC or .BLC files.

Logic Synthesis Options

Working environment
Compilation options
Simulation options
Logic synthesis options

This command allows you to specify preferences for pairing, gate splitting, register optimization, polarity, and treatment of unspecified default conditions.

When you select this command, a form appears that contains text and option fields that display specifications currently in effect.

LOGIC SYNTHESIS OPTIONS

Use automatic pin/node pairing?	N
Use automatic gate splitting?	N . . . if 'Y', Max = 4
Optimize registers for D/T-type	Best type for device
Ensure polarity after minimization is	Best for device
Use 'IF-THEN-ELSE', 'CASE' default as	Don't care

Use automatic pin/node pairing?

This text field defines whether pairing a node with a pin is enabled or not.

- Y specifies automatic input and output pairing.
- N disables automatic pairing.¹²

Use automatic gate splitting?

This text field allows you specify whether or not product terms are allocated from adjacent macrocells in a different block of the device during compilation.

- Y enables gate splitting and allows you to specify up to the maximum number of product terms that can be allocated between MACH blocks.
- N disables gate splitting.

Max =

This option field defines the total number of product terms that can be allocated between adjacent macrocells in different blocks when automatic gate splitting is enabled. Options include the following; 4 is the default.

OPTIONS	DEFINITIONS
4	Four product terms
8	Eight product terms
12	Twelve product terms
16	Sixteen product terms

Each macrocell contains four product terms. Therefore, gate width is defined as the number of product terms

¹² Refer to Chapter 10 for details about pairing a node with a pin.

rounded up to the nearest multiple of four. The maximum gate width is device dependent.

- A maximum gate width of 12 product terms is allowed for MACH 1 device designs.
- A maximum gate width of 16 product terms is allowed for MACH 2 device designs.

Optimize registers for D/T-type

This option field allows you to specify which register type is required for the design. You can choose from four options.

OPTIONS	DEFINITIONS
As specified in design file	Leave design as specified.
D, change all to D-type	Convert flip-flops to D-type.
T, change all to T-type	Convert flip-flops to T-type.
Best type for device	Convert flip-flops first to one type then to the other, compare results, then choose best type based on utilization.

Ensure polarity after...

This option field allows you to specify the polarity required for the design. You can choose from four options.

OPTIONS	DEFINITIONS
As specified in design file	Leave as specified in the design.
Best for device	Use both active high and active low, compare results, choose the one that results in fewest product terms after minimization.
Low, active low	Convert to active low polarity.
High, active high	Convert to active high polarity.

Use 'IF-THEN-ELSE', 'CASE'...

This option field allows you to specify how the software treats default values for the IF-THEN-ELSE and CASE statements. You can choose from two options; Don't care is the default.

OPTIONS	DEFINITIONS
Don't care	Unspecified default conditions are assumed to be don't care.
Off	Unspecified default conditions are assumed to be false.

The don't-care option requires you specify both the on and off sets. The off option requires you to specify only the on sets; the software assumes all other conditions to be off.

You may lose signals from the design If you select the Don't care-option and do not specify all of the default conditions. If the software treats these signals as don't care, they will be eliminated from the design during logic reduction.

Important: When translating designs created with PLPL, you must select the Off option because PLPL treats unspecified default conditions as false.

Use fast minimization?

This option allows the user to run an abbreviated version of the new Minimizer first introduced with PALASM 4 version 1.4. If it is OFF, a more exhaustive minimization is performed. Turn it ON only if an *Out of memory* error is generated or Minimizer compilation times are unusually long, change the fast Minimizer may produce equations with more product terms than the standard Minimizer did.

9.2.1.7 Go To System

FILE

Begin new design
Retrieve existing design
Merge design files
Change directory
Delete specified files
Set up ...

Go to system

Quit

This command temporarily transfers you to the operating system. Once there, you can peruse directories and use any other operating-system commands as usual.

To leave the operating system, just type the word `exit` and press [Enter]. You're returned to the PALASM environment.

9.2.1.8 Quit

FILE

Begin new design
Retrieve existing design
Merge design files
Change directory
Delete specified files
Set up ...
Go to system

Quit

The Quit command transfers you to a confirmation form before exiting the PALASM environment.

Are you sure? Y/N **N**

- Y exits the PALASM environment and displays the operating system.
- N returns you to the PALASM environment.

Important: Pressing [Esc] when a top-level menu is displayed initiates the Quit command automatically.

9.2.2 EDIT MENU

EDIT

Text file
Schematic file
Control file for schematic design
Auxiliary simulation file
Other file

The Edit menu provides two kinds of commands that operate on the specified design in the current working directory.

- **Schematic editor command**
Schematic file
- **Text editor commands**
Text file
Control file for schematic design
Auxiliary simulation file
Other file

Important: All commands on this menu operate on the current specified design.¹³

9.2.2.1 Text File

EDIT

Text file
Schematic file
Control file for schematic design
Auxiliary simulation file
Other file

This command transfers you to the text editor and loads the PDS file you specified using the Retrieve existing design command on the File menu. You can edit information in this file as usual.¹⁴

- If this is a new PDS file, some data may have been entered using the PDS declaration-segment form when the design was created.
- If the PDS file was converted from schematic data, information in the declaration segment is derived from the control file; equations were produced during either compilation or conversion.

When you leave the editor, you're returned to the PALASM environment.

¹³ Refer to discussions 9.2.1.1 and 9.2.1.2 for details about creating and retrieving designs. Refer to 9.2.1.4 for details about changing the current working directory.

¹⁴ Refer to Section V, Appendix A, for details about the AMD-supplied text editor.

9.2.2.2 Schematic File

EDIT

Text file

Schematic file

Control file for schematic design

Auxiliary simulation file

Other file

This command transfers you to the OrCAD/SDT III editor and loads the top-level schematic from the current specified design.

All commands and options in OrCAD/SDT III operate as usual.¹⁵ When you leave OrCAD, you're returned the PALASM environment.

Important: Any device type you specify in the schematic is ignored. The device type must be specified in the control file for the schematic.

9.2.2.3 Control File for Schematic Design

EDIT

Text file

Schematic file

Control file for schematic design

Auxiliary simulation file

Other file

This command transfers you to the control-file form for the schematic-based design, as discussed under 9.2.1.1. You can edit any field in the form to change information in the declaration segment of the resulting PDS file.

Important: The device type must be specified in the control file for the schematic. Any device type you specify in the schematic is ignored.

Also: If you change the device type in the control-file form, the information in the lower-right corner of the screen is not updated until you run the next process.

When you leave the form, you're returned to the PALASM environment.

¹⁵ Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about using the schematic editor.

9.2.2.4 Auxiliary Simulation File

EDIT

Text file
Schematic file
Control file for schematic design
Auxiliary simulation file
Other file

This command transfers you to the text editor and loads the auxiliary simulation file for the current design. If a file named with a .SIM extension does not exist in the current directory, a blank file becomes available so you can enter simulation commands.¹⁶

It's a good idea to produce and store simulation commands in a separate file under certain circumstances, for the following reasons.

- When you combine PDS files into a single design, the simulation segment is stripped out of the combined PDS file.
- When you enter a design as a schematic, each time you compile the design a new PDS file is produced so any simulation commands you enter are lost.

In either case, name the simulation file after the design and include a .SIM extension.

When you leave the editor, you're returned to the PALASM environment.

9.2.2.5 Other File

EDIT

Text file
Schematic file
Control file for schematic design
Auxiliary simulation file
Other file

Use this command to identify a specific file to view or edit. When you select this command, a form appears with a text field so you can specify the name of the file.

<input type="text" value="*.*"/>

The intelligent text field in this form allows you to proceed using one of two methods.

¹⁶ Refer to Section V, Appendix A, for details about the AMD-supplied text editor.

A. Type the complete file name.

When you confirm the name, the file is loaded into the appropriate editor and made available on screen.

or

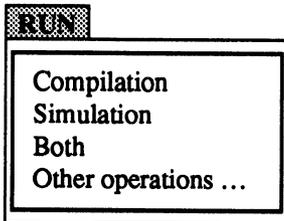
B. Display a list of files using one of the techniques below.

- Press [Enter] to display a list of all files in the current directory.
- Type part of a name, such as design.*, to display a list of specific files, such as all files relating to the named design.
- Type a different drive or directory path to display a list of files elsewhere.

In any case, once you select a name from the resulting list, you're transferred to the appropriate editor and the file is automatically loaded. When you leave the editor, you're returned to the PALASM environment.

9.2.3 RUN MENU

This menu provides a list of operations you can perform on the current design.



- **Primary commands**
Compilation
Simulation
Both

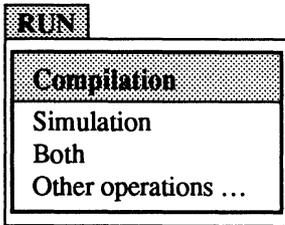
- **Secondary command**
Other operations ...

The Other operations command displays a submenu of commands that perform secondary tasks, as discussed under 9.2.3.4.

Important: All commands on this menu operate on the current specified design.¹⁷

Discussions below explain each command on the Run menu.

9.2.3.1 Compile



This command initiates the compilation process. Depending on the working environment options you specified, forms that define compilation and MACH-fitting options may appear automatically.¹⁸

In any case, a window opens when the process begins and messages scroll by to keep you informed.

Schematic designs

- A. Certain OrCAD utilities are run to check schematics for electrical design-rule violations and the verified schematic is converted into a PDS file.

All designs

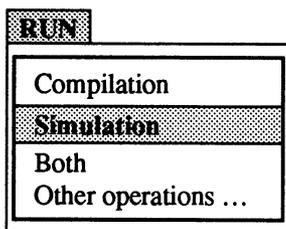
- B. A syntax check is performed on the PDS file.
- C. Equations are expanded, then minimized.
- D. Assembly procedures are completed for PLD designs; the fitting process is performed for MACH-device designs.

¹⁷ Refer to discussions 9.2.1.1 and 9.2.1.2 for details about creating and retrieving designs. Refer to 9.2.1.4 for details about changing the current working directory.

¹⁸ Refer to discussion 9.2.1.6, Set Up, for details about compilation options.

The device pin out and JEDEC files are produced if all processes are successful. Other files are also produced.

9.2.3.2 Simulation



This command verifies design logic using commands placed in either the simulation segment of a PDS file or in an auxiliary simulation file. However, no timing verification is done.

Depending on the working environment options you've set, a form may appear asking if you're using an auxiliary simulation file.

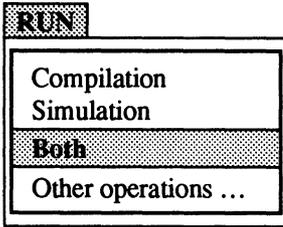
- Y indicates you are using a separate simulation file.
- N, the default, indicates simulation commands reside in the PDS file.

When the process begins, a window opens and messages scroll by to keep you informed. Two types of files are produced during simulation that can help you debug your design.

- Simulation data
- History Waveform display

If the design has not been compiled before selecting this command, it is first compiled, then simulated.

9.2.3.3 Both



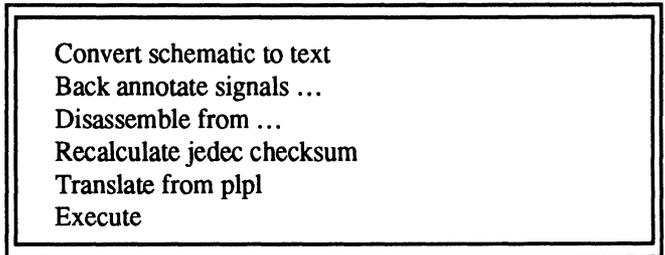
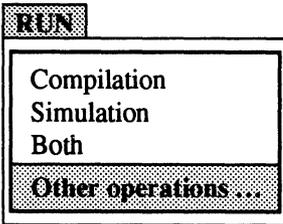
This command saves time when you want to compile and simulate the design at once. Depending on your design and set up options, several forms may appear automatically: compilation, MACH fitting, and simulation options.

When the process begins, a window opens and messages keep you informed, as usual.

- All compilation functions are performed and output files are produced unless errors occur.
- Simulation is performed and the simulation-results files are produced.

This command displays a submenu that lists secondary commands you can use to **either** debug a design or recover a lost design.

9.2.3.4 Other Operations



Discussions below define each command on the submenu.

Convert Schematic to Text

Schematic conversion ordinarily occurs each time you compile the design. However, this command converts data from the current specified schematic into a PDS file without compiling the design.

When the conversion starts, a window opens and messages scroll by. A single PDS file is produced; however, a syntax check is not performed.

Back Annotate Signals

This command pertains only to MACH-device designs. You use this command to display a list of options that indicate the source for signal-placement data. The data from the specified source replaces existing signal locations in the PDS file and pin out report.¹⁹

OPTIONS	DEFINITIONS
Change all to floating	Ignore locations in the PDS file and float all pins and nodes.
Use last successful placement	Replace the locations in the pin/node statements in the PDS file with those from the PLC file.
Take from saved placement	Replace the locations in the pin/node statements in the PDS file with those from the BLC file.

- The PLC file is created during the last successful placement; this file is overwritten during each successful fitting process.
- The BLC file is created when you press [F3] after a successful placement and contains information from the PLC file.

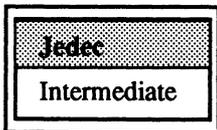
When you select a command from the submenu, the process is initiated. Upon completion, a window opens and provides status information.

Note: If the software detects an error in mapping the signal placement data into the .PDS file, the results of the back annotation will be stored in a separate file with the name design.PBK. For example, defining unused signals in the PDS file or using an illegal number will create this condition.

Disassemble From

This command displays a submenu with the two choices discussed below.

¹⁹ Refer to discussion 9.2.1.6, Set Up, Compilation options, for details about signal back annotation on the MACH fitting options form.



The Jedec command converts JEDEC fuse data into Boolean equations, which is useful to reconstruct a design for which other files are missing. When you select this command, the form below appears with two text fields and an option field.

Input file name:	UDCNTR.JED
Output file name:	UDCNTR.PL2
Device name:	MACH110

Input file name:

This field provides the name of the JEDEC file, which corresponds to the currently specified design name followed by a .JED extension. A name in this field does not indicate the corresponding file exists. You can enter a new name to use a different file as input. You can enter *.JED to display a list of all JEDEC files in the current working directory.

Output file name:

This field names the Boolean equation file created during disassembly. Again, the name matches the design followed by a .PL2 extension. You can enter a new name to store the results in a different file.

Device name:

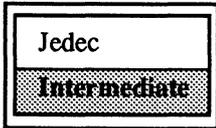
This option field allows you to select the device type corresponding to the original design. JEDEC disassembly is not supported for all devices. You can only disassemble designs created for the devices on the option list, displayed by pressing [F2] in this field.

Once you confirm specifications in the disassembly form, the process is initiated.

Note: When JEDEC fuse data is converted to Boolean equations, the pin names, comments, and simulation vectors in the original PDS file are removed.

Also: Entry formats, such as state machine, waveform, and truth tables, are converted to Boolean equations.

When finished, a Boolean equation output file is stored in the current directory under the name you specified.



The Intermediate command disassembles the intermediate file, PALASM2.TRE, so you can view the results of the minimization and expansion processes in a sum-of-products format.

Recalculate JEDEC Checksum

This command recalculates the checksum in the JEDEC test-data file. When you select this command, the form below appears.

Input file name:	UDCNTR.JED
Output file name:	UDCNTR.JDM
Device name:	MACH110

Input file name:

This field contains the name of the JEDEC checksum file in the current directory, which corresponds to the current specified design name with a .JED extension. A name in this field does not indicate the corresponding file exists. You can enter a new name to use a different file as input.

Output file name:

This field contains the default name of the output file that will be created. Again, the name matches the design followed by a .PDS extension. You can enter a new name to store the results in a different file.

Device name

This option field allows you to select the device type corresponding to the original design. Checksum recalculation is not supported for all devices. You can only recalculate the checksum for the devices on the option list, displayed by pressing [F2] in this field.

Once you confirm specifications in the recalculation form, the process is initiated. When finished, a JDM file is stored in the current directory under the name you specified.

Translate from PLPL

This command converts a PLPL design file into a PDS file. When you select this command, a form appears so you can enter the input and output file names.

Input file name:	*.pld
Output file name:	PALASM.PDS

Input file name:

This text field provides the name of the PLPL file, which corresponds to the current specified design name followed by a .PLD extension. A name in this field does not indicate the corresponding file exists. You can enter a new name to use a different file as input.

Output file name:

This text field names the file that will be produced. Again, the name matches the design followed by a .PDS extension. You can enter a new name to store the results in a different file.

Once you confirm specifications in the translation form, the process is initiated. When finished, a new PDS file is stored in the current directory under the name you specified.

Execute

This command sets OrCAD/SDT III configuration options, such as the printer or plotter drivers, library prefix, etc. When you select this command, a form appears requesting a parameter.

draft.exe /c[parameter]

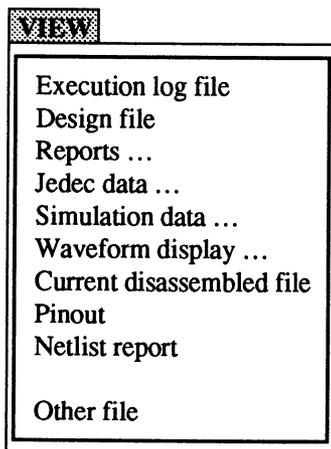
- Press [Enter] to display a blank parameter form.

[Parameter]

- Press [Enter] a second time to invoke OrCAD's configuration program.
- Change options as usual, then update the information and quit.

When you quit, you're returned to the PALASM environment.²⁰

9.2.4 VIEW MENU



The View menu provides commands to display all files related to the currently specified design. However, you cannot edit files in view mode.

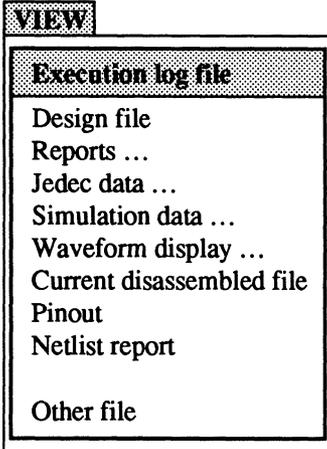
- You can press [Esc] to dismiss the file and return to the View menu.
- You can use the Go to system command on the File menu to suspend to the operating system, then use the print command to print a file. Type exit and press [Enter] to return to the PALASM environment.

Brief definitions of available commands and other information about each file are provided below.

²⁰ Refer to the *OrCAD/SDT III Schematic Design Tools* manual for details about specifying options using DRAFT/C.

9.2.4.1 Execution Log File

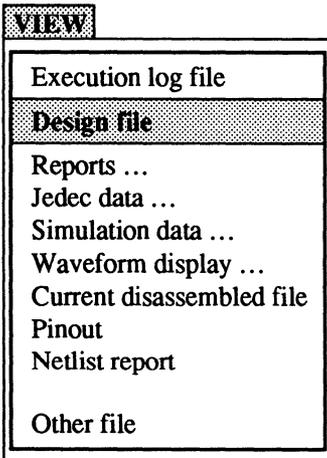
This command displays the log file that contains all messages generated by the last software process run on the current design.



The log file is rewritten each time you initiate a new process. All error, warning, and status messages appear as they are recorded.

9.2.4.2 Design File

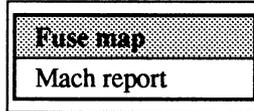
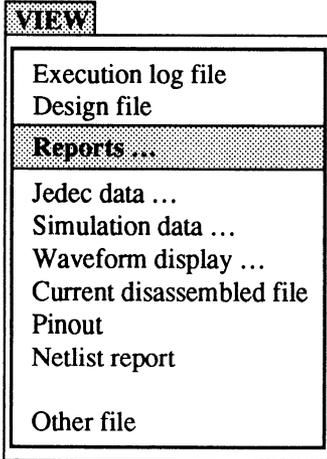
This command displays the current design file in the appropriate editor, either the text editor or the schematic editor. You can edit the file and print it using the associated editor commands.



If you retrieved a schematic design and want to view the PDS version created during compilation, you must use the Other file command on the View menu.

9.2.4.3 Reports

This command provides a submenu that lists the names of the files produced either during the PLD assembly or MACH fitting-process.



Fuse Map

This file provides both programmed and unprogrammed fuse data generated during the assembly process for non-MACH device designs.

- The symbol for a programmed fuse is –.
- The symbol for an unprogrammed fuse is X.

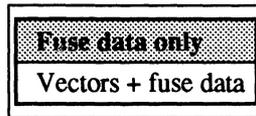
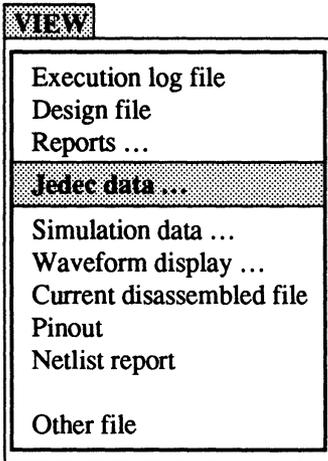
Mach Report

This file contains placement, block, and device pin-out information for MACH-device designs. The content of this file is controlled by the output-report specification on the MACH Fitting Options form, as shown below and discussed under 9.2.1.6.

...
Report level Detailed
...

9.2.4.4 JEDEC Data

This command displays a submenu that lists commands to view JEDEC fuse and vector data.



Fuse Data Only

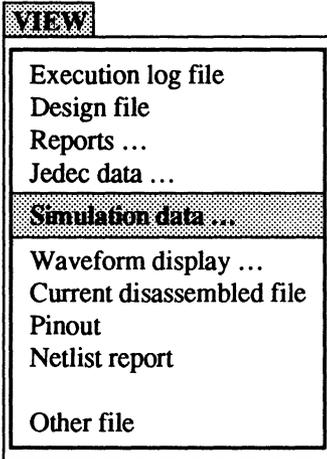
The fuse data file is created during the assembly or fitting process. The information in this file is in a machine-readable format that you can download to program a device.

Vectors + Fuse Data

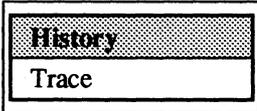
Vectors are added to the fuse file after a successful compilation and simulation. Information in this file includes the following.

- Fuse data from the JEDEC fuse data file
- Test vectors added during simulation that can be used to verify a device on a programmer

9.2.4.5 Simulation Data



History



This command displays a submenu of commands to view the simulation history and trace files in a text format.

The following information is presented in each file.

- Each instance of **g** represents the SETF command in the simulation file.
- Each instance of **c** represents a complete clock cycle, which is defined by the CLOCKF command in the simulation file.

The history file contains the behavior of all signals defined in the pin statements. Information in this file is divided into two columns. You can track values using the cursor, which is displayed as a thick vertical bar.

- The left column lists pin names for each pin listed in the declaration segment of the PDS file.
- The right column records the simulation results in text-format waveform.

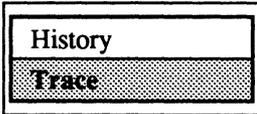
H = high

L = low

X = undefined

Z = output disabled

Trace



The trace file contains the behavior of only those signals specified using the Trace command in the simulation segment or file. Again, you can track values using the cursor, which is displayed as a thick vertical bar.

Information is displayed in the same text format as the history file.

- The left column provides the pin names you specified using the Trace command.
- The right column records high and low signals as a text-format waveform.

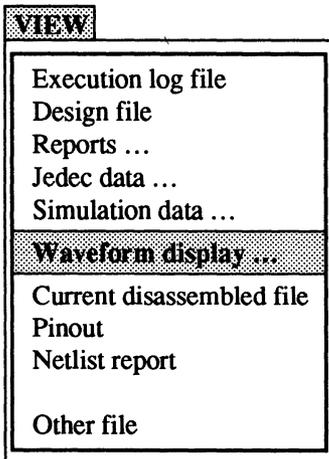
H = high

L = low

X = undefined

Z = output disabled

9.2.4.6 Waveform Display

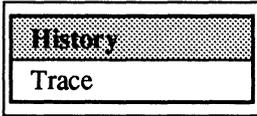


This command displays a submenu that lists the simulation waveform files.

The following information is presented in each file.

- Each instance of g represents the SETF command in the simulation file.
- Each instance of c represents a complete clock cycle, which is defined by the CLOCKF command in the simulation file.

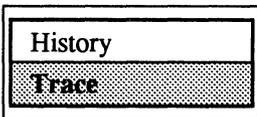
History



This file displays the simulation history of **all signals** defined in the pin statements in a graphic format. Again, you can track values using the cursor, which is displayed as a thick vertical bar.

- The left column provides pin names for each pin listed in the declaration segment in a PDS file.
- The right column records high and low signals graphically.

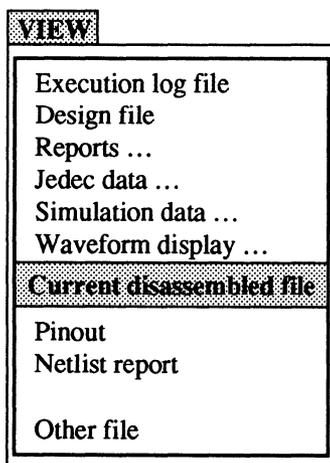
Trace



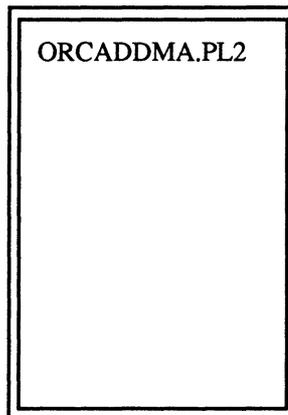
This file displays only the simulation **trace data** in a graphic format. Again, you can track values using the cursor, which is displayed as a thick vertical bar.

- The left column provides names of the pins you specified using the **Trace command** in the simulation segment or file.
- The right column displays the simulation results graphically.

9.2.4.7 Current Disassembled File



This command lists the current disassembled file, if any, which contains the results of disassembling either the intermediate .TRE file or the JEDEC file. After selecting the name from the list, the file appears on the screen.



9.2.4.8 Pinout

VIEW

Execution log file
Design file
Reports ...
Jedec data ...
Simulation data ...
Waveform display ...
Current disassembled file

Pinout

Netlist report

Other file

This file provides the device pin-out information in graphic form. For MACH-device designs only, this file also includes pin and node assignments for the specified device following a successful compilation.

Note: For MACH-device designs, if you specified all pins as floating, you must first back annotate the PDS file with the signal placement that's created during the fitting process, as discussed in 9.2.3.4. Otherwise, the pinout report shows all pins as NC, no connect.

9.2.4.9 Netlist Report

VIEW

Execution log file
Design file
Reports ...
Jedec data ...
Simulation data ...
Waveform display ...
Current disassembled file
Pinout

Netlist report

Other file

The netlist report is an intermediate file created when schematic data is converted to PDS format. This text file identifies the following.

- Signal names as they appear in the schematic
- Reference designators after annotation by OrCAD utilities
- AMD-supplied macro type
- Sheet information and X and Y locations on the sheet

Note: This report is **not** created if you type the letter N beside the Generate netlist report field on the working-environment form.

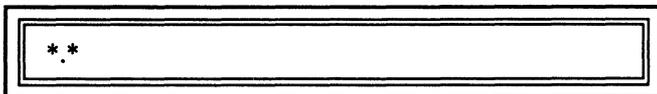
...

Generate netlist report N

9.2.4.10 Other File



This command allows you to view files not available through explicit commands on the View menu, including those in other directories. When you select this command, the form below appears.

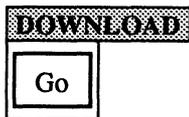


The intelligent text field in this form allows you to display a file using one of three methods.²¹

- Press [Enter] to display a list of all files in the current directory.
- Type part of a name, such as design.*, to display a list of specific files, such as all files relating to the named design.
- Type a different drive or directory path to display a list of files elsewhere.

In any case, when you select a name from the list the file is displayed.

9.2.5 DOWNLOAD MENU



The Download menu provides access to the communication program specified in the working-environment form discussed under 9.2.1.6.

You select this command to download a JEDEC file to a device programmer via the communications software. The Go command initiates loading the JEDEC file to a device programmer using one of the following packages.

²¹ If the specified file is a schematic, the file is loaded and displayed in the OrCAD/SDT III editor. In this case, you can edit or print the schematic, as usual.

- The AMD-supplied PC2 programmer communications software.

```
...
RS-232 communication pro...:  C:\MACH\EXE\PC2.EXE
...
```

- The programmer communications software that's specified in the field of the working-environment form shown below.

```
...
RS-232 communication program: C:...
...
```

When you select this command, the communications software screen appears. When you leave the communications environment, you're returned to the PALASM environment.

9.2.6 DOCUMENTATION MENU

The Documentation menu provides access to online help. Three topics are available.

Index of topics
Language reference
Help on errors

Important: You select items from a menu using arrow keys to highlight the item and the [Enter] key to select it. In addition, you can use a mouse, in which case, you highlight the desired item and double click the left mouse button. **However**, you **cannot** select a menu item by typing the first letter.

9.2.6.1 Index of Topics

When you select this command, the on-line help index screen appears. Available topics are listed in a sub-menu.

Topics include release notes, system configuration, troubleshooting techniques, etc.

When you select a topic from the submenu, such as release notes, the information appears and you can use PgUp or PgDn to scroll through the file.

- You press [Esc] or select Resume to return to the Online Help Index.
- You select Navigate to obtain error recovery or language reference information.
- You press [Esc] or select Quit and respond Yes to the prompt to return to the PALASM menu.

9.2.6.2 Language Reference

When you select this command, available language-construct topics are listed in a sub-menu.²²

Important: You select items from a menu using arrow keys to highlight the item and the [Enter] key to select it. In addition, you can use a mouse, in which case, you highlight the desired item and double click the left mouse button. However, you **cannot** select a menu item by typing.

When you select a construct, a brief overview appears, including the correct syntax and supported devices. Additional topics are listed across the top of the screen; status and prompt messages appear at the bottom of the screen. To select a topic, just move the cursor to the right or left and press [Enter].

The following discussions describe each topic that's displayed.

²² These topics are also described in Section IV, Chapter 10.

Overview

A brief overview of the construct includes a sample syntax and list of supported devices.

Syntax

A repeat of the syntax and a simple example of its use in a PDS file are shown.

Definitions

Descriptors following the keyword are usually defined in their order of appearance in the statement. Definitions include exact syntax requirements and details about the results of using the construct when applicable.

Note: If more than one screen is required to display the information, use the PgUp and PgDn buttons provided in the upper-right corner. You select a button by pressing [Enter].

Use

This topic provides details about using the construct.

Related Topics

This command displays a menu of related topics you can refer to for additional information. Press [Enter] to go to a related construct. If none are listed a message informs you.

Previous Menu

This command returns you to the language constructs menu. To return to the documentation menu, either

- Press [Esc] once and respond Yes to quit online help, or
- Select Quit and respond Yes.

9.2.6.3 Help On Errors

This command searches the execution log file that's generated during compilation for error and warning messages. The log file appears and the first message is displayed.

- Select Next message for the next error explanation and recovery.

The total number of messages and the currently displayed message number appear in the lower-right corner. When the number of messages is greater than one, two additional commands appear: Next message, Previous message.

- Select Prev message to display the previous error explanation and recovery.
- Select the Browse file command and use the PgUp and PgDn buttons to scroll through the file.
- Select Enlarge recovery to view a detailed explanation and suggested recovery.
- Select Other to view additional error files or navigate to the Index of Topics or the Language Reference.
- To return to the Documentation menu, select either the Quit command or press [Esc], then respond Yes to confirm.

9.2.7 [F1] FOR HELP

Pressing [F1] transfers you to the online help and provides information about the use or function of the currently selected menu, command, option, text, or status field.

If more than one page is required to display all of the information, PgDn and PgUp buttons are provided in the lower right and upper-left corners.

- Use the up or down arrow keys to activate a button and press [Enter] to select or use the [Page Up] [Page Down] keys.
- Press [Esc] to return to PALASM.

CHAPTER 10

LANGUAGE REFERENCE

CONTENTS

LANGUAGE REFERENCE	1
OVERVIEW	2
BOOLEAN-EQUATION ELEMENTS	4
STATE-MACHINE CONSTRUCTS	5
SPECIFYING OUTPUTS IN IF-THEN-ELSE AND CASE STATEMENTS	6
SYNTAX AND EXAMPLES	12
ASSIGNMENT OPERATOR.....	14
AUTHOR.....	18
BOOLEAN EQUATION	20
CASE	24
CHECK.....	30
CHECKQ.....	34
CHIP.....	38
.CLKF.....	40
CLKF.....	44
CLOCKF	46
.CMBF.....	48
COMBINATORIAL.....	50
COMMENT.....	52
COMPANY.....	54
CONDITIONS.....	56
DATE	60
DECLARATION SEGMENT	62
DEFAULT_BRANCH.....	66
DEFAULT_OUTPUT.....	70
EQUATIONS SEGMENT	72
EXPRESSION.....	74
FLOATING PINS AND NODES.....	78
FOR-TO-DO.....	82
FUNCTIONAL EQUATIONS	86
GND	90
GROUP.....	92

IF-THEN-ELSE, EQUATIONS	96
IF-THEN-ELSE, SIMULATION	100
.J EQUATION	102
.K EQUATION.....	104
LATCHED.....	106
LOCAL DEFAULT	108
MACH_SEG_A.....	110
MASTER_RESET.....	114
MEALY_MACHINE.....	116
MINIMIZE_OFF.....	118
MOORE_MACHINE.....	120
NODE.....	122
OPERATOR.....	126
.OUTF.....	128
OUTPUT_ENABLE.....	132
OUTPUT_HOLD	134
PAIR.....	136
PATTERN	140
PIN	142
PRELOAD.....	146
.PRLD.....	148
PRLDF	150
.R EQUATION.....	152
REGISTERED.....	154
REVISION.....	156
.RSTF	158
.S EQUATION.....	160
.SETF	162
SETF	164
SIGNATURE.....	166
SIMULATION.....	170
START_UP	172
STATE	176
STATE ASSIGNMENT EQUATION.....	180
STATE EQUATIONS.....	184
STATE OUTPUT EQUATION.....	188
STATE TRANSITION EQUATION.....	192
STRING.....	194
TEST	198

.T EQUATION.....	202
.T1 EQUATION.....	204
.T2 EQUATION.....	206
TITLE	208
TRACE_OFF.....	210
TRACE_ON	212
.TRST	214
VCC.....	218
VECTOR	220
WHILE-DO	222

This chapter describes language elements available through the PALASM 4 software for all PLD- and MACH-device designs.¹ Language elements are organized alphabetically by name and include the following information.

- **Name** identifies a specific element and explains its purpose.
- **Syntax** identifies the segment of the PDS file where the element is used and shows required and optional syntax and variables; an example of use in a PDS file is included.
- **Definitions** describe each parameter and identifies specific syntax requirements.
- **Use** provides additional details.

Important: In the syntax boxes of this chapter, required information appears in all capital letters; variables and optional information have initial caps.

¹ All information is also available at the workstation through Online Help.

OVERVIEW

The PALASM 4 software supports three kinds of design specifications.

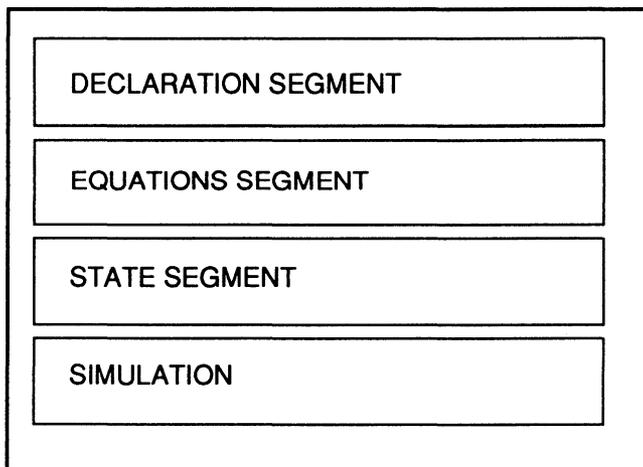
- Boolean equations
- State-machine descriptions
- Schematics

Only text-based designs use the language elements described in this chapter.

Important: A design entered as a schematic has its logic converted to Boolean equations during the compilation process. In addition, you can manually convert a schematic-based design to Boolean equations.

In any case, you can edit the resulting PALASM description specification (PDS) file to modify the design. At that point, all file and language elements relating to Boolean-equation specifications apply.

Text-based designs are described in PDS files, which are divided into several segments.



All PDS files include a declaration segment and usually a simulation segment.² These segments contain identical information regardless of the kind of description you produce. Depending upon the type of design description you choose, the PDS file will contain an equations segment and/or a state segment. Files that contain both an equations segment and a state segment are referred to as mixed-mode designs.³

The software is not case sensitive. You can enter any information using upper- or lowercase letters or a combination. For example, PALASM 4 treats the following as the same.

- ABC
- abc
- Abc

This chapter does not describe how to create a PDS file⁴ using the text editor, nor how to use the declaration-segment form,⁵ available when you create a new text-based design.

2 Refer to Section II, Chapter 6, for details about when to include the simulation commands in a separate file rather than in the PDS file.

3 Refer to Section II, Chapter 4, for details about producing mixed-mode designs.

4 Refer to Section I, Chapter 2, for a step-by-step guide to producing a PDS file for both Boolean and state-machine designs.

5 When you begin a new text-based design, a declaration-segment form appears, which you can complete to specify header information, PIN/NODE statements, chip name, and PLD or MACH-device type. Refer to Chapter 9, in this section, for details.

BOOLEAN-EQUATION ELEMENTS

The next figure identifies all elements available for use in each segment of a PDS file for a Boolean-equation design specification.

DECLARATION

AUTHOR	NODE
CHIP	PATTERN
COMBINATORIAL	PIN
COMMENT	REGISTERED
COMPANY	REVISION
DATE	SIGNATURE
DECLARATION	STRING
GROUP	TITLE
LATCHED	VECTOR

EQUATIONS

BOOLEAN EQUATION	OPERATOR
EXPRESSION	.PRLD
CASE	.R EQUATION
.CLKF	.RSTF
.CMBF	.S EQUATION
COMMENT	.SETF
EQUATIONS	.T EQUATION
FUNCTIONAL EQUATION	.T1 EQUATION
GND	.T2 EQUATION
IF-THEN-ELSE	.TRST
.J EQUATION	VCC
.K EQUATION	VECTOR

SIMULATION

EXPRESSION	PRLDF
CHECK	SETF
CLOCKF	SIMULATION
COMMENT	TRACE_OFF
FOR-TO-DO	TRACE_ON
IF-THEN-ELSE	VECTOR
PRELOAD	WHILE-DO

STATE-MACHINE CONSTRUCTS

The following figure identifies all elements available for each segment of a PDS file for a state-machine design specification.

DECLARATION	
AUTHOR	NODE
CHIP	PATTERN
COMBINATORIAL	PIN
COMMENT	REGISTERED
COMPANY	REVISION
DATE	SIGNATURE
DECLARATION	STRING
GROUP	TITLE
LATCHED	VECTOR
STATE	
EXPRESSION	.OUTF
CLKF	.OUTPUT_ENABLE
COMMENT	OUTPUT_HOLD
CONDITIONS	START_UP
DEFAULT_BRANCH	STATE
DEFAULT_OUTPUT	STATE ASSIGNMENT
LOCAL DEFAULT	STATE EQUATIONS
MASTER_RESET	STATE TRANSITION
MEALY_MACHINE	VECTOR
MOORE_MACHINE	
SIMULATION	
EXPRESSION	PRLDF
CHECK	SETF
CLOCKF	SIMULATION
COMMENT	TRACE_OFF
FOR-TO-DO	TRACE_ON
IF-THEN-ELSE	VECTOR
PRELOAD	WHILE-DO

SPECIFYING OUTPUTS IN IF- THEN-ELSE AND CASE STATEMENTS

The following rules apply to each pin or node for outputs defined in the IF-THEN-ELSE or CASE statement.

Specify the desired output for each test condition of the IF-THEN-ELSE or CASE statement you want generated.

Note: If you do not request a specific output, the software assumes either you do not care about the output for this condition, or the output for this condition is actually defined through some other equations.

If you define the behavior of an output in an IF-THEN-ELSE or CASE statement, and then define it further using a Boolean equation or a separate IF-THEN-ELSE or CASE statement, you must obey certain rules. Specifically, you must avoid situations when both of the following occur.

- More than one of the resulting equations can be simultaneously evaluated as true.
- The output behavior defined in one equation contradicts behavior defined in the other.

If you violate these rules, the software reports the following error message: Overlapping on/off covers (check equations for completeness).⁶

Four Ways of Specifying Outputs

You can specify the behavior of each pin or node as follows.

1. Define the output as a function of the same inputs for each possible test condition.
2. Define the output as a function of different inputs for different test conditions.

⁶ Techniques for avoiding this error are discussed under the heading Avoiding Overlap Errors.

3. Define the output for some of the possible test conditions and not for others.
4. Define the output two different ways for the same test condition.

The following examples are given for the IF-THEN-ELSE statement, but they apply to the CASE statement also. The IF-THEN-ELSE statement is a special instance of the CASE statement in which IF and ELSE portions replace the individual test conditions of the CASE statement.⁷

Example 1

Defining the output as a function of the same inputs for each possible test condition.

```
IF X = 0 THEN
  BEGIN
    Q = A * B
  END
ELSE
  BEGIN
    Q = 0 ;equivalent to Q = GND or /Q = 1 or /Q = VCC
  END
```

The software interprets this statement as follows.

- When X = 0, output Q is high when the expression A * B evaluates as true, and low when the expression A * B evaluates as false.
- When X = 1, output Q is always low.

The statement is reduced to the following Boolean form.

$$Q = A * B * \bar{X}$$

⁷ Refer to TUTOR6.PDS and TUTOR7.PDS in the PALASMEXAMPLES directory for additional examples of IF-THEN-ELSE and CASE statements.

This statement is then combined with any other equations for Q by the minimizer.

Example 2

Defining the output as a function of different inputs for different test conditions.

```
IF X = 0 THEN
  BEGIN
    Q = A * B
  END
ELSE
  BEGIN
    Q = C * D
  END
END
```

The software interprets this statement as follows.

- When X = 0, output Q is high when the expression A * B evaluates as true, and low when the expression A * B evaluates as false.
- When X = 1, output Q is high when the expression C * D evaluates as true, and low when the expression C * D evaluates as false.

The statement is reduced to the following Boolean form.

$$Q = C * D * X + A * B * /X$$

This statement is then combined with any other equations for Q by the minimizer.

Example 3

Defining the output for some of the possible test conditions and not for others.

```
IF X = 0 THEN
  BEGIN
    Q = A * B
  END
; No ELSE condition or no output
; equation
; in some of the CASE conditions
```

The software interprets this statement as follows.

- When $X = 0$, output Q is high when the expression $A * B$ evaluates as true, and low when the expression $A * B$ evaluates as false.
- When $X = 1$, output Q is undefined. In the Karnaugh map, don't-care values are used for all map locations where $X = 1$, unless a location is defined by other equations for Q .

This statement is reduced to the following Boolean form.

$$Q = A * B * /X + \text{"don't care"} * X$$

This is then combined with any other equations for Q by the minimizer. In the event that there is no other equation for Q , this equation reduces to $Q = A * B$, in which case output Q is no longer a function of input X . This occurs because of the don't-care values introduced by omitting the definition of output Q when $X = 1$. If you want output Q to remain a function of X , rewrite the IF-THEN-ELSE statement, adding the ELSE clause shown below.

```
IF X = 0 THEN
  BEGIN
    Q = A * B
  END
ELSE
  BEGIN
    Q = 0
  END
```

Example 4

Defining the output two different ways for the same test condition.

```
IF X = 0 THEN
  BEGIN
    Q = A * B
    /Q = /A
  END
ELSE
  BEGIN
    Q = 0 ;equivalent to Q = GND or /Q = 1 or /Q = VCC
  END
END
```

Here, the designer has defined explicitly when the output should be high or low, when $X = 0$. The software interprets this statement as follows.

- When $X = 0$, output Q is high when the expression $A * B$ evaluates as true, and low when the expression input A is low.
- When $X = 1$, output Q is always low.

This statement is reduced to the following Boolean form.

$$Q = A * /X$$

This statement is then combined with any other equations for Q by the minimizer. Note that output Q is no longer a function of input B . This occurs because the software assumes don't-care values for all conditions that are not covered by the equations

$$Q = A * B \text{ and } /Q = /A, \text{ where } X = 0.$$

Avoiding Overlap Errors

The following example shows how two separate IF-THEN-ELSE statements can interact to produce an Overlapping on/off covers (check equations for completeness) error. This error is typically caused by unrealizability, as illustrated in the following example.

```

IF A = 1 THEN
  BEGIN
    Q = C * D
  END
IF B THEN
  BEGIN
    Q = 1
  END

```

A problem occurs when both A and B are high and C or D or both C and D are low. In this situation, the first IF statement tells output Q to go low, while the second IF statement tells the same output to go high. Since Q cannot be high and low simultaneously, the equation set is unrealizable.

There are several ways to avoid problems of this nature, as described next.

Method 1

Make it impossible to have more than one IF-THEN-ELSE or CASE statement evaluate as true simultaneously.

```

IF A = 1 THEN
  BEGIN
    Q = C * D
  END
IF /A * B THEN           ;Condition /A prevents both IF
  BEGIN                 ;statements from evaluating as
    Q = 1               ;true simultaneously.
  END

```

Method 2

Make it impossible for the output specifications to conflict by always using the same output equation.

```
IF A = 1 THEN
  BEGIN
    Q = 1                ;This output equation...
  END

IF B = 1 THEN
  BEGIN
    Q = 1                ;...is identical to this.
  END
```

SYNTAX AND EXAMPLES

In this chapter, all language elements are presented in two forms, as shown below.

Syntax

Output_pn	Assignment Operator	Expression
<i>Example</i>		
EQUATIONS	=	I1 * I2 * I3 + I4 * I5
/02		
...		

- The syntax area shows the syntactical name of each element needed to complete the statement, and shows the proper order.
- The example provides a sample as it is used in a PDS file.

Important: Spacing is exaggerated in the samples shown in this chapter for readability. For example, tabs separate each element in the example to align it with the corresponding identifier above. In most cases, however, blanks can be used as separators in the PDS file.

You may not use curly braces, { }, inside IF-THEN-ELSE or CASE statements. Instead of the curly-braced shorthand, the right side of the referred-to logic equation must be copied in full as the right side of the equivalent logic equation (as in output pairing). Also, even though it is legal to do so, it is not recommended that you use curly braces outside IF-THEN-ELSE and CASE constructs to refer to an equation that is not defined inside of such constructs. The result may not what you expect or desire.

ASSIGNMENT OPERATOR

The assignment operator is a symbol that defines a specific operation interpreted by the software when processing design files. There are several assignment operators, which perform different functions depending on the software operation and where the operator appears.

Devices Supported: All PLD devices.

SYNTAX

You can use the assignment operator in equations, as shown below, and in state and conditions segments of a PDS file for either Boolean or state-machine descriptions.

Syntax

	Pn	Assignment Operator	Expression
<i>Example</i>	Q0	=	EXT1
	Q1	:=	EXT2
	Q2	*=	EXT3

Definitions

The various assignment operators are defined below.

Assignment Operator

The := and *= operators are provided to support older PLD designs that include a pin list not containing a storage type.

ASSIGNMENT OPERATOR

=	<p>This universal assignment operator agrees with any data-storage type: Combinatorial, registered, or latched.</p> <p>You can use this assignment operator in expressions whether or not the PIN or NODE statements contain a storage type. This operator is best suited to new designs.</p>
:=	<p>This assignment operator defines a registered output. The signals in the expression must be defined in either the PIN or NODE statement as registered. Otherwise, an error occurs and processing stops.</p>
* =	<p>This assignment operator defines a latched output. The signals in the expression must be defined in either the pin or node statement as latched or an error is reported during processing.</p>

USE

In the equations segment, the assignment operator performs two functions.

- It defines the output on the left side of the equation as a function of the various inputs and feedback signals listed on the right side.
- It defines the output storage type.

You can assign the storage type in a PIN or NODE statement. In this case, you can use the universal assignment operator, =, in equations or expressions. If you use the := or * = operators, they must agree with the storage type defined in the PIN or NODE statement or an error occurs.

ASSIGNMENT OPERATOR

You can use assignment operators in the STATE and CONDITIONS statements in the state-machine designs, as shown below.

```
...  
STATE  
MOORE_MACHINE  
ZERO.OUTF = /CNT2 * /CNT1 * /CNT0  
CONDITIONS  
CNTUP = ENABLE * CNT_UP  
...
```


AUTHOR

This keyword defines the name of the design's author. This statement is useful for documenting the design.

Devices Supported: All PLD devices.
--

SYNTAX

You can use the AUTHOR statement only in the declaration segment, as shown below.⁸

Syntax

AUTHOR	Designer
--------	----------

Example

TITLE	
PATTERN	
REVISION	
AUTHOR	Karen Olsen
COMPANY	
DATE	

Definitions

Only the descriptor following the keyword, Author, is discussed.

Designer

Designer defines the name of the individual who created the design. Observe the guidelines below.

- Include the AUTHOR statement and the designer name in either Boolean or state-machine files or in the schematic-control file form.
- Place the statement and name in the order specified in the example above, following REVISION and preceding COMPANY.
- Use any combination of up to 59 alphanumeric characters and the period character.

⁸ Refer to Chapter 9, in this section, for details about the declaration-segment form for new text-based designs and the schematic-control form for new schematic-based designs.

AUTHOR

Reserved words are allowed within this statement; however, **do not** use any punctuation or symbols other than the period.

USE

The following error conditions pertain to the AUTHOR statement.

- Without the AUTHOR statement, the software issues a warning and continues processing the file.
- With multiple AUTHOR statements, the software issues an error and stops processing the file.

BOOLEAN EQUATION

These equations are used to define the desired logic functions produced at the outputs. Boolean equations form the backbone of any PDS file containing a Boolean description.⁹

Devices Supported: All PLD devices.
--

SYNTAX

You use Boolean equations in the equations segment of a PDS file, following the keyword EQUATIONS.

Syntax

Output_pin	Assignment Operator	Expression
<i>Example</i>		
/02	=	I1 * I2 * I3 + I4 * I5

Definitions

All parameters are described below. Additional details are provided under Use.

Output_pin

Output_pin is the name of an output pin or node defined in the declaration segment of the PDS file. Once defined, you can include the name in a Boolean equation to define how this output is to be used.¹⁰

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.¹¹

⁹ Refer to BOOLEAN EQUATION and EXPRESSION, in this chapter, for additional details.

¹⁰ Refer to NODE and PIN, in this chapter, for details about including a forward slash, /, to define polarity.

¹¹ Refer to ASSIGNMENT OPERATOR, in this chapter, for more information.

BOOLEAN EQUATION

Expression

Expression is a group of signals or product terms, on the right side of an equation, separated by logical operators.¹² The product operator, *, denotes a logical AND function and the sum operator, +, denotes a logical OR function. You can use strings and vectors in an expression.

All values in an expression for a PLD design are signal names that must be defined before their use. The name must be defined in PIN and NODE statements in the declaration segment of the PDS file.

USE

The following conventions should be observed.

- Place a Boolean equation in any order, unless it's a substitution, in which case you must define the substitute expression before using it.
- Follow the structure shown in the syntax example; however, you can include strings and vector notation.
 - a. Include either blanks or a tab character before and after the assignment operator.
 - b. Use an equal sign as the assignment operator.

The software determines storage type, either registered, combinatorial, or latched, from the PIN and NODE statements in the declaration segment.

¹² Refer to EXPRESSION, in this chapter, for additional details.

BOOLEAN EQUATION

- Include up to 80 characters per line in each equation and include as many lines as necessary.

You can place the expression on a separate line.

You can use the following high-level language statements and constructs to specify logic behavior. These are converted to Boolean equations during software processing.

- CASE
- FOR-TO-DO
- IF-THEN-ELSE
- WHILE-DO

State equations are another high-level representation of logic behavior for state-machine designs. When you compile a state-machine design, state equations are converted to Boolean equations.

Functional equations are a type of Boolean equation that defines an input. Functional equations have a similar format but a different use.¹³

¹³ Refer to the following topics, in this chapter, for additional details: COMBINATORIAL, EXPRESSION, FUNCTIONAL EQUATIONS, LATCHED, REGISTERED, and STATE EQUATIONS.

CASE

This keyword provides a condition-testing structure for Boolean equations. The CASE statement more efficiently supports multiple values than the nested IF-THEN-ELSE statement.

Devices Supported: All PLD devices.
--

SYNTAX

You can only use the CASE statement in the equations segment of Boolean designs.

Syntax

```
CASE (Condition Signals)
  BEGIN
    Value:          BEGIN
                    Action statement
                    END

    Value:          BEGIN
                    Action statement
                    END

    Keyword:        BEGIN
                    Action statement
                    END

  END
```

Example

```
CASE (A,D)
  BEGIN
    0:              BEGIN
                    C = A * B
                    END

    3:              BEGIN
                    C = A + B
                    END

    2:              BEGIN
                    C = 1
                    END

    OTHERWISE:      BEGIN
                    C = 0
                    END

  END
```

Definitions

You can use groups, vector notation, and strings in a CASE statement. Parameters following the keyword, CASE, are defined below; additional details appear under Use.

Condition Signals

Condition signals define a set of signals, which are tested against a list of values, to determine subsequent actions in following statements.

The set is concatenated into a single binary value using signal values as individual bits. Each **signal** value in the set is represented by a name and has a binary value of either 0 or 1. Each **signal name** can be either an input or an output. The following rules must be observed.

- Follow the keyword in a CASE statement with the condition signals.
- Enclose signal names or vector notation in parentheses separated by commas: for example, (A,B,C[0..4])

Important: Signal names or vector notation must be separated by commas.
--

- Define each signal name or vector notation and its value in a PIN or NODE statement in the declaration segment.

Value

Value defines the constant in the subsequent action statement. This value is tested against the present value of the condition signals to determine whether that action is taken or skipped. The rules below apply.

- The value can be any constant; however, do not duplicate values.

CASE

The default constant is decimal. Non-decimal constants are expressed with the following prefixes.

#b Binary
#h Hexadecimal
#o Octal

All values are converted to binary during compilation.

- The value must end with a colon and precede an action statement.

Action Statement

The action statement defines the logic for a particular condition-signal value. This logic is implemented when the value preceding this statement matches the current value of the condition signals. The following rules apply.

- Precede each action statement with a value.
- Enclose each statement with BEGIN and END.

Action statements inside the BEGIN and END block can be any expression that is valid within the equations section of the design.

An action is a series of any legal PALASM statements within the equations section.

OTHERWISE

Use this keyword to identify the beginning of an optional statement indicating the action for default values of the CASE statement. When you define every possible value, you do not need this statement. However, any unspecified conditions are assumed to be don't care; they are not assumed to be false. Signals for which default conditions are not specified are eliminated from the design during the logic-reduction process.

CASE

The OTHERWISE statement generates the condition as identified and shown below.

- OR all conditions for values defined in action statements before this statement.
- Complement the entire OR term, then AND it with the right side of the OTHERWISE statement.

```
CASE (A,D)
BEGIN
  0:          BEGIN C = A * B END
  3:          BEGIN C = A + B END
  2:          BEGIN C = 1          END
  OTHERWISE: BEGIN C = 0          END
END
```

At each clock cycle, or anytime the signals change, the condition signals (A,B) are evaluated. When the value of the signals matches the value in one of the following action statements, the logic defined in that statement is implemented; otherwise, C = 0.

The previous example is expanded during compilation into the following Boolean equation.

```
C = (VCC * /A * /B)
    + /(VCC * A * B)
    + (D * /( /A * /B + A * B))
```

USE

You can specify how the software treats default values for CASE statements by selecting one of two options on the File/Set up/Logic synthesis options form, as follows.

OPTIONS	DEFINITIONS
Don't care	Unspecified default conditions are assumed to be don't care.
Off	Unspecified default conditions are assumed to be false.

CASE

The don't-care option requires you specify both the on and off sets. The off option requires you to specify only the on sets; the software assumes all other conditions to be off.

You may lose signals from the design if you select the don't-care option and do not specify all the default conditions. If the software treats these signals as don't care, they will be eliminated from the design during logic reduction.

Important: When translating designs created with PLPL, you must select the off option because PLPL treats unspecified default conditions as false.

You can nest CASE statements within IF-THEN-ELSE statements and other CASE statements.

```
CASE (A,D)
BEGIN
  0:    BEGIN C=A*B END
  1:    CASE (C,D)
        BEGIN
          0:    BEGIN E=C*D END
          1:    BEGIN E=C+D END
          OTHERWISE:  E=1  END
        END
  OTHERWISE:  C=0 END
END
```

There is no limit to the number of nested statements you can include in a design; however, too many levels of nesting may cause you to run out of memory.¹⁴

If any equations are incompletely specified or ambiguous, the unspecified signals are assumed to be don't care. This increases the amount of logic reduction possible during the minimization process. However, if you have not fully evaluated the consequences of these don't-care signals, the equations resulting from compilation may be different than you intended.¹⁵

¹⁴ Refer to the following topics, in this chapter, for additional details: GROUP, IF-THEN-ELSE, and VECTOR.

¹⁵ Refer to Chapter 9, in this section, for details about default options for CASE and IF-THEN-ELSE statements on the Logic Synthesis Options form.

CHECK

This keyword in a simulation command verifies signal values at the **pin** are equal to expected values.

Devices Supported: All PLD devices.

SYNTAX

You use the CHECK command in either the simulation segment of a PDS file or in an auxiliary simulation file for Boolean, state-machine, or schematic-based designs.

Syntax

CHECK Prefix_pns

Example

```
SIMULATION
CHECK      01 /02 ^03 %04 PLAYING
```

DEFINITIONS

If the signal being tested is defined with the same polarity as in the Pin/Node declaration segment, the signal is checked to verify it is a logical 1. If the polarity is reversed, the signal is checked to verify it is a logical 0.

Note: The syntax examples are valid only if the signals are defined as active-high in the Pin/Node declaration segment.

Parameters following the keyword are defined below. Additional details are provided under Use.

Prefix

Prefix indicates the logic state of the corresponding pin, node, or state. Do not leave a blank between Prefix and pns. There are four prefixes: null, forward slash, /, caret sign, ^, and percent sign, %.

CHECK

- The null prefix indicates an **active-high** signal is checked to verify it is a logical 1. In the syntax example, O1 has a null prefix.

When used in conjunction with a state name, a null prefix indicates the specified state should be checked. In the syntax example, PLAYING has a null prefix.

- The forward slash, /, indicates an **active-high** signal is checked to verify it is a logical 0. In the syntax example, O2 has a forward-slash prefix.
- The caret sign checks the corresponding signal for a high-impedance state. High impedance occurs when a three-state buffer on an I/O pin is disabled. In this case, the letter Z appears in the simulation files to indicate the high-impedance state. In the syntax example, O3 has a caret prefix.
- The percent sign checks the corresponding signal for a don't-care state. A don't-care condition occurs when combinatorial logic is not initialized. In this case, the letter X appears in the simulation files to indicate the don't-care state. In the syntax example, O4 has a percent prefix.

Pns

Pns defines the names of the pins, nodes, or states to be verified.

- Each signal name can be up to 14 characters in length.
- Include up to 76 characters per line and use as many lines as you need.

The screen displays up to 76 characters per line; however, all information is processed properly even if it extends beyond the 76th character.

CHECK

- Include a blank between the keyword and the first pin, node, or state in the list.

You can include **multiple** pin and node names. You can use strings or vector notation to define the signal list.

- Separate multiple prefixed pin and node names with a blank.

USE

The CHECK command verifies values of the defined signal at the pin. In contrast, the CHECKQ command verifies values at the Q output of a register.

If the signal being tested is defined with the same polarity as in the Pin/Node Declaration segment, the signal is checked to verify it is a logical 1. If the polarity is reversed, the signal is checked to verify it is a logical 0.

A conflict occurs when the value at the pin does not match the expected value. Each conflict is identified with a question mark, ?, in the simulation output files; a warning is issued and the expected value is reported in the execution-log file.¹⁶

The CHECK command verifies logical operations only and does not add test vectors in the JEDEC file.¹⁷

¹⁶ Refer to Section II, Chapter 4, for additional details.

¹⁷ Refer to TEST, in this chapter, for details about creating test vectors in JEDEC files.

CHECKQ

This keyword in a simulation command verifies values at the Q outputs of **registers** are equal to expected values.

Devices Supported: All PLD devices.
--

SYNTAX

You use the CHECKQ command in either the simulation segment of a PDS file or in an auxiliary simulation file for Boolean, state-machine, or schematic-based designs.

Syntax

CHECKQ Prefix_rns

Example

```
SIMULATION
CHECKQ Q0 /Q1 PLAYING
```

DEFINITIONS

Parameters following the keyword are defined below. Additional details are provided under Use.

Because CHECKQ verifies signal values at the Q output of registers, you do not need to account for active-low pin declarations. This makes CHECKQ especially useful for verifying states.

Prefix

The prefix indicates the logic state of the corresponding register, node, or state. Do not leave a blank between Prefix and rns. There are two prefixes: null and forward slash.

- The null prefix indicates the register or node should be a logical 1. In the syntax example, Q0 has a null prefix.

When used in conjunction with a state name, a null prefix indicates the specified state should be checked. In the syntax example, PLAYING has a null prefix.

- The forward slash indicates the signal should be a logical 0. In the syntax example, Q1 has a forward-slash prefix.

Rns

Rns defines the names of the output registers, nodes, or states to be verified. Each value represents both the signal name or state and the expected output value.

- Each signal name can be up to 14 characters in length.
- Include up to 76 characters per line and use as many lines as you need.

The screen displays up to 76 characters per line; however, all information is processed properly even if it extends beyond the 76th character.

- Include a blank between the keyword and the first register, node, or state in the list.

You can include **multiple** register and node names. You can use strings or vector notation to define the signal list.

- Separate multiple prefixed register and node names with a blank.

CHECKQ

USE

The CHECKQ command verifies that signal values at the register outputs are equal to the expected values. In contrast, the CHECK command verifies pin and node values at the output pin.

Because CHECKQ verifies signal values at the Q output of registers, you do not need to account for active-low pin declarations. This makes CHECKQ especially useful for verifying states.

A conflict occurs when the value of the output register does not match the value defined in the CHECKQ command. Each conflict is identified with a question mark in the simulation output files; a warning is issued and the expected value is reported in the execution log file.

The CHECKQ command verifies logical operations only and does not create test vectors in the JEDEC file.

CHIP

This keyword introduces the statement that defines the chip name and the PLD or MACH device for the design being created.

Devices Supported: All PLD devices.

SYNTAX

You use the CHIP statement in the declaration segment of either Boolean or state-machine designs¹⁸

Syntax

CHIP	Name	Device
<i>Example</i>		
TITLE		
PATTERN		
REVISION		
AUTHOR		
COMPANY		
DATE		
CHIP	Counter	PALCE22V10H-25

Definitions

The CHIP statement must follow the DATE statement and precede PIN and NODE statements. Parameters that follow the keyword are defined below.

Name

Name assigns a chip name that conforms to the rules below.

- Use up to 14 alphanumeric characters in the PDS file. The first character cannot be numeric.

Do not use operators, reserved words, carriage returns, tabs, or blanks.

¹⁸ Refer to Chapter 9, in this section, for details about the declaration-segment form for new text-based or control-file form for schematic-based designs.

Device

Device assigns a valid AMD device name, such as PAL16R8.

- The name must conform to the basic device number. Power, speed, package, and operating range designators are optional.¹⁹

USE

The software cannot process a design file if it does not contain a CHIP statement. Errors can cause the software to misinterpret the CHIP statement and, often, the PIN and NODE statements.

¹⁹ Refer to the specific device datasheet for the correct device name.

.CLKF

This reserved word defines a clock signal for a pin or node. The operation differs depending upon the device you've chosen.²⁰ For example, for devices with programmable clocks, you can use .CLKF to assert the clock on a flip-flop.

Devices Supported

PAL16RA8	PAL20RA10	PAL26V12	PALCE29M16	PALCE29MA16	PAL32VX10
PALCE610	PLS30S16	MACH 1	MACH 2		

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state designs or in any registered-latched design.

Syntax

Pn.CLKF	Assignment Operator	Expression
---------	---------------------	------------

Example

Q0.CLKF	=	EXT
---------	---	-----

Definitions

All parameters are defined below.

Pn.CLKF

Pn.CLKF assigns a pin or node name followed by the reserved word .CLKF. The name must be defined in an earlier PIN or NODE statement in the declaration segment. A forward slash before the pin or node name defines a falling edge clock. The absence of a forward slash defines a rising edge clock.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.²¹

²⁰ Refer to Chapter 11, in this section, for more information regarding .CLKF statements.

²¹ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

Expression

Expression includes the signal name or expression that defines the clock source.

Important: In PLDs with multiple clocks, you must specify the clock pin; don't use an expression.

All values in an expression for a PLD design are signal names that must be defined in earlier PIN and NODE statements.²² In the syntax example, when EXT is true, the clock associated with Q0 is asserted.

USE

Multiple .CLKF statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting.

If no clock pin is defined using the .CLKF instruction, registered outputs default to the nominated default pin for the specified device.²³

Note: For devices without a default clock pin, such as the PAL20RA10, you must define a clock pin.

You can use a group, string, or vector notation to define signals, which is an excellent way to assign a functional equation to several pins.²⁴

The next example shows how to use vector notation.

²² Refer to NODE and PIN, in this chapter, for details about including a forward slash to define polarity.

²³ Refer to Chapter 11, in this section, for details regarding default clock pins.

²⁴ Refer to FUNCTIONAL EQUATIONS and GROUP, in this chapter, for additional details.

.CLKF

```
;DECLARATION SEGMENT
  PIN          14..16      Q[0..2]
  PIN          8..10       CLK[0..2]
;EQUATIONS
  Q[0..2].CLKF = CLK[0..2]
```

This generates the following equations.

```
Q[0].CLKF = CLK[0]
Q[1].CLKF = CLK[1]
Q[2].CLKF = CLK[2]
```

The following example shows how to use a STRING statement.

```
;DECLARATION SEGMENT
  STRING  IN1  'A1 * A2'
;EQUATIONS
  Q0.CLKF = IN1
```

This generates the following equation.

```
Q0.CLKF = A1 * A2
```

The following example shows how to use a GROUP statement.

```
;DECLARATION SEGMENT
  GROUP  CLOCKS  A  B  C
;EQUATIONS
  CLOCKS.CLKF = D * E
```

This generates the following equations.

```
A.CLKF = D * E
B.CLKF = D * E
C.C.KF = D * E
```


CLKF

This keyword identifies which clock is used for the synchronized operation of the state machine.

Devices Supported					
PAL26V12	PALCE29M16	PALCE29MA16	PALCE610	PLS30S16	MACH 1
MACH 2					

SYNTAX

You can use this statement only in the state segment of state-machine designs.

Syntax

CLKF	Assignment Operator	Clock_pin
------	---------------------	-----------

Example

CLKF	=	CLK1
------	---	------

Definitions

Parameters following the keyword are defined below.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.²⁵

Clock_pin

Clock_pin is the pin you define as the clock source for the state machine.²⁶

- Specify the name exactly as it's defined in the PIN statement in the declaration segment.
- Place a forward slash before the equation to select a falling edge clock. For example,

`/CLKF = CLK1`

²⁵ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

²⁶ Refer to the specific device datasheet to determine which pin to use.

Note: To define a falling edge clock for PAL29M16, you must use NCLKF instead of the forward slash.

USE

Without a CLKF equation, you cannot specify a non-default clock pin if you use automatic state-bit assignment.

This equation creates clock assignments for both state bits and output registers. ²⁷

²⁷ Refer to BOOLEAN EQUATION and .CLKF, in this chapter, for additional details.

USE

The CLOCKF command is similar to the SETF command,²⁸ except that CLOCKF generates a low-to-high-to-low pulse.

Important: You cannot use a CLOCKF command for registered devices without clock pins. These devices require two SETF lines to generate the CLOCKF pulse.

- If you do not use a SETF command to set the clock signal low before clocking the register, the first clock pulse has no effect.

Note: At the start of simulation, clock signals must be initialized low.

- If you design a system using multiple banks of independent clock pins and you connect these pins externally for synchronous clock cycles, you must list all clock pins in the CLOCKF command to synchronize these banks during simulation.
- If you do not specify a clock pin in the CLOCKF command, the nominated default clock pin for that device is used.

Some devices, such as the PAL30S16, include a pin you can configure as either an input or a clock.

- If you define the pin as a clock pin using a corresponding .CLKF command, the CLOCKF command generates a clock pulse for that pin.
- If you define the pin as a clock input on the PLS30S16, you can still use the SETF command to create a test vector for that pin.

²⁸ Refer to SETF, in this chapter, for additional details.

.CMBF

This reserved word allows you to customize a flip-flop for dynamic register bypass.

Devices Supported: PAL32VX10.

SYNTAX

You include this functional equation only in the equations segment of either Boolean or state designs.

Syntax

	Pn.CMBF	Assignment Operator	Expression
<i>Example</i>	OUT1.CMBF	=	IN4 * /IN3

Definitions

All parameters are defined below.

Pn.CMBF

Pn.CMBF is a pin or node name followed by the reserved word .CMBF. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

- You cannot use negative polarity on the left side of the equation.
- You can use the group, string, and vector notation features to define signals, which is an excellent way to assign this statement to several pins and nodes.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.²⁹

Expression

Expression is the logic you define to be bypassed. In the syntax example, when IN4 and /IN3 are true, OUT1 is true and the flip-flop associated with OUT1 is bypassed.

²⁹ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

The `.CMBF` statement overrides the registered pin or node type you defined in the `PIN` or `NODE` statement in the declaration segment.³⁰ The default case is set to combinatorial.

If you have multiple functional equations for the same pin or node, an error occurs during assembly.

³⁰ Refer to the following topics, in this chapter, for additional information: `BOOLEAN EQUATION`, `COMBINATORIAL`, `EXPRESSION`, `FUNCTIONAL EQUATIONS`, and `REGISTERED`.

COMBINATORIAL

This optional reserved word defines the output type for devices with programmable outputs. When a pin or node is defined as combinatorial, the logic output is immediate; the output value is not stored.

Devices Supported: All devices **except** the PAL16R8, PAL20R8, PAL20X10.

SYNTAX

You include the optional reserved word in the PIN or NODE statement of Boolean and state-machine designs.

Syntax

Pn	Number	Location_name	Storage
<i>Example</i>			
PIN	14	OUT1	COMBINATORIAL
NODE	15	OUT2	COMB

Definitions

Only the reserved word COMBINATORIAL is discussed below.³¹

Storage

Assign COMBINATORIAL as a value to define a specific pin or node storage type. Combinatorial is also the default when you do not specify a storage type.

- Place the reserved word COMBINATORIAL after the pin or node name in the corresponding statement.
- Use either the complete word COMBINATORIAL or the four-letter abbreviation, COMB.

³¹ Refer to the following topics, in this chapter, for additional details: DECLARATION SEGMENT, NODE, PIN, and STORAGE.

USE

There are two ways to enter the storage type.

- Use the declaration segment form, select the Storage field, display the option list, and select an option.
- Type the storage value in the appropriate PIN or NODE statement in the PDS file using a text editor.

COMMENT

A comment can explain a statement in the PDS file to assist you or another designer when editing or debugging the design.

Devices Supported: All PLD devices.
--

SYNTAX

You can place comments anywhere in the design file.

Syntax

`;Comment`

Example

```
;State setup and defaults
MOORE MACHINE
DEFAULT_BRANCH HOLD_STATE ;defines default state as holding
;State Assignments
...
```

Definitions

Details about comments follow.

Comment

A comment is any combination of alphanumeric characters, symbols, or punctuation preceded by a semicolon. The software ignores everything in the line following the semicolon. A comment can be several lines in length; however, each new line must begin with a semicolon.

USE

You can use comments to separate the various segments in either a Boolean or state-machine design.

COMPANY

This keyword begins the statement to define the name of the author's company.

Devices Supported: All PLD devices.

SYNTAX

You use this keyword in the declaration segment of either a Boolean or state-machine design.

Syntax

COMPANY

Company Name

Example

TITLE
PATTERN
REVISION
AUTHOR
COMPANY
DATE
CHIP

Advanced Micro Devices, Inc.

Definitions

Only the descriptor following the keyword, COMPANY, is discussed.

Company Name

The company name is optional and includes any combination of up to 59 alphanumeric characters.

- You can use other symbols or punctuation; however you cannot use the dollar sign, \$.
- You can use reserved words in this statement.

USE

There are two ways to enter the company name.

- Use the declaration segment form, select the company field, and type the name.
- Type the company name after the AUTHOR statement and before the date in the PDS file using a text editor.

COMPANY

The following error conditions pertain to the COMPANY statement.

- Without a COMPANY statement, a warning is issued and processing continues.
- With multiple COMPANY statements, an error is reported and processing stops.³²

³² Refer to the following topics, in this chapter, for additional details: AUTHOR, DATE, DECLARATION SEGMENT, PATTERN, REVISION, and TITLE.

CONDITIONS

This keyword identifies the beginning of the state transition equations segment.

Devices Supported: All synchronous registered PAL devices.

SYNTAX

This keyword must begin the condition-equations segment, and it must be the last part of the state segment in a PDS file.

Syntax

CONDITIONS			
Name	Assignment Operator	Expression	
<i>Example</i>			
STATE			
...			
CONDITIONS			
QRUN	=	Q2 * /Q1 * /Q0	
		+ Q3 * Q2	
		+ Q3 * Q1	
		+ Q3 * Q0	

Definitions

All parameters following the keyword CONDITIONS are identified below.³³

Name

Condition equations define the branching conditions that determine transitions and outputs.

- Ensure each name is unique.
- Include up to 14 alphanumeric characters.
- Do not include operators or reserved words.

³³ Refer to the following topics, in this chapter, for additional details: DEFAULT_BRANCH, DEFAULT_OUTPUT, LOCAL_DEFAULT, OPERATOR, STATE, STATE EQUATIONS, and STATE TRANSITION EQUATION.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.³⁴

Expression

Assign a Boolean expression to define the condition.³⁵

- All signal names in the expression must be defined in PIN and NODE statements in the declaration segment of the PDS file.
- Parentheses are allowed in the expression.
- VCC can be specified for unconditional-high signals.
- Multiple product terms are allowed in the expression.

USE

Condition equations define condition names used in state transition equations, as shown in the next example.

```
S1 := C1 -> S2
      C2 -> S3

CONDITIONS           ;begin conditions segment
C1 = I1
C2 = /I1
```

If the condition consists of a single input, the input name can be used in place of a condition name. As shown in the next example.

```
S4 := /I3 -> S5
```

34 Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

35 Refer to EXPRESSION, in this chapter, for more information.

CONDITIONS

If the condition consists of more than a single input, you must write a condition equation and use the condition name in the state transition equation.

You must avoid conflicting conditions when branching, as shown below; the design must be changed to prevent overlapping conditions. In this example, it is impossible to determine whether S1 goes to S2, or S1 goes to S3, when A, B, and C all equal 1 or 0.

```
;conflicting conditions example
S1 := C1 -> S2
      + C2 -> S3
CONDITIONS          ;begin conditions segment
C1 = A * B
C2 = A * C
```


DATE

This keyword begins the statement that identifies the creation date of the design.

Devices Supported: All PLD devices.
--

SYNTAX

You use this keyword in the declaration segment of either a Boolean or state-machine design.

Syntax

DATE	Creation Date
------	---------------

Example

TITLE	
PATTERN	
REVISION	
AUTHOR	
COMPANY	
DATE	01/09/91
CHIP	

Definitions

Only the descriptor following the keyword DATE is discussed.

Creation Date

The creation date is a set of numbers that represent the month, day, and year the design was created or edited. The date may be entered as any 60-character string. Place the DATE statement after the COMPANY statement and before the CHIP statement.

USE

The following error conditions pertain to the DATE statement.

- Without a DATE statement, a warning is issued and processing continues.
- With multiple DATE statements, an error is reported and processing stops.³⁶

³⁶ Refer to the following topics, in this chapter, for additional details: AUTHOR, DATE, DECLARATION SEGMENT, PATTERN, REVISION, and TITLE.

DECLARATION SEGMENT

The declaration segment is the first segment of any PDS file. It identifies basic design information, the chip name and device type and PIN and NODE statements required to process the design, and special definitions. A form automates entry of this information when you begin a design.

Devices Supported: All PLD devices.

SYNTAX

This segment is required for all designs and must be the first segment in the PDS file.

Syntax

```
-----Declaration Segment-----  
;-----  
design information  
;-----Pin Declarations -----  
Pin / Node statements(s)  
;Special definitions
```

Example

```
-----Declaration Segment-----  
TITLE      16 Bit Counter (up/down); replaced w/standard settings  
PATTERN    XS.PDS  
REVISION   2  
AUTHOR     Gail Tiberi  
COMPANY    Mystique, Inc.  
DATE       08/09/90  
CHIP       Counter                PAL16R6  
;-----Pin Declarations -----  
PIN 2 INP1 REG  
NODE ? INP2 COMB  
:  
SIGNATURE = V2_5/90  
STRING IN1 'A1 + /A2 + A3'  
GROUP BANK1 OUT1, OUT2  
...  
-----
```

Definitions

Groups of descriptors within the declaration segment are discussed next.³⁷

³⁷ Refer to the following topics, in this chapter, for additional details: AUTHOR, CHIP, COMPANY, DATE, GROUP, NODE, PATTERN, PIN, REVISION, SIGNATURE, STRING, and TITLE.

DECLARATION SEGMENT

Design Information

Design information includes statements that document basic information about the design. The following statements must appear in the order shown below.

- Title
- Pattern
- Revision
- Author
- Company
- Date
- Chip Device

These statements can be useful in describing the function of the design.

Pin Declarations

PIN and NODE statements required to process the design must follow the device type. Each statement defines the name, location number, and optional storage type assigned to each device pin in the design.

Special Definitions

Special definitions are optional statements you can include after PIN and NODE statements in the declaration segment.³⁸

- STRING statements can be used with any device.
- GROUP statements can be used with any device.
- SIGNATURE statements are device specific.

Though not required, it is a good idea to place each statement on a separate line. These statements can appear in any order.

³⁸ Refer to the following topics, in this chapter, for additional details: GROUP, SIGNATURE, and STRING.

DECLARATION SEGMENT

USE

The following error conditions can occur.

- If design information is incomplete, a warning is issued during processing.
- If multiple information statements appear, processing stops.
- If PIN and NODE statements are incomplete or incorrect, processing stops.
- For PAL and PLS devices, the signal list maps into a DIP package. For MACH devices, it maps into a PLCC package.

DEFAULT_BRANCH

This keyword begins the statement that defines the global default branch for state machines. There are three possibilities.

- A specific state
- The present state
- The next state listed on the left side of the state transition equation

The default branch will be executed if none of the conditions in the transition equation are satisfied and no local default is defined.

This statement can also utilize the complement array of certain devices.

Devices Supported: All synchronous registered PAL devices.

SYNTAX

Include the keyword once only, anywhere in the state segment of state-machine designs.

Syntax

DEFAULT_BRANCH	State	Comp
DEFAULT_BRANCH	HOLD_STATE	
DEFAULT_BRANCH	NEXT_STATE	

Example

STATE			;State Setup and Defaults
DEFAULT_BRANCH	INIT		;defaults to INIT state
DEFAULT_BRANCH	INIT	Comp	;uses complement array
DEFAULT_BRANCH	HOLD_STATE		;defaults to present state
DEFAULT_BRANCH	NEXT_STATE		;defaults to next state

...

Definitions

The next discussions explain parameters following the keyword DEFAULT_BRANCH. For more information about the various states and how they are used, refer to Use.

DEFAULT_BRANCH

State

State defines the default state enabled when the next state cannot be determined from the transition equations. The state name must adhere to the following guidelines.

- It must be unique.
- It can include up to 14 alphanumeric characters.
- It cannot include operators or reserved words.

Comp

Comp defines the name of the complement array node, which must be listed in the pin and node statements. This use results in shorter equations and fewer product terms; however, it slows device performance.

- You can use a complementary array node only for the PLS105, PLS167, PLS168, and PLS3016.
- You cannot use the complement array when specifying the next or hold state branches.

HOLD_STATE

When the next state cannot be determined from the transition equations, the machine will remain in its present state.

If no DEFAULT_BRANCH or local default branch are defined, the default becomes HOLD_STATE.

NEXT_STATE

When the next state cannot be determined from the transition equations, the machine will transition to the next state listed in the PDS file.

DEFAULT_BRANCH

USE

The software recognizes two types of default states, global and local.³⁹

- A local default state applies to a specific state and overrides the global default.

Using a DEFAULT_BRANCH statement can eliminate typing a LOCAL DEFAULT statement for each state.

- The global default state applies to all states.

The following guidelines apply.

- When the next state cannot be determined from the design, the local or global default provides the state so the machine knows where to branch next.
- When you include multiple DEFAULT_BRANCH statements, only the last one is used.

Note: DEFAULT_BRANCH does not work for undefined states.

³⁹ Refer to the following topics, in this chapter, for additional details: DEFAULT_OUTPUT, LOCAL DEFAULT, .OUTF, OUTPUT_HOLD, and STATE.

DEFAULT_OUTPUT

This keyword begins the statement that defines a global default, allowing you to specify the next output-pin value when the value cannot be determined from the design.

Devices Supported: All registered PAL devices.

SYNTAX

You include this keyword at the beginning of the state segment in state-machine designs. It must follow setup statements and precede state-assignment equations.

Syntax

DEFAULT_OUTPUT	Output_pins
----------------	-------------

Example

STATE		;State Setup and Defaults
...		
DEFAULT_OUTPUT	/OUT1	;State Equations

Definitions

Only the parameter following the keyword is defined below.

Output_pins

Output_pins defines default output pin values when they are not defined in the current state. You must separate output pin values with a blank; multiple blanks are reduced to one. You can specify the values you want regardless of polarity; polarity is adjusted automatically during processing.

- If you specify the output value with nothing before the pin name, a logical 1, or high, is assumed.
- If you specify the output value by placing a forward slash before the pin name, a logical 0, or low, is assumed.

DEFAULT_OUTPUT

Note: If the output pin and state bit are the same, **do not** use output equations nor a DEFAULT_OUTPUT statement.

Also: You can use the reserved word OUTPUT_HOLD as a default in PLS devices. In this case, the output pin value is held to its current state when a default is used.⁴⁰

USE

You can simplify output equations by assigning the most common output value as the default, then include an output value in the equation only when it differs from the default.⁴¹

⁴⁰ Refer to OUTPUT_HOLD, in this chapter, for additional details.

⁴¹ Refer to the following topics, in this chapter, for additional details: DEFAULT_BRANCH, LOCAL DEFAULT, .OUTF, and STATE.

EQUATIONS SEGMENT

This segment of the PDS file contains the Boolean equations you create to produce the desired device behavior.

Devices Supported: All PLD devices.

SYNTAX

You use the keyword, EQUATIONS, at the beginning of the equations segment in Boolean-based designs.

Syntax

```
;-----Boolean Equations Segment-----  
EQUATIONS  
Boolean equations
```

Example

```
EQUATIONS                               ;Equations Segment  
QA = Q2 * /Q1 * /Q0  
      + Q3 * Q2  
      + Q3 * Q1  
      + Q3 * Q0
```

Definitions

The following discussions define elements in the equations segment.

Equations

You use this keyword to define the beginning of the equations segment in a PDS file.

Boolean Equations

You assign equations that include an output pin or node name, an assignment operator, and an expression. Equations can be standard Boolean, Case, or IF-THEN-ELSE constructs.⁴²

⁴² Refer to BOOLEAN EQUATION, in this chapter, for additional details.

EQUATIONS SEGMENT

USE

The keyword EQUATIONS is required when Boolean equations follow.

Note: The PALASM 4 software allows both Boolean equations and state-machine descriptions in the same design file.

EXPRESSION

Expressions can be used in both Boolean and state-machine designs as part of a Boolean or functional equation, or in a state equation, or in equations in simulation commands.⁴³

Devices Supported: All PLD devices.
--

SYNTAX

You can include expressions in the equations, state, or simulation segments of any PDS file or in a separate simulation file.⁴⁴

Syntax

	Pn	Assignment Operator	Expression
<i>Example</i>			
EQUATIONS	Q1	=	Q1 * Q0 + /Q1 * /Q0 + /FRM1B * SOF * SUNK + RSTB
	...		

Definitions

All parameters are described next. Additional details are provided under Use.

Pn

Pn identifies the name of the output pin or node defined in the corresponding statement in the declaration segment of a PDS file. Once declared, you can include the name in an equation or expression.

⁴³ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, FUNCTIONAL EQUATIONS, and STATE EQUATIONS. Also, refer to NODE and PIN for details about including a forward slash to define polarity.

⁴⁴ Refer to Section II, Chapter 6, for details about using a separate simulation file.

Assignment Operator

The assignment operator is a symbol that defines the appropriate operation interpreted by the software when processing design files.⁴⁵

Expression

Expression assigns a group of signal values or product terms on the right side of an equation, represented as product, *, and sum, +, lines. The product operator is a logical AND function and the sum operator is a logical OR function.

All values in an expression for a PLD design are signal names that must be defined in PIN and NODE statements, in the declaration segment of the PDS file, before their use elsewhere.⁴⁶

Expressions in Boolean equations use Boolean-logic operators to combine the values of pins and nodes. **State equations** use Boolean expressions to define states and their outputs. **Functional equations** use Boolean expressions to define inputs.

USE

When using expressions, the following conventions should be observed.

- Place the expression on the right side of a Boolean, functional, or state equation.
- Use blanks or tab characters between pin or node names and logic operators.
- Place each part of the expression that uses the sum operator, +, on a separate line.

⁴⁵ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

⁴⁶ Refer to NODE and PIN, in this chapter, for details about including a forward slash to define polarity.

EXPRESSION

The following table lists the order of precedence for expressions without parentheses.

PRECEDENCE	OPERATOR/DEFINITION
1	/ NOT
2	* AND
3	+ OR
4	::: XOR
4	::: XNOR

FLOATING PINS AND NODES

A question mark in the location field of a PIN or NODE statement **floats** the pin or node. Each floating pin and node is automatically placed by the software during the fitting process.⁴⁷ Floating pins and nodes can increase the probability of a fit in all MACH-device designs.

Devices Supported: All MACH device designs.

SYNTAX

You can use the question mark only in the **declaration segment** of Boolean or state-machine designs.

Syntax

Keyword	Location_number	Name	Storage
<i>Example</i>			
Pin	?	Sensor1	Registered
Node	?	Equation1	Combinatorial
Pin	?	Output1	Latched

Definitions

Only descriptors following the keywords PIN or NODE are discussed.

Location_number

Place a question mark in the location-number field to float the corresponding pin or node. The signal is assigned to a specific pin automatically during compilation.

Important: Do not float NODE 1.

⁴⁷ You may want to group signals within the same MACH-device block. Refer to MACH_SEG, in this chapter, and to Chapter 11, in this section, for more information.

FLOATING PINS AND NODES

To assign a fixed location to a pin or node, you enter a number or range of numbers in the statement.

In PDS files produced from **schematics**, the question mark appears in the pin or node location field unless you assigned a fixed location. To assign a location in a schematic, you use the Part field 1 command on the OrCAD/SDT III Edit Part menu.⁴⁸

Name

Name defines the name of the pin or node. Each name must be unique and must follow the location field.

- Begin the name with an alpha character; use any combination of up to 14 upper- or lowercase alphanumeric characters: A–Z and 0–9.

Important: Keep names in a schematic less than or equal to 14 characters. Part and pin names in the schematic may be concatenated while the data is converted into PDS format. Any name longer than 14 characters is automatically truncated.

- Use underscore, `_`, as a connector and a forward slash to affect polarity; **no** other symbols or punctuation are allowed and no keywords, reserved words, or logic operators are allowed.

⁴⁸ Refer to Section III, Chapter 7, for more information about assigning attributes in a schematic.

FLOATING PINS AND NODES

Note: The forward slash is not supported for schematic-based designs.

Storage

Storage defines the optional storage type for a pin or node, and must follow the pin or node name.⁴⁹ Enter the reserved word or abbreviation listed below; the default is combinatorial.

- COMBINATORIAL or COMB
- REGISTERED or REG
- LATCHED or LAT

USE

Floating pins and nodes is recommended to assist the fitting process. You can assign a fixed location when needed; however, this may result in a no-fit situation.

Important: Do not float NODE 1.

⁴⁹ Refer to PAIR, in this chapter, for details about pairing a node with a pin.

FOR-TO-DO

This construct, also known as a FOR loop, defines the number of times to repeat a simulation block bounded by BEGIN and END statements.

Devices Supported: All PLD devices.
--

SYNTAX

You use this construct in either the simulation segment of a PDS file or in an auxiliary simulation file for any PLD design.

Syntax

```
FOR Variable := Start TO End DO
    BEGIN
                                Action Statement
    END
```

Example

```
SIMULATION
SETF /OE /CLOCK COUNT
FOR X := 1 TO 20 DO
    BEGIN
                                CLOCKF CLK
    END
```

Definitions

All elements of the statement are defined below.

Variable

Variable defines the index used in the FOR loop.

- A variable can include up to 14 alphanumeric characters and must end with :=, which is a convention, not the registered operator.
- A variable cannot be used elsewhere in the design file.

Start

Start defines the lower limit of the loop.

- Use only integers following the variable := and preceding the word TO.
- Ensure the number is greater than or equal to zero and less than the upper limit.

Important: If the start and end numbers are the same, the task is executed once.

End

END defines the upper limit of the loop. The number must be greater than or equal to zero.

- Use only integers following the word TO and preceding the word DO.
- Ensure the number is greater than or equal to zero and more than the lower limit.

Important: If the start and end numbers are the same, the task is executed once.

Action Statement

Action statement defines the simulation task to be repeated using this loop. The following rules apply.

- Enclose each statement with BEGIN and END.
- Use any valid simulation statements between BEGIN and END, including nested FOR loops.

USE

There is no limit to the number of constructs you can include in a design. However, minimal nesting makes the file easier to follow and faster to compile. You can nest FOR loops within other FOR loops and within the following statements and constructs.⁵⁰

- IF-THEN-ELSE
- WHILE-DO

It's best to use the WHILE-DO construct when you do not have a fixed number of repetitions and you prefer to base repetitions on a condition.

For example, you can test the FOR loop by nesting WHILE-DO or IF-THEN-ELSE constructs within the FOR loop, as shown below.

```
FOR X := 1 to 5 DO
  BEGIN
    IF X = 1 THEN
      BEGIN
        SETF A * B
      END
    END
  END
  ...
```

⁵⁰ Refer to the following topics, in this chapter, for additional details: CASE, IF-THEN-ELSE, SIMULATION, and WHILE-DO.

FUNCTIONAL EQUATIONS

Functional equations control signals that cannot be regulated directly through a pin or node. These are typically control signals associated with an input/output pin or node, such as set, clock, and three-state. Functional equations work like Boolean equations, in that the function is asserted when the expression is true.

Devices Supported: Refer to the specific functional-equation reserved word, such as .CLKF, for supported devices.

SYNTAX

You include functional equations anywhere in the equations segment of Boolean and state-machine designs.

Syntax

EQUATIONS			
Pn.Function	Assignment Operator		Expression
<hr/>			
<i>Example</i>			
EQUATIONS			
Q0.CLKF	=		CLOCK
Q1.SETF	=		SET * /RST
Q1.RSTF	=		RST * /SET

Definitions

All parameters are described next. Additional details are provided under Use.

Pn.Function

Pn.Function defines the pin or node name, as defined in the PIN and NODE statements, and the specific function, which can be any of the following.

FUNCTION	DEFINITION
.CLKF	Clock control
.CMBF	Register bypass control
.PRLD	Register preload control
.RSTF	Flip-flop power up or reset control
.SETF	Flip-flop preset control
.TRST	Three-state buffer control

FUNCTIONAL EQUATIONS

You **cannot** include multiple functional equations for the same pin or node. If you do, an error is reported during either assembly or fitting. However, you can use the group, string, and vector features to define signals, which is an excellent way to assign a function to several pins and nodes.

Note: The .CLKF statement is the only functional equation where you can use negative polarity on the left side of the equation to define a falling edge clock on devices that support this feature.

The .J, .K, .R, .S, .T, .T1, and .T2 equations are not functional equations.⁵¹

Assignment Operator

A symbol that defines a specific operation as interpreted by the software when processing design files.⁵² Only the equal sign, =, can be used with functional equations. The registered assignment operator, :=, can be used for registered operations.

Expression

Expression is a group of signal values or product terms, on the right side of an equation, represented as product and sum lines.⁵³ The product operator, *, is a logical AND function and the sum operator, +, is a logical OR function. You can use strings and vectors in an expression.

51 Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, .CLKF, .CMBF, .PRLD, .RSTF, .SETF, and .TRST.

52 Refer to ASSIGNMENT OPERATOR, in this chapter, for more information.

53 Refer to EXPRESSION, in this chapter, for additional details.

FUNCTIONAL EQUATIONS

All values in an expression for a PLD design are signal names that must be defined before their use. The name must be defined in PIN and NODE statements in the declaration segment of the PDS file.

USE

To use negative polarity on devices that support clock polarity, just omit the forward slash after the period, .. extension, for example, Q0./CLKF. For the PALCE29M16, place the forward slash before the equation, for example, /Q0.CLKF.

The defaults for each functional equation are as follows.

- .CLKF always defaults to the default clock pin for the device.
- .CMBF defaults to combinatorial.
- .PRLD defaults to ground.
- .SETF and .RSETF both default to a bank if they are part of a bank expression. Otherwise, they default to ground.
- .TRST defaults to VCC if an output equation is defined. Otherwise, it defaults to ground.

GND

You can include this reserved word in an equation to hold a pin, node, or functional equation unconditionally low. GND is always treated as logical zero.

Devices Supported: All PLD devices.

SYNTAX

You use the reserved word, GND, in the equations segment of Boolean and state-machine designs.

Syntax

Pn or Functional Equation	Assignment Operator	GND
------------------------------	---------------------	-----

Example

OUT4	=	GND
OE1.TRST	=	GND

Definitions

The element preceding the reserved word is described below.

Pn or Functional Equation

Pn or functional equation defines the element to be held low.

- The pin or node name defined in the PIN or NODE statement of the declaration segment
- The Pn.function defined in an earlier functional equation.⁵⁴

⁵⁴ Refer to the following topics, in this chapter, for additional details: FUNCTIONAL EQUATIONS, NODE, and PIN.

USE

GND is normally used in functional equations. You can enter 0 instead of GND anywhere you want an unconditional low value.⁵⁵

Important: You must define the GND pin in the pin statements.

⁵⁵ Refer to VCC, in this chapter, for additional details.

GROUP

This keyword clusters several pins or nodes under a single name. You can then use the group name in the equations or state segment of your design.

Devices Supported: All PLD devices.
--

SYNTAX

You use the keyword, GROUP, in the declaration segment of Boolean and state-machine designs.

Syntax

GROUP	Group_name	Pn_list
<i>Example</i>		
PIN 15 OUT1 COMB PIN 16 OUT2 COMB NODE 5 NB COMB GROUP	BANK1	OUT1 OUT2 NB
:		
EQUATIONS		
:		
BANK1 = IN1 * IN2 * IN3		

DEFINITIONS

Parameters following the keyword, GROUP, are defined below.⁵⁶

Group_name

Group_name is the name assigned to a cluster of pins or nodes. This name can then be used in the equations or state segments of a design to refer to the entire cluster without having to list them separately. Follow the rules below.

- Assign a unique name of up to 14 alphanumeric characters.

Do not use keywords, operators, or reserved words.

⁵⁶ Refer to MACH_SEG, in this chapter, for details about using a group name to cluster signals within a MACH device.

GROUP

- Place the name after the keyword GROUP and before the PIN or NODE statements.

The syntax example shows BANK1 as a group name.

Pn_list

This is a list of the pins or nodes, defined in the pin and node statement, that are associated with the group name.

- Place this list after the group name.
- Separate pin and node names by a single blank; multiple blanks are reduced to one.

Do not use commas or any other punctuation. You can use the range operator, [], to define a group of pins.

In the syntax example, the equation $BANK1 = IN1 * IN2 * IN3$ is automatically expanded into the three equations shown below.

```
OUT1 = IN1 * IN2 * IN3
OUT2 = IN1 * IN2 * IN3
NB = IN1 * IN2 * IN3
```

USE

You can place the group statement either before or after SIGNATURE and STRING statements. You can use the group name, wherever appropriate in the design file, in place of the defined group of pins or nodes. This can simplify the equations segment of the file by reducing the number of equations required.

- You can only use the group name on the left side of an equation.
- You can also use the group name to define pins or nodes in the simulation segment.

GROUP

Note: You can use a group name when controlling registers consisting of banks of flip-flops with common reset or other control lines.

For example, use a group name to combine four outputs that share a common reset line. The software then writes four .RSTF equations.⁵⁷

```
;DECLARATION SEGMENT
  PIN      2      A      REG
  PIN      3      B      REG
  PIN      4      C      REG
  PIN      5      D      REG
  PIN      6      E      REG
  GROUP    QRST   A      B      C      D

;EQUATIONS
  QRST.RSTF =      RES

;This expands to

  A.RSTF    =      RES
  B.RSTF    =      RES
  C.RSTF    =      RES
  D.RSTF    =      RES
```

⁵⁷ Refer to the following topics, in this chapter, for additional details: DECLARATION SEGMENT, NODE, and PIN.

IF-THEN-ELSE, EQUATIONS

This construct provides a conditional statement for Boolean logic. This construct literally means "if the condition is true, do this; if not, do that."⁵⁸

Devices Supported: All PLD devices.
--

SYNTAX

You use this construct in the equations segment of Boolean and state-machine designs.

Syntax

```
IF (Condition) THEN
    BEGIN
        Boolean equations
    END
ELSE
    BEGIN
        Boolean equations
    END
```

Example

```
EQUATIONS
IF (/I1,/I2 = #b11) THEN
    BEGIN
        O3 = A * B
    END
ELSE
    BEGIN
        O3 = A * /B
    END
...

```

Definitions

Both elements of the statement are defined next.

⁵⁸ Refer to IF-THEN-ELSE and SIMULATION, in this chapter, for details on using this construct in the simulation segment or separate simulation file of a design.

IF-THEN-ELSE, EQUATIONS

Condition

Condition defines any Boolean expression whose form consists of a pin, signal, range, or vector, an equal sign, =, and a binary, decimal, hexadecimal, or octal radix number.

- You can use more than one condition if you separate them by commas.

The software ANDs multiple conditions together.

- You can use parentheses to enclose the condition; they are optional, but it is better to include them. You can nest parentheses.
- You cannot use group names or arithmetic expressions.
- You can use a test condition in place of any variable name in a Boolean expression as in the example, $A * B * (C, D = 1)$.

The software ANDs conditions with vectors. For example:

<pre>IF (A[1..3]) becomes IF (A[1] * A[2] * A[3]) IF (/A[1..3]) becomes IF (!(A[1] * A[2] * A[3]))</pre>
--

Boolean Equation

This equation defines any Boolean equation or set of equations, as well as IF-THEN-ELSE, and CASE constructs. The equation must be enclosed by BEGIN and END statements.

IF-THEN-ELSE, EQUATIONS

USE

You can specify how the software treats default values for IF-THEN-ELSE constructs by selecting one of the following options on the File/Set up/Logic Synthesis Options form.

OPTIONS	DEFINITIONS
Don't care	Unspecified default conditions are assumed to be don't care.
Off	Unspecified default conditions are assumed to be false.

The don't-care option requires you specify both the on and off sets. The off option requires you to specify only the on sets; the software assumes all other conditions to be off.

You may lose signals from the design if you select the don't-care option and do not specify all the default conditions. If the software treats these signals as don't care, they will be eliminated from the design during logic reduction.

Important: When translating designs created with PLPL, you must select the off option because PLPL treats unspecified default conditions as false.

There is no limit to the number of constructs you can have in your design. However, minimal nesting makes the file easier to follow and faster to compile.⁵⁹

⁵⁹ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, CASE, IF-THEN-ELSE, and SIMULATION.

IF-THEN-ELSE, EQUATIONS

You can use an IF clause without an ELSE clause, but no logic is defined when the IF clause is false. In the case of multiple or nested IF-THEN-ELSE statements, an ELSE clause always matches the last IF-THEN clause.

You can nest IF-THEN-ELSE constructs within CASE and other IF-THEN-ELSE constructs. The following examples show how the software expands IF-THEN-ELSE constructs.

IF -THEN-ELSE	EXPANDS TO
IF (A) THEN B = 1	$B = 1 \cdot A$ that is $B = A$
IF (A) THEN B = 1 ELSE B = 0	$B = VCC \cdot A$ that is $B = A$ $/B = VCC \cdot /A$ that is $/B = /A$
IF (A) THEN B = 0	$/B = VCC \cdot A$ that is $/B = A$
IF (A) THEN /B = 1	$/B = 1 \cdot A$ that is $/B = A$

IF-THEN-ELSE, SIMULATION

This construct provides conditional statements during simulation. This construct literally means "if the condition is true, do this; if not, do that."

Devices Supported: All PLD devices.
--

SYNTAX

You use this construct in Boolean and state-machine designs.

Syntax

```
IF (Condition) THEN
  BEGIN
    Task
  END
ELSE
  BEGIN
    Task
  END
```

Example

```
SIMULATION
SETF /OE /CLOCK COUNT
  IF (J = 5) THEN
    BEGIN
      CHECK Q0
    END
  ELSE
    BEGIN
      CHECK /Q0
    END
```

...

Definitions

Both elements of the statement are defined below.

Condition

Condition defines any Boolean expression.

- You can use more than one condition if you separate them by commas.

The software ANDs multiple conditions together.

- You can use parentheses to enclose the IF condition. However, you cannot nest parentheses.

IF-THEN-ELSE, SIMULATION

The condition can be any Boolean expression of logic signals or mathematical equality: =, >, <, >=, <=, and <>.

Task

Task defines the simulation task the software performs during the IF-THEN-ELSE loop. You use BEGIN and END statements to enclose the task and indent them to make your PDS file easier to follow.

USE

There is no limit to the number of constructs you can have in your design. However, minimal nesting makes the file easier to follow and faster to compile.⁶⁰

You can nest IF-THEN-ELSE constructs within other IF-THEN-ELSE constructs and with the following statements.

- CASE
- FOR-TO-DO
- WHILE-DO

If the condition is false when the construct is reached, the task is not executed.⁶¹

Note: If you nest the IF-THEN-ELSE construct in a FOR-TO-DO construct, the condition can also be the index variable of the FOR-TO-DO construct. You cannot use an index variable outside its defining FOR-TO-DO construct.

⁶⁰ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, EQUATIONS SEGMENT, CASE, and IF-THEN-ELSE.

⁶¹ Refer to the following topics, in this chapter, for additional details: CASE, FOR-TO-DO, and WHILE-DO.

.J EQUATION

This equation defines when to set the J input on J type flip-flops high.

Devices Supported: PALCE610.

SYNTAX

You use the .J equation in the equations segment of Boolean or state-machine designs.

Syntax

	Pn.J	Assignment Operator	Expression
<i>Example</i>			
	EQUATIONS		
	...		
	Q1.J	=	IN1 * /IN2
	...		

Definitions

All parameters are defined below.

Pn.J

Pn.J identifies the pin or node associated with the J flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

Assignment operator defines a specific operation as interpreted by the software when processing design files.⁶²

Expression

Expression identifies the logic that defines when the input on .J type flip-flops is set high. In the example, when IN1 is true and IN2 is false, the flip-flop associated with the pin or node Q1 is set high.

⁶² Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .J equation anywhere in the equations segment. Follow the rules below.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and processing stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.J is not allowed.
- You can use group, string, and vector notation to define signals. This is an excellent way to assign the .J equation to several pins.⁶³

⁶³ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

.K EQUATION

This equation defines when to set the K input on K-type flip-flops high.

Devices Supported: PALCE610.

SYNTAX

You use the .K equation in the equations segment of Boolean or state-machine designs.

Syntax

	Pn.K	Assignment Operator	Expression
<i>Example</i>			
EQUATIONS			
...			
	Q1.K	=	IN1 * /IN2
...			

Definitions

All parameters are defined below.

Pn.K

Pn.K identifies the pin or node associated with the K flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.⁶⁴

Expression

Expression identifies the logic that defines when the input on .K type flip-flops is set high. In the syntax example, when IN1 is true and IN2 is false, the flip-flop associated with the pin or node Q1 is set high.

⁶⁴ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .K equation anywhere in the equations segment. Follow the rules below.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.K is not allowed.
- You can use group, string, and vector notation to define signals. This is an excellent way to assign the .K equation to several pins.⁶⁵

⁶⁵ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

LATCHED

This reserved word defines the output data storage type on devices that allow latched outputs. A latched output functions as a combinatorial output until the data is latched. Once latched, the last data present is held regardless of input changes.

Device Support					
PAL10H20EG8	PAL10H20G8	PALCE29M16	PALCE29MA16	PAL16V8HD	MACH 2

SYNTAX

You include this optional reserved word in the PIN or NODE statement of Boolean and state-machine designs.

Syntax

Pn	Number	Name	Storage
<i>Example</i>			
PIN	3	OUT1	LATCHED
NODE	4	OUT2	LAT

Definitions

Only the reserved word is discussed below.

Storage

Storage defines the pin or node storage type. If you do not specify a storage type, combinatorial is the default.

- Place the reserved word LATCHED after the pin or node name in the corresponding statement.
- Use either the complete word LATCHED or the three-letter abbreviation, LAT.

USE

There are two ways to enter the storage type.

- Use the declaration segment form: Select the Storage field, display the option list, and select an option.

- Type the storage value in the appropriate PIN or NODE statement in the PDS file using a text editor.

The PALCE29M16 and PALCE29MA16 have programmable latches. The polarity of the latch can be inverted by placing a forward slash before the .CLKF functional equation. The default is active-low enable. The following is an example of an active-low enable equation.

```
PIN 1 EN
PIN 3 OPIN
...
/IOPIN.CLKF = EN
```

The following equation provides an active high enable.

```
IOPIN.CLKF = EN
```

The final polarity of the latch enable as seen from outside the chip is determined by the following conditions.

- The polarity of the latch enable as defined in the PIN statement
- The polarity of the .CLKF functional equation
- The polarity of the latch enable on the right-hand side of the .CLKF functional equation

The latch enable input controls whether the flip-flop is latched or not. On the PAL10H20EG8 and PAL10H20G8 the latch is transparent when the gate pin is low. The latch is enabled when the pin is high.⁶⁶

⁶⁶ Refer to the following topics, in this chapter, for additional details: .CLKF, COMBINATORIAL, PIN, and REGISTERED.

LOCAL DEFAULT

This branch of a state transition equation is executed if none of the conditions in the equation are satisfied. Local defaults override global defaults.⁶⁷

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Include the local default (state branch) in state transition equations of state-machine designs.

Syntax

```
State1 := condition1 -> state2
        + condition2 -> state3
        +-> default state
```

Example

```
STATE

;State transition equations
RED := NOTRAFFIC -> GRN
      + RED2 -> YLW
      +-> RED

...
```

Definitions

Only the term Default is discussed below.

Default State

This parameter defines the next state in a state-machine transition equation when that state cannot be determined from previous conditions.

⁶⁷ Refer to DEFAULT_BRANCH and DEFAULT_OUTPUT, in this chapter, for additional details regarding global defaults.

LOCAL DEFAULT

- You must use the default operator, `+>`, to define a local default.
- You can define only one local default state for each state transition equation.

USE

Defaults ensure the state machine does not behave unpredictably if none of the conditions in the state transition equation is satisfied.

A local default overrides any global default.

The local default branch must be the last branch in the state-transition equation.

Global defaults are defined by the `DEFAULT_BRANCH` statement.

MACH_SEG_A
MACH_SEG_B
MACH_SEG_C
MACH_SEG_D

These reserved words can be used as the name in a GROUP statement to cluster signals within a single block of a MACH device. The same control logic is applied to all signals in the block.

Once declared, you can include the group name in any equation rather than writing a separate equation for each pin or node in the group.

Devices Supported: All MACH device designs.

SYNTAX

You can use the reserved word only in the GROUP statement in the **declaration segment** of Boolean or state-machine designs.

Constructs

GROUP	Group_name	Pn_list
-------	------------	---------

Example

pin/node statements ...														
GROUP	MACH_SEG_A	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 0 10px;">R[0]</td> <td style="padding: 0 10px;">R[1]</td> <td style="padding: 0 10px;">R[2]</td> </tr> <tr> <td style="padding: 0 10px;">R[3]</td> <td style="padding: 0 10px;">0[0]</td> <td style="padding: 0 10px;">0[1]</td> </tr> <tr> <td style="padding: 0 10px;">0[2]</td> <td style="padding: 0 10px;">0[3]</td> <td style="padding: 0 10px;">0[4]</td> </tr> <tr> <td style="padding: 0 10px;">0[5]</td> <td style="padding: 0 10px;">0[6]</td> <td style="padding: 0 10px;">0[7]</td> </tr> </table>	R[0]	R[1]	R[2]	R[3]	0[0]	0[1]	0[2]	0[3]	0[4]	0[5]	0[6]	0[7]
R[0]	R[1]	R[2]												
R[3]	0[0]	0[1]												
0[2]	0[3]	0[4]												
0[5]	0[6]	0[7]												

Definitions

Only descriptors following the keyword, GROUP, are discussed.

MACH_SEG_A
MACH_SEG_B
MACH_SEG_C
MACH_SEG_D

These reserved words identify the block of a MACH device within which the named group of signals will be clustered. The block you specify must be **one** of the following.

- **MACH 110**
MACH_SEG_A
MACH_SEG_B

MACH_SEG_A, MACH_SEG_B, MACH_SEG_C, MACH_SEG_D

- **MACH 210**
MACH_SEG_A
MACH_SEG_B
MACH_SEG_C
MACH_SEG_D

Once declared, you can use the name **either** on the left side of an equation, as shown under Use, **or** to define pins or nodes in the simulation segment or file.

In PDS files produced from converted schematic designs, signals are clustered into one block when a common value is found in Part field 2 of certain macros.⁶⁸

Pn_list

Pn_list identifies the pins or nodes to be included in the group. This list must follow the group name.

- Names must match those used in previous PIN or NODE statements.

You can include a range operator, [], to define a group of pins or nodes if they are so defined in previous statements.

- Blanks or tab characters should be used to separate each pin or node listed; no [Return] characters are allowed.

⁶⁸ Refer to Section III, Chapter 7, for more information about assigning attributes in a schematic.

MACH_SEG_A, MACH_SEG_B, MACH_SEG_C, MACH_SEG_D

USE

Using the reserved word as a group name can be helpful when modifying a design that doesn't fit.⁶⁹ The following example shows a declared group, MACH_SEG_A, and its use in an equation in the PDS file.

```
;... pin/node statements ...  
GROUP MACH_SEG_A R[0] R[1] R[2] R[3] O[0] O[1] O[2] O[3] O[4] O[5]  
        O[6] O[7]  
  
;... equations ...  
MACH_SEG_A.TRST = IN [1]
```

The equation above enables all outputs in block A when input IN [1] is high. The next example shows how the previous equation is automatically expanded during software processing.

```
R[0].TRST = IN[1]  
R[1].TRST = IN[1]  
...  
R[3].TRST = IN[1]  
O[0].TRST = IN[1]  
O[1].TRST = IN[1]  
...  
O[7].TRST = IN[1]
```

⁶⁹ Refer to Section II, Chapter 5, for strategies to use when a design does not fit.

MASTER_RESET

This reserved word selects the preset function on PLS devices that provide a preset/enable pin.

Devices Supported		
PLS105	PLS167	PLS168

SYNTAX

You use this reserved word in the state segment of state-machine designs.

Syntax

MASTER_RESET

Example

```
MEALY_MACHINE
MASTER_RESET
OUTPUT_HOLD OUT1 OUT2
```

Definitions

Only the reserved word is defined below.

MASTER_RESET

This reserved word dedicates the preset/enable pin to active high preset. Conversely, the reserved word OUTPUT_ENABLE dedicates the preset/enable pin to active-low output enable.

- You can place MASTER_RESET anywhere within the setup and default statements.

It must precede the state-assignment and transition equations.

- You cannot use both MASTER_RESET and OUTPUT_ENABLE in the same design file.

When the device is preset, the state machine goes to the state which has a value of all 1s.

MASTER_RESET

USE

The software selects preset as the default if you do not use MASTER_RESET or OUTPUT_ENABLE. If you write a .SETF equation and do not use MASTER_RESET or OUTPUT_ENABLE, the software selects preset. If you write a .TRST equation, the software selects output enable.⁷⁰

⁷⁰ Refer to OUTPUT_ENABLE and STATE, in this chapter, for additional details.

MEALY_MACHINE

This reserved word identifies the type of state machine you are designing.

Devices Supported: All PLD devices.
--

SYNTAX

Include the reserved word in the state segment of state-machine designs.

Syntax

```
MEALY_MACHINE
```

Example

```
STATE ;State Setup and Defaults  
MEALY_MACHINE
```

Definitions

Only the reserved word is defined below.

MEALY_MACHINE

This reserved word defines a state-machine design as one of two possible types, either Mealy or Moore. If you do not define a type in the state segment of the design, the program defaults to Mealy machine.

A Mealy machine determines its outputs from the present state and inputs.⁷¹

USE

You can place this reserved word statement anywhere within the STATE segment. However, for design clarity, the following guidelines are advised.

- Place the reserved word statement at the beginning of the STATE segment before the state global defaults.

⁷¹ Refer to MOORE_MACHINE and STATE, in this chapter, for additional details.

MEALY_MACHINE

- Use the reserved word statement only once in a file.

The software ignores redundant state-machine definitions.

You cannot have both **MEALY_MACHINE** and **MOORE_MACHINE** statements in the same design file. If you want to include both types of state machines in the design file, only one can be written using state-machine syntax. The other must be written using Boolean equations.

MINIMIZE_OFF MINIMIZE_ON

These two keywords allow you to specify equations that will not undergo logic reduction during the minimization process.

Devices Supported: All PLD devices.
--

SYNTAX

You use these keywords in the equations segment of Boolean designs.

Syntax

```
MINIMIZE_OFF  
Boolean equations  
MINIMIZE_ON
```

Example

```
MINIMIZE_OFF  
OUT1 = A * B :+: C  
MINIMIZE_ON
```

Definitions

All parameters are defined below.

Boolean Equation

Boolean equations control storage inputs and other combinatorial functions to produce the desired device behavior. These equations form the backbone of any PDS file containing a Boolean description.⁷²

MINIMIZE_OFF

This keyword prevents logic reduction from occurring on the equation or equations that follow. You can suppress logic reduction only for an entire equation. **Do not** place this keyword in the middle of an equation.

MINIMIZE_ON

This keyword reactivates logic reduction on Boolean equations after it has been suppressed using the keyword MINIMIZE_OFF.

⁷² Refer to BOOLEAN EQUATION and EXPRESSION, in this chapter, for additional details.

MINIMIZE_OFF, MINIMIZE_ON

USE

During the minimization process, all Boolean equations are reduced to their simplest form. However, sometimes the results of logic reduction may produce equations that are not functionally what you intend. You can selectively skip the logic reduction on certain equations by bracketing them within MINIMIZE_ON and MINIMIZE_OFF commands.

These commands do not affect other logic conversions that occur during the minimization process. The example below shows that parentheses are expanded for all expressions regardless of whether logic reduction is suppressed or not.

BEFORE MINIMIZATION	AFTER MINIMIZATION
MINIMIZE_OFF	MINIMIZE_OFF
01 = A*B + A*/B	01 = A*B + A*/B
02 = /(A+B)	01 = /A * /B
MINIMIZE_ON	MINIMIZE_ON
03 = A*B + A*/B	03 = A
04 = /(A+B)	04 = /A * /B

You can have as many pairs of MINIMIZE_ON and MINIMIZE_OFF commands as you wish. The software ignores redundant commands.

You need not place the MINIMIZE_OFF command on its own line, but it makes the design easier to follow.

MOORE_MACHINE

This reserved word identifies the type of state machine you are designing.

Devices Supported: All PLD devices.
--

SYNTAX

Include the reserved word in the state segment of state machine designs.

Syntax

```
MOORE_MACHINE
```

Example

```
STATE                                ;State Setup and Defaults
MOORE_MACHINE
```

Definitions

Only the reserved word is defined below.

MOORE_MACHINE

This reserved word defines a state-machine design as one of the two possible types, either Moore or Mealy. If you do not define a type in the state segment of the design, the program defaults to Mealy machine.

A Moore machine determines its outputs from the present state only.⁷³

USE

You can place the reserved word statement anywhere within the STATE segment. However, for design clarity, the following guidelines are advised.

- Place the reserved word statement at the beginning of the STATE segment before the state global defaults.

⁷³ Refer to MEALY_MACHINE and STATE, in this chapter, for additional details.

MOORE_MACHINE

- Use this reserved word statement only once in a file.

The software ignores redundant state-machine definitions.

You cannot have both MOORE_MACHINE and MEALY_MACHINE statements in the same design file. If you want to include both types of state machines in the design file, only one can be written using state-machine syntax. The other must be written using Boolean equations.

NODE

This keyword is the logical name assigned to a feedback signal or internal-control product term. When used in the declaration segment of a PDS file, this keyword allows you to assign names and attributes to internal device nodes. A node can be one of the following.

- A buried flip-flop or flip-flop feedback line
- An internal-control line, such as a global reset, preset, or observability line
- A complement array term

Devices Supported

PAL22V10	PAL32VX10	PAL18U8	PAL23S8	PAL26V12	PALCE29M16
PALCE29MA16	PALCE610	PLS30S16	PLS105	PLS167	PLS168
MACH 1	MACH 2				

SYNTAX

Include this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

NODE	Location_number	Name	Storage
------	-----------------	------	---------

Example

...			
NODE	12	ST	REG
...			

Definitions

Constructs following the keyword are defined below. Additional details are provided under Use.

Number

Number identifies the node number exactly as defined in the device reference.

Name

Name defines the node name. Each name must be unique and must follow the location_number field.

- Begin the name with an alpha character; use any combination of up to 14 upper- or lowercase alphanumeric characters: A–Z and 0–9.

Important: Keep names in a schematic equal to or less than 14 characters. Part and node names in the schematic may be concatenated when data is converted into PDS format. Any name longer than 14 characters is automatically truncated.

- Use underscore as a connector and a forward slash to affect polarity; **no** other symbols or punctuation are allowed and no keywords, reserved words, or logic operators are allowed.

Note: The forward slash is not supported for schematic-based designs.

Also: Polarity works differently for nodes used as inputs on the PALCE29M16 and PALCE29MA16.⁷⁴

You can use ranges and vector notation to define node names. You must use the same number of nodes as names in ranges and vector notation. All nodes defined within a range or vector notation have the same storage type and polarity attributes.⁷⁵

Storage

Storage defines the optional storage type for a node, which must follow the node name.⁷⁶ Enter the reserved word or abbreviation listed below; the default is combinatorial.

⁷⁴ Refer to Chapter 11, in this section, for more information.

⁷⁵ Refer to VECTOR, in this chapter, for more information on vector notation and ranges.

⁷⁶ Refer to PAIR, in this chapter, for details about pairing a node with a pin.

NODE

- COMBINATORIAL or COMB
- REGISTERED or REG
- LATCHED or LAT

Important: COMBINATORIAL is a valid node storage attribute only for MACH devices. For non-MACH PLDs, you must specify either REG or LAT. Otherwise, the software issues an error during compilation.

USE

NODE statements must follow the CHIP statement. Use a separate line for each NODE statement. You do not have to place the NODE statements in numerical order. You can only place COMMENT statements between NODE statements, not within the NODE statement.

Declare only the nodes you are using. The software automatically assigns the name NC, no connect, to all nodes that are not declared.

PIN or NODE statements in the current version of the software differ from the pin list of previous versions. However, the old syntax is fully compatible with the new.

Use of the NODE statement is device dependent.⁷⁷

⁷⁷ Refer to Chapter 11, in this section, and the following topics, in this chapter, for additional details: BOOLEAN EQUATION, CHIP, DECLARATION SEGMENT, FLOATING PINS AND NODES, LATCHED, OPERATOR, PIN, and REGISTERED.

OPERATOR

This is a general term that describes any symbol interpreted by the software when processing design files. It can be a mathematical term such as "+", for OR, a defining term such as ":", for registered, or a descriptive term such as "#", for radix. The following table defines each operator and provides an example.

OPERATOR	DEFINITION	EXAMPLE
/	NOT	/A
*	AND	A * B
+	OR	A + B
:+:	XOR	A :+: B
:*:	XNOR	A :* B
=	COMBINATORIAL	INPUT1 = A * B
:=	REGISTERED equation	INPUT1 := A * B
:=	STATE EQUATION	STATE1 := START -> END
*=	LATCHED	INPUT1 *= A * B
->	STATE TRANSITION	STATE1 := START -> END
+>	LOCAL DEFAULT	+> RED -> WAIT
;	COMMENT	;set low before clocking
,	Literal separator	IN[1,3,4] IN[1..4,6..9]
..	Range	INPUT[0..9]
:	CASE value	0,1:
[]	Term brackets	INPUT[0..9]
()	EXPRESSION	IN1 = (A * B) (C * D * F)
{}	Substitute	OUT1 = A * B * C OUT2 = {OUT1} * F>
>	Greater than	IF A > 2 THEN...
<	Less than	WHILE A < 2 DO...
<>	Not equal to	IF A <> 2 THEN...
<=	Less than or equal to	WHILE A <= 2 DO...
>=	Greater than or equal	WHILE A >= 2 DO...
%	Don't care	DEFAULT_OUTPUT %OUT1
?	CHECK clash	-----??????
' '	String delimiters	STRING INPUT 'A1 + /A2'
#b	Binary radix	#b101000
#d	Decimal radix	#d40
#o	Octal radix	#o50
#h	Hexadecimal radix	#h28B
Space, tab	Separator	PIN 2 IN1 REG

.OUTF

This keyword defines state output equations for Mealy and Moore machines.

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS1	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Include .OUTF in output equations in the state segment of state-machine designs.

Syntax

Moore machines	Statename.OUTF	=	Output expression
Mealy machines	Statename.OUTF	=	Condition 1 -> Output 1 + Condition 2 -> Output 2 ... + Condition n -> Output n +> Local default

Example

Moore machines	... TWO.OUTF	=	/CNT2 * CNT1 * /CNT0
Mealy machines	TWO.OUTF	=	RUN_UP -> /CNT2 * CNT1 * /CNT0 ... TEST -> CNT2 * CNT1 * CNT0 +> /CNT2 * /CNT1 * /CNT0
	...		

Definitions

The construct immediately preceding, and all constructs following, the keyword are defined below. Additional details are provided under Use.

Statename

Statename identifies the name of the state as specified in the state assignments or state-transition equations. It must be unique and it can have up to 14 alphanumeric characters.

Outputs

Outputs are pin names with their appropriate logic sense to create the desired logic values. Outputs are separated by an asterisk, *. In the syntax example, when the Moore machine is in state TWO, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

When the Mealy machine is in state TWO and the inputs match the condition defined as RUN_UP, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

You specify the output values regardless of pin polarity. The software adjusts polarity as necessary.

Conditions

In a Mealy machine, the outputs depend on the current state and the current input conditions. This field defines the condition under which the specified output will occur. The condition names must be defined in the conditions section of the state-machine design.

If the condition consists of a single input, the input name may be used in place of the condition name. You can use VCC to specify an unconditional output.

Moore machine outputs do not have conditions since their outputs are determined only by the present state.

Local Default

This output is generated if none of the conditions is satisfied. Local defaults are valid only for Mealy machines and will override global defaults.

.OUTF

USE

You can place output equations anywhere within the state segment. You may prefer to have all of the output equations after all of the state equations, or follow a state equation with its corresponding output equation.

You can use the following operators in state output equations.

OPERATOR	DEFINITION
->	Conditional output for Mealy machines
+>	Local default for a Mealy machine
=	Combinatorial assignment operator
:=	Registered assignment operator

For Mealy machines, conditions in an .OUTF equation don't have to match conditions in the state-transition equations. However, conditions that don't match are unusual.

You can omit the output equations if you use the state bits as outputs. You do this by making the output pins the same as the state bits and performing manual state bit assignment.⁷⁸

If you omit output equations, don't use the following constructs.

- DEFAULT_OUTPUT
- OUTPUT_HOLD

You can define some outputs with state bits and some with output equations.

If you don't use the state bits as outputs, you must specify output equations. Default output specifications are optional.

⁷⁸ Refer to Section II, Chapter 4, for additional details regarding assigning state bits.

Registered Mealy machine outputs are valid one clock cycle after reaching the new state. Combinatorial Mealy and Moore machine outputs and registered Moore machine outputs are valid upon reaching the new state. Undefined output pins have a don't-care value.⁷⁹

⁷⁹ Refer to the following topics, in this chapter, for additional details: `CONDITIONS`, `DEFAULT_BRANCH`, `DEFAULT_OUTPUT`, `LOCAL_DEFAULT`, `MEALY_MACHINE`, `MOORE_MACHINE`, `OPERATOR`, `OUTPUT_HOLD`, `STATE`, `STATE ASSIGNMENT EQUATION`, `STATE EQUATION`, and `STATE TRANSITION EQUATION`.

OUTPUT_ENABLE

This reserved word selects the output-enable function of the preset/enable pin.

Devices Supported

PLS105	PLS167	PLS168
--------	--------	--------

SYNTAX

Use this reserved word in the state segment of state-machine designs.

Syntax

OUTPUT_ENABLE

Example

```
STATE
    ;State Setup and Defaults
    ...
    MEALY_MACHINE
    OUTPUT_ENABLE
    OUTPUT_HOLD
    ;State Bit Assignment
    ...
```

Definitions

Only this reserved word is defined below.

OUTPUT_ENABLE

OUTPUT_ENABLE sets the preset/enable pin to output enable. You can place OUTPUT_ENABLE anywhere in the setup and default section. It must precede the state assignment and transition equations.

USE

When you use OUTPUT_ENABLE, the software dedicates the preset/enable pin to active-low output enable. Conversely, when you use MASTER_RESET, the software dedicates the preset/enable pin to active-high preset. You cannot use both OUTPUT_ENABLE and MASTER_RESET in the same design file. Follow the guidelines outlined next.

OUTPUT_ENABLE

- If you do not use MASTER_RESET or OUTPUT_ENABLE, the software selects preset as the default.
- If you write a .SETF equation and do not use MASTER_RESET or OUTPUT_ENABLE, the software selects preset as the default.
- If you write a .TRST equation, the software selects output enable as the default.⁸⁰

⁸⁰ Refer to MASTER_RESET and STATE, in this chapter, for additional details.

OUTPUT_HOLD

This keyword defines a global default that allows you to hold the present output value when there is no output defined for the current state and input condition.

Devices Supported

PLS105	PLS167	PLS168
--------	--------	--------

SYNTAX

Use this keyword in the state segment of state-machine designs.

Syntax

OUTPUT_HOLD	Output_pins
-------------	-------------

Example

```
STATE
;State Setup and Defaults
...
OUTPUT_HOLD      OUT1 OUT2 OUT3
...
;State Bit Assignment
...
```

Definitions

The parameter following the keyword is defined below. Additional details are discussed under Use.

Output_pins

Output_pins identifies the user-defined output pins that are held when the next output value cannot be determined from the equations segment of the state-machine design. Use a blank to separate the output pins; multiple blanks are reduced to one blank.

USE

You must place the OUTPUT_HOLD statement at the beginning of the state segment. It can follow setup statements but must precede any state assignment and transition equations.

If you use the state bits as outputs, do not use OUTPUT_HOLD.

OUTPUT_HOLD

You can reduce the number of output equations required by specifying the global default as `OUTPUT_HOLD`. Using this technique, you only need to write equations for outputs that differ from the default.

PAIR

This keyword is an optional attribute in a PIN or NODE statement you use to direct input or output pairing.

- **Input pairing:** include the PAIR attribute in a PIN statement to logically associate an input pin with a node.
- **Output pairing:** include the PAIR attribute in a NODE statement to logically associate a node with an output pin.

Devices Supported: MACH-device designs only.

Input pairing can only be implemented in MACH devices with buried macrocells. Output pairing can be implemented in all MACH devices.

SYNTAX

You can use the PAIR keyword in the declaration segment of Boolean or state-machine designs as shown below.

Required Elements			Optional Attributes		
<i>Syntax</i>					
Keyword	Location_ number	Name	Storage	Pair	Pn_name
<i>Example, Input Pairing</i>					
Pin	?	I1	---	Pair	R1
Node	?	R1	Reg	---	---
<i>Example, Output Pairing</i>					
Node	?	L2	Combinatorial	Pair	Output1
Pin	?	Output1	Comb	---	---

Definitions

The following discussions pertain only to the descriptors for NODE and PIN statements.

Location

Location defines the location of the pin or node. When both the pin and node locations are fixed, you must assign both to the same macrocell.

Name

Name defines the name of the pin or node.

Note: An optional forward slash is supported here.

Storage

Storage defines the optional storage type for the pin or node; the default is combinatorial.

Important: Combinatorial is not a valid **node** storage attribute for input pairing. When specifying an **Input** pair, use the registered or latched attribute in the **NODE** statement.

Pair

Include this optional keyword to indicate input pairing in a PIN statement or output pairing in a NODE statement. PAIR cannot be abbreviated. The keywords OPAIR and IPAIR are also valid, and denote output and input pairing, respectively.

- Output pairs are generated when there are duplicate pin/node equations.
- Input pairs are generated when a buried input node is equated to an input pin.

Pairing occurs automatically during compilation unless you enable manual pairing by typing the letter N beside the Use automatic pin/node pairing field on the Logic Synthesis Options form.

Recommendation: It is best to enable the automatic pin/node pairing option on the Logic Synthesis Options form.

Pn_name

Pn_name defines the pin or node that completes the pair. Each name must be unique and follow the keyword PAIR.

PAIR

Note: A node and its corresponding output pin should not be paired if the three-state control line is tied to ground. This permanently disables the output pin.

Also: No forward slash is allowed in the pin or node name following the keyword PAIR.

USE

When paired, the pin and node are logically associated with the same macrocell. Input pairing applies only to registered or latched inputs.

PATTERN

This keyword begins the statement defining the design's pattern, which is useful for documentation purposes.

Devices Supported: All PLD devices.
--

SYNTAX

Use the PATTERN keyword in the declaration segment of Boolean and state-machine designs.

Syntax

PATTERN

Design_pattern

Example

```
TITLE
PATTERN
REVISION
AUTHOR
COMPANY
DATE
CHIP COUNTER PAL16R8
```

F00345

Definitions

Only the construct following the keyword PATTERN is defined below.

Design_pattern

Design_pattern can be any combination of up to 60 alphanumeric characters. The following rules apply.

- Place the PATTERN statement after the title and before revision, as shown in the syntax example. The software assumes this order.
- Write the PATTERN statement on one line.
- Do not use a dollar sign.

You can use reserved words within the PATTERN statement.

USE

The PATTERN statement is optional. If you do not include it, the software issues a warning and continues processing the file. If you include multiple PATTERN statements, the software issues an error and stops processing the file.⁸¹

⁸¹ Refer to the following topics, in this chapter, for additional details: AUTHOR, COMPANY, DATE, DECLARATION SEGMENT, REVISION, and TITLE

PIN

This keyword begins a statement that allows you to assign names and attributes to device pins.

Devices Supported: All PLD devices.
--

SYNTAX

Include the PIN keyword in the declaration segment of Boolean and state-machine designs.

Syntax

	PIN	Location_number	Name	Storage
<i>Example</i>				
	...			
	PIN	1	IN1	COMB
	PIN	8	IN2	REG
	PIN	16	OUT	
	...			

Definitions

Constructs following the keyword PIN are defined below. Additional details are provided under use.

Location_number

Location_number identifies the pin number, as defined in the device datasheet or Chapter 11. For MACH-device designs you can place a question mark in this field to define a floating pin.⁸²

Name

Name defines the name of the pin. Each name must be unique and must follow the location_number field.

- Begin the name with an alpha character; use any combination of up to 14 upper- or lowercase alphanumeric characters: A–Z and 0–9.

82 Refer to FLOATING PINS AND NODES, in this chapter, for additional details.

Important: Keep names in a schematic less than or equal to 14 characters. Part and pin names in the schematic may be concatenated when data is converted into PDS format. Any name longer than 14 characters is automatically truncated.

- Use the underscore as a connector and a forward slash to affect polarity; **no** other symbols or punctuation are allowed and no keywords, reserved words, or logic operators are allowed.

Note: The forward slash is not supported for schematic-based designs.

You can use ranges and vector notation to define pin names. You must use the same number of pins as names in ranges and vectors. All pins defined within a range or vector notation have the same storage type and polarity attributes.

Storage

Storage defines the pin storage type.⁸³ You can specify one of the following three storage types.

- COMBINATORIAL or COMB
- REGISTERED or REG
- LATCHED or LAT

Note: You need only enter the first three or four letters of the storage attribute.

If you do not select a pin type, the software defaults to combinatorial, even if the device you selected in the CHIP statement is fully registered. This helps portability of designs across all devices.

⁸³ Refer to Chapter 11, in this section, to determine the correct storage type for the device.

PIN

USE

PIN statements must follow the CHIP statement. Use a separate line for each PIN statement. You do not have to place the PIN statements in numerical order. You can only place comments between PIN statements, not within a PIN statement. Separate each pin attribute by one or more blanks.

Declare only the pins you are using. The software automatically assigns the name NC, no connect, to all pins that are not declared.

You must declare the VCC and GND pins.⁸⁴ You cannot use different names for VCC and GND.

Pin statements in the current version of the software differ from the pin list of previous versions. However, the old syntax is fully compatible with the new syntax.⁸⁵

84 Refer to the individual device datasheet or Chapter 11, in this section, for the correct VCC and GND pin numbers.

85 Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, CHIP, COMBINATORIAL, DECLARATION SEGMENT, LATCHED, NODE, OPERATOR, PAIR, REGISTERED, and VECTOR.

PRELOAD

This simulation keyword loads specified values into the register outputs. Even if a device does not physically support the preload feature, you can simulate the design as though it does, but JEDEC test-vector generation is turned off.

Devices Supported: All registered PLD devices, except PAL22IP6, PAL23S8, PAL16RA10, and PAL20RA10.

SYNTAX

Use this keyword in the simulation segment of Boolean and state-machine designs.

Syntax

```
PRELOAD Prefix_pns
```

Example

```
SIMULATION  
PRELOAD Q0 /Q1 PLAYING
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

Prefix

The prefix specifies the logic state of the corresponding pin, node, or state. Do not leave a blank between Prefix and pns. There are two prefixes: null and forward slash.

- You specify the null prefix to load a logical 1 into the associated register output. In the syntax example, Q0 has a null prefix.

When used in conjunction with a state name, the null prefix loads the specified state. In the syntax example, PLAYING has a null prefix.

- You specify the forward slash to load a logical 0 into the associated register output. In the syntax example, Q0 has a forward-slash prefix.

Pns

You specify the pin, node, or state to be preloaded immediately following the corresponding prefix.

You can list more than one pin or node. You can also use groups and strings.

USE

PRELOAD loads specified logic values into the corresponding device registers. Therefore, if the signal is inverted between the node and pin, the value at the pin will be the inverse of the preloaded value.⁸⁶

Some devices provide a hardware preload feature that is activated by a dedicated pin or product term. Use the SETF command for control of these preload features.

Note: Even if a device does not physically support the preload feature, you can simulate the design as though it does.

PRELOAD places a "P" in the clock field(s) of the JEDEC vector for PAL and PLS devices, as specified in the JEDEC 3A standard. For MACH devices, the buried preload vector "B" is used as defined by the JEDEC 3B standard.

⁸⁶ Refer to the specific device datasheet to determine the correct preload value for the particular flip-flop.

.PRLD

This reserved word defines the conditions in a functional equation when the preload function is activated in devices with logic-controlled preload. When the .PRLD equation is true, you can preload flip-flops with the desired value from the corresponding I/O pin.

Devices Supported

PALCE29M16	PALCE29MA16
------------	-------------

SYNTAX

Use the .PRLD functional equation in the equations segment of Boolean designs.

Syntax

	Node.PRLD	Assignment Operator	Expression
<i>Example</i>			
EQUATIONS			
	Q0	=	/Q0
	GLO.PRLD	=	Q0 * /Q1
	...		

Definitions

All parameters are defined below.

Node.PRLD

Node.PRLD identifies the node associated with the register you want to preload. On the PALCE29M16, this is the global node.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.⁸⁷

Expression

Expression identifies logic you assign to define the conditions when the preload function is activated.

⁸⁷ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .PRLD equation anywhere in the equations segment. The .PRLD equation may fit on more than one product term, depending on the device. The following rules apply.

- Do not use multiple functional equations for the same pin or node. Otherwise, an error is reported during compilation and the processing stops.
- Do not use negative polarity on the left side of the equation. For example, /Q1.PRLD is illegal.

Logic preloads use SETF and the pin polarity defined in the pin declaration segment to determine preload polarity.

<p>Note: Super voltage preloads evaluate a complemented pin, /A, as preload low. If there is no complement, then it is a preload high.</p>

- Disable any product-term enables when .PRLD is low.⁸⁸

In simulation, the SETF command asserts the product term defined by .PRLD to determine the preload values.

In the simulation history file, .PRLD is represented by the letter P, SETF by the letter H, and .RSTF by the letter L.⁸⁹

⁸⁸ Refer to the specific device datasheet for complete information.

⁸⁹ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, PRELOAD, and SETF.

PRLDF

PRLDF assigns a value to a register output to force the specified value at the pin. Use the PRLDF keyword in the simulation segment of Boolean and state-machine designs.

Devices Supported

PAL22IP6	PAL23S8	PAL16RA10	PAL20RA10
----------	---------	-----------	-----------

SYNTAX

Use this keyword in the simulation segment of Boolean and state-machine designs.

Syntax

PRLDF Prefix_rns

Example

```
SIMULATION
PRLDF 01 /02
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

Prefix

The prefix specifies the logic state of the corresponding register, node, or state. Do not leave a blank between Prefix and rns. There are two prefixes: null and forward slash.

- The null prefix indicates the pin value should be a logical 1 if the polarity is not inverted in the pin declaration of the design. In the syntax example, 01 has a null prefix.
- The forward slash indicates that the pin or node should be a logical 0 if the polarity is not inverted in the pin declaration of the design. In the syntax example, 02 has a forward-slash prefix.

Rns

You specify the value to be preloaded immediately following the corresponding prefix.

You can list more than one pin or node. You can also use groups and strings.

USE

PRLDF loads a value into a register so that specified logic values appear at the pin. If an inverter exists between the register output and the pin, PRLDF compensates for the inversion by inverting the register contents.⁹⁰

Some devices provide a hardware preload feature that is activated by a dedicated pin or product term. Use the SETF command to control preload.

For devices with a preload pin, PRLDF disables the outputs, enables preload, loads the registers with the values, disables preload, and then enables the outputs.

Note: Even if a device does not physically support the preload feature, you can simulate the design as though it does.⁹¹

PRLDF places a P in the clock field of the JEDEC vector for devices with supervoltage preloads.

-
- 90 Refer to the individual device datasheet to determine the correct preload value for the particular flip-flop.
- 91 Refer to the following topics, in this chapter, for additional details: CHECK, .PRLD, PRLDF, and SETF.

.R EQUATION

This equation defines when to set the R input high on S-R flip-flops.

Devices Supported

PLS105	PLS167	PLS168	PLS30S16	PALCE610	PAL22IP6
--------	--------	--------	----------	----------	----------

SYNTAX

Use the .R equation in the equations segment of design files with Boolean or state-machine designs.

Syntax

Pn.R	Assignment Operator	Expression
------	---------------------	------------

Example

EQUATIONS

...		
Q1.R	=	IN1 * /IN2
...		

Definitions

All parameters are defined below.

Pn.R

Pn.R identifies the pin or node associated with the S-R flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.⁹²

Expression

Expression identifies the logic you define to determine when the R input in an S-R flip-flop is set high. In the example, when IN1 is true and IN2 is false, the R input in the S-R flip-flop associated with the pin or node Q1 is set high.

⁹² Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .R equation anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin. If you do, the software reports an error during compilation and processing steps.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.R is not allowed.
- You can use the group, string, and vector notation to define signals. This is an excellent way to assign a .R equation to several pins.⁹³

⁹³ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, and .S EQUATION.

REGISTERED

This reserved word defines the output data-storage type on devices with registered outputs. A registered output stores its value regardless of any data changes. New data is placed in the register by an edge-sensitive clock signal. A register may consist of D, T, S-R, and 2-T flip-flops.

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RA8	PAL16RP4
PAL16RP6	PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6
PAL20R8	PAL20RA10	PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8
PAL20X4	PAL20X8	PAL20X10	PAL22IP6	PAL22RX8	PAL22V10
PAL23S8	PAL24R10	PAL24R4	PAL24R8	PAL26V12	PALCE29M16
PALCE29MA16	PAL32R16	PAL32VX10	PAL64R32	PALCE610	PLS105
PLS167	PLS168	PLS30S16	MACH 1	MACH 2	

SYNTAX

You include the optional reserved word in the PIN or NODE statement of Boolean and state-machine designs.

Syntax

PIN or NODE	Number	Location_name	Storage
----------------	--------	---------------	---------

Example

```
...
CHIP COUNTER PAL16R8
PIN      15      OUT1      REGISTERED
PIN      16      OUT2      REG
...
```

Definitions

Only the reserved word is discussed below.

Storage

The storage value defines the pin or node storage type. If you do not specify a storage type, combinatorial is the default.

REGISTERED

- Place the reserved word REGISTERED after the pin or node name in the corresponding statement.
- Use either the complete word REGISTERED or the three-letter abbreviation, REG.

USE

There are two ways to specify the storage type.

- Use the declaration segment form: you select the storage field, display the option list, and select an option.
- Type the storage value in the appropriate PIN or NODE statement in the PDS file using a text editor.

PALASM requires a data-storage type for each I/O or output pin: COMBINATORIAL, LATCHED, or REGISTERED. The software requires the registered pin or node type even if the pin or node can only be registered.⁹⁴

⁹⁴ Refer to the following topics, in this chapter, for additional details: COMBINATORIAL, DECLARATION SEGMENT, LATCHED, NODE, and PIN.

REVISION

This keyword begins the statement that defines the revision of the current design. The design revision is useful for documentation purposes.

Devices Supported: All PLD devices.
--

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

REVISION	Design_revision
----------	-----------------

Example

TITLE	
PATTERN	
REVISION	2.2B
AUTHOR	
COMPANY	
DATE	
CHIP	

Definitions

Only the descriptor following the keyword REVISION is discussed.

Design_revision

Design_revision is optional and may be any combination of up to 59 alphanumeric characters that designates the current version of the design.

- You can use other symbols or punctuation; however, you cannot use the dollar sign.
- You can use reserved words in this statement.

USE

The following error conditions pertain to the REVISION statement.

- Without the REVISION statement, the software issues a warning and continues processing the file.
- With multiple REVISION statements, the software issues an error and stops processing the file.⁹⁵

⁹⁵ Refer to the following topics, in this chapter, for additional details: AUTHOR, COMPANY, DATE, PATTERN, and TITLE.

.RSTF

This reserved word defines when to assert a reset signal high on devices having flip-flops or registers with a reset input.

Devices Supported

PAL10H20EG8	PAL10H20EV8	PAL16RA8	PAL20RA10	PAL22IP6	PAL22RX8
PAL22V10	PAL23S8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state-machine designs.

Syntax

Pn.RSTF	Assignment Operator	Expression
---------	---------------------	------------

Example

EQUATIONS

Q0	=	/Q1 */Q2
Q0.RSTF	=	RST * /SET
...		

Definitions

All parameters are defined below.

Assignment Operator

The assignment operator is a symbol that defines a specific operation, as interpreted by the software when processing design files.⁹⁶

Pn.RSTF

Pn.RSTF identifies the pin or node associated with the flip-flop having a reset input.

Expression

Expression defines the logic conditions that determine when to assert the reset signal high. In the previous example, the reset pulse is initiated when RST is true and SET is false.

⁹⁶ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

Multiple .RSTF statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting. You can specify a global reset on devices with global nodes, for example, global node.RSTF.

You cannot use negative polarity on the left side of an equation. For example, /Q1.RSTF is not allowed.

You can use the GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a functional equation to several pins and nodes.

Depending on the device, the reset line is synchronous or asynchronous. On the PAL16RA8, if both preset and reset of a flip-flop are high, the flip-flop is bypassed.

In the simulation history file, .RSTF is represented by the letter L, SETF by the letter H, and .PRLD by the letter P.⁹⁷

⁹⁷ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, PRELOAD, and SETF.

.S EQUATION

This equation defines when to set the S input on S-R flip-flops high.

Devices Supported

PLS105	PLS167	PLS168	PLS3016S	PALCE610	PAL22IP6
--------	--------	--------	----------	----------	----------

SYNTAX

Use the .S equation in the equations segment of Boolean or state-machine designs.

Syntax

Pn.S	Assignment Operator	Expression
------	---------------------	------------

Example

EQUATIONS

...		
Q1.S	=	IN1 * /IN2
...		

Definitions

All parameters are defined below.

Pn.S

Pn.S is the pin or node associated with the S-R flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.⁹⁸

Expression

Expression identifies the logic you define to determine when the S input in an S-R flip-flop is set high. In the syntax example, when IN1 is true and IN2 is false, the S input in the S-R flip-flop associated with the pin or node Q1 is set high.

⁹⁸ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .S EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.S is not allowed.
- You can use GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .S equation to several pins.⁹⁹

⁹⁹ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

.SETF

This reserved word defines when to assert a preset signal high on devices having flip-flops or registers with a preset input.

Devices Supported

PAL10H20EG8	PAL10H20EV8	PAL16RA8	PAL20RA10	PAL22IP6	PAL22RX8
PAL22V10	PAL23S8	PAL26V12	PALCE29M16	PALCE29MA16	
PAL32VX10	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state-machine designs.

Syntax

Pn.SETF	Assignment Operator	Expression
---------	---------------------	------------

Example

EQUATIONS

Q0	=	/Q1 */Q2
Q0.SETF	=	/RST * SET

...

Definitions

All parameters are defined below.

Pn.SETF

Pn.SETF identifies the pin or node associated with the flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.¹⁰⁰

Expression

Expression defines the logic conditions that determine when to assert the preset signal high on flip-flops or registers that support preset inputs.

¹⁰⁰ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

Multiple .SETF statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting. You can specify a global preset on devices with global nodes, for example, global node.SETF.

You cannot use negative polarity on the left side of the equation. For example, /Q1.SETF is not allowed.

You can use the GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a functional equation to several pins and nodes.

Depending on the device, the preset line is synchronous or asynchronous.¹⁰¹

If you define the outputs as COMBINATORIAL, the default value for preset is VCC, unconditional high. If you define the outputs as registered, the default value is GND, unconditional low.

In the simulation history file, .SETF is represented by the letter H, RSTF by the letter L, and .PRLD by the letter P.¹⁰²

¹⁰¹ Refer to the Chapter 11, in this section, for details on using .SETF with a specific device.

¹⁰² Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, GROUP, .RSTF, STRING, and VECTOR.

SETF

SETF assigns values to specific inputs during simulation.

Devices Supported: All PLD devices.
--

SYNTAX

Include the SETF command in the simulation segment or auxiliary simulation file for Boolean and state-machine designs.

Syntax

```
SETF Prefix_pn
```

Example

```
SIMULATION  
SETF IN1 /OE
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

If the signal being set is defined with the same polarity as in the PIN or NODE declaration segment, the signal is set to a logical 1. If the polarity is reversed, the signal is set to a logical 0.

Note: The following examples are valid only when the signals are defined as active-high in the PIN or NODE declaration segment.
--

Prefix

The prefix specifies the logic state of the associated input pin or node. There are two prefixes: null and forward slash.

- The null prefix sets the corresponding input to a logical 1. In the syntax example, IN1 has a null prefix.
- The forward slash sets the corresponding input to a logical 0. In the syntax example, OE has a forward-slash prefix.

Pn Pn is the input pin or node to be set.

USE You can list more than one pin or node. Separate multiple pins or nodes with a blank. You can also use groups and strings.

If the signal being set is defined with the same polarity as in the PIN or NODE declaration segment, the signal is set to a logical 1. If the polarity is reversed, the signal is set to a logical 0.

You cannot use the SETF command to set states. However, you can use the SETF command to set input values.

The software shows each occurrence of SETF by placing the letter g in the header of the waveform and text simulation files.

SIGNATURE

This keyword allows you to program user-defined data into devices having a SIGNATURE word function. This data can be used for such purposes as user identification, revision control, or inventory control.

Devices Supported	
PALCE16V8	PALCE20V8

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

SIGNATURE	Assignment Operator	Number or String
-----------	---------------------	------------------

Example

```
DECLARATION
...
PIN 18 /OUT8 REG
SIGNATURE = V2_5/89
...
```

Definitions

Only the parameters Number and String are discussed below.

Number or String

Number or string is either a base (radix) number or an alphanumeric character string.

SIGNATURE supports four number radices; the default is decimal.

- Binary #B or #b
- Decimal #D or #d
- Hexadecimal #H or #h
- Octal #O or #o

The software allows a maximum of 64 bits for the radix number. This translates to the following list of maximum digits allowed for each radix.

SIGNATURE

RADIX	MAXIMUM NUMBER OF DIGITS
Binary	64
Hex	16
Octal	21
Decimal	15

If you exceed the maximum number of digits allowed for a radix, the software issues a warning and truncates the extra most significant bits.

If a number contains a blank, non-number, a decimal number, or any other alphanumeric character except the radix operator, the software treats the entire character string as alphanumeric.

In using alphanumeric characters, observe the following guidelines.

- You can use any combination of alphanumeric characters up to a maximum of eight characters.

If the number exceeds eight, the software issues a warning and truncates the extra characters to the right.

- You can use underscores and blanks.

The software converts alphanumeric characters to ASCII and all lowercase characters to uppercase.

USE

Place the SIGNATURE statement after the PIN or NODE statements. Observe the following rules.

- You can place the SIGNATURE statement in any order with the GROUP or STRING statements.
- You can use only one SIGNATURE statement for each device.

SIGNATURE

- You must use the assignment operator, =, in the statement.

If you have multiple SIGNATURE statements, the software issues a warning and programs the last SIGNATURE statement.

You can access signature information even if the security fuse has been programmed.¹⁰³

¹⁰³ Refer to DECLARATION SEGMENT and OPERATOR, in this chapter, for additional details.

SIMULATION

Use the SIMULATION keyword to start the simulation segment .

Devices Supported: All PLD devices.
--

SYNTAX

Syntax

SIMULATION

Example

SIMULATION

Definitions

No parameters are required with this keyword.

USE

Use the SIMULATION keyword to start the simulation segment or auxiliary simulation file of Boolean and state-machine designs.

START_UP

This keyword allows you to power up in a specific state or asynchronously branch to a state whenever an initialization condition occurs.

Devices Supported: All devices that support state-machine descriptions.

SYNTAX

The START_UP keyword is used in Moore machines. START_UP.OUTF is used in registered Mealy machines.

Syntax

```
START_UP      :=      POWER_UP -> State1
                  + Condition1 -> State2

START_UP.OUTF :=      POWER_UP -> Outputs
                  + Condition1 -> Outputs
```

Example

```
STATE
    ;State Setup and Defaults
    ...
```

Moore machines

```
START_UP      := POWER_UP -> S1
                  + INIT -> S1
    ...
;Powers up and initializes to S1
```

Mealy machines

```
START_UP.OUTF :=      01 * 02
                  + INIT -> 01*02

;State Assignments
S1 = /STATE BIT1 * /STATE BIT2
;S1 value is 00
...
;State Equations
S1 = FC -> S3
    + FCC -> S7
...

```

Definitions

Parameters following the keyword are defined below. Additional details are discussed under Use.

State1

When power is applied to the device, the device goes to state1.

- In devices that initialize with all flip-flops high or all flip-flops low, the START_UP command assigns the appropriate all-high or all-low state-bit code to the specified state.
- In devices with programmable power up, the START_UP command programs the device to power up in the specified state. If you specify a particular state-bit code using the manual state-bit assignment syntax, the software programs the flip-flops to initialize with the specified values.

Condition1

Condition1 is a user-defined condition that specifies when an initialization occurs. When the condition is true, the device is initialized asynchronously to state2.

A condition must be defined in the condition section of the state-machine design. If the condition consists of a single input, the input name can be used in place of the condition name.

State2

This state occurs as a result of the initialization condition. This state may differ from the power-up state.

USE

Use initialization routines to ensure the state machine powers up in a known state or branches to a known state whenever initialization occurs.

If you do not include a START_UP statement, the device will power up in the state that appears in the first transition equation in the PDS file.

START_UP

The first line of the example contains the power-up state. When power is applied to the device, it goes to this state. It may be helpful to think of power up as a "cold boot."

The second line of the START_UP statement defines the initialization state. Only devices with programmable initialization support this function of the START_UP statement.¹⁰⁴

STARTUP.OUTF allows you to define the outputs at power-up and initialization. This is especially useful for synchronous Mealy machines where outputs are delayed by one cycle from their respective states.¹⁰⁵

You cannot use a default branch in the START_UP statement.

¹⁰⁴ Refer to Chapter 11, in this section, for information on specific devices.

¹⁰⁵ Refer to .OUTF and STATE EQUATIONS, in this chapter, for additional details.

STATE

Use the STATE keyword to identify the state segment of state-machine designs. The state segment contains setups, defaults, and state equations.

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R10
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Use the state keyword in the PDS file after the declaration segment. If your design contains a mix of state-machine and Boolean equations, it can appear before or after the Boolean segment.

Syntax

STATE

State Setup and Defaults

State Equations

 Transition Equations

 Output Equations

 State Assignment Equations

 Condition Equations

Example

```
STATE
;State Setup and Defaults
MOORE MACHINE
START_UP := POWER_UP -> S1
DEFAULT_BRANCH HOLD_STATE
;State Transition Equations
S1 := COND1-> S3
      + COND2 -> S7
...
;State Output Equations
S1.OUTF = OUT1*/OUT2
...
;State Assignment Equations
S1 = /STATE BIT1 * /STATE BIT2
...
;State Condition Equations
CONDITIONS
COND1 = IN1 * /IN2 * IN3
...
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

Setup and Defaults

Statements at the beginning of the state segment identify the state machine. CLKF, MASTER_RESET, MEALY_MACHINE, MOORE_MACHINE, START_UP, and START_UP.OUTF appear in this section.

STATE

State-machine designs can have global and local defaults. Global defaults are defined by `DEFAULT_BRANCH`, `DEFAULT_OUTPUT`, and `OUTPUT_HOLD`. Local defaults are defined in the state equations with the local default operator, `+>`.

State Equations

There are four types of state-machine equations. They have the following functions.

- Transition equations (required)

For each state, these conditions specify what the next state will be under various conditions. See Condition equations below.

- Output equations (optional)

These equations specify the outputs of the state machine. No output equations are required in cases where the state bits themselves are the outputs.

- State assignment equations (optional)

These equations specify the bit code to be assigned to each state name used in the design. If these equations are omitted, the software assigns the bit codes automatically.

- Condition equations (normally required)

These equations specify a condition name for each set of input values used to determine a transition. You can use input names directly only if a single input controls the transition; otherwise, you must use condition names.

USE

The state segment follows the declaration segment of the PDS file. If the design has an equations segment, the state segment can precede or follow it.

Important: The state segment typically replaces the equations segment. It is possible to modify state equations with Boolean equations by including both equation and state segments, in any order. In this case, you must select the Merge Mixed Mode option from the Compile Setup menu.

The STATE segment supports the following reserved words.

- CLKF
- CONDITIONS
- DEFAULT_BRANCH
- DEFAULT_OUTPUT
- HOLD_STATE
- MASTER_RESET
- MEALY_MACHINE
- MOORE_MACHINE
- NEXT_STATE,
- .OUTF
- OUTPUT_ENABLE
- OUTPUT_HOLD
- START_UP
- STATE

STATE ASSIGNMENT EQUATION

State assignment equations define the unique bit codes to be assigned to each state name used in the design. The bit codes are composed of state bits that are stored in flip-flops.

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Use state assignment equations in the state segment of state-machine designs.

Syntax

State name	Assignment Operator	State Bits
------------	---------------------	------------

Example

STATE

```
    ;State Assignments
S1      =          /STATE BIT1 * /STATE BIT2
S2      =          /STATE BIT1 * STATE BIT2
S3      = S        STATE BIT1 * /STATE BIT2
S4      =          STATE BIT1 * STATE BIT2
...
    ;State Equations
S1 := FC -> S3
    + FCC -> S7
...
```

Definitions

Parameters following the keyword are defined below. Additional details are provided under Use.

State Name

The user-defined state name must be unique and can have up to 14 alphanumeric characters. It cannot contain operators or reserved words.

STATE ASSIGNMENT EQUATION

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.¹⁰⁶

Statebits

The state bit name composes the bit code. The state bit name is register name, as defined in the PIN or NODE statements. Use an asterisk, *, to separate state bits. Use polarity notation to indicate the value of each state bit.

USE

State assignments follow state defaults and precede condition equations. Use the assignment operator, =, to define state assignments.

Each state assignment must include the complete set of state bits.¹⁰⁷ If you use three state bits for eight states, each assignment must include all three bits.

The example uses two state bits, State Bit1 and State Bit2. When both are low, the device is in state S1. When State Bit1 is low and State Bit2 is high, the device is in state S2. When State Bit1 is high and State Bit2 is low, the device is in state S3; when both are high, the device is in state S4.

For large designs in which you use many state bits (six or more), you may not want to list all possible state bit combinations. However, not defining all possible state bit combinations leaves some undefined or illegal states.

¹⁰⁶ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

¹⁰⁷ Refer to Section II, Chapter 4, for information on choosing state bit assignments.

STATE ASSIGNMENT EQUATION

The present state is defined by the contents of the state register, which consists of n bits capable of defining 2^n possible states. Before you can determine the present state, you must know the contents of all n bits. For example, three-state register bits define up to 2^3 , or eight, possible states.

To define a state machine with 2^n states, you need a device with n registered outputs or buried registers (nodes) to use as state register bits. For example, the PAL16R8 has eight registered outputs and accommodates up to 2^8 , or 256, states, provided you do not use any of the registers for other purposes such as independent outputs.¹⁰⁸

If you do not assign state bits, the software assigns them automatically. The software assigns state bits to outputs that aren't defined by PIN or NODE statements. Look at the Execution log file to determine the automatic state bit assignments made by the software.¹⁰⁹

108 Refer to STATE and STATE EQUATIONS, in this chapter, for additional details.

109 Refer to Chapter 9, in this section, for more information on execution log file.

STATE EQUATIONS

STATE EQUATIONS control the transitions, outputs, state assignment and conditions of state machines.

Devices Supported					
PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PAL64R32	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

Use these equations in the state segment of the design to define the behavior of the state machine.

Syntax

State Equations

- Transition Equations
- Output Equations
- State Assignment Equations
- Condition Equations

Example

```
STATE
    ;State Setup and Defaults
    MOORE MACHINE
    START_UP := POWER_UP -> S1
    DEFAULT_BRANCH HOLD_STATE
    ;State Transition Equations
    S1 := COND1-> S3
        + COND2 -> S7
    ...
    ;State Output Equations
    S1.OUTF = OUT1*/OUT2
    ...
    ;State Assignment Equations
    S1 = /STATE BIT1 * /STATE BIT2
    ...
    ;State Condition Equations
    CONDITIONS
    COND1 = IN1 * /IN2 * IN3
    ...
```

STATE EQUATIONS

Definitions

Parameters following the keywords STATE EQUATIONS are defined below. Additional details are discussed under Use.

Transition Equations

These equations specify, for each state, what the next state will be under various conditions.

Output Equations

Output equations specify the output of the state machine. For a Moore machine, they define the outputs for a given state. For a Mealy machine, they define the outputs for a given state and input condition.

State Assignment Equations

These assignments define states as unique combinations of register bits.

STATE EQUATIONS

Condition Equations

Condition equations specify a condition name for each set of input values used to determine a transition. You can use input names directly only if a single input controls the transition; otherwise, you must use condition names.

USE

The order of state equations is not important except for condition equations which have their own section under the **CONDITIONS** keyword. The conditions segment must terminate the state segment.

Setups and defaults are not equations and must appear at the beginning of the state segment.¹¹⁰

¹¹⁰ Refer to the following topics, in this chapter, for additional details: **CONDITIONS**, **.OUTF**, **STATE**, **STATE ASSIGNMENT EQUATION**, and **STATE TRANSITION EQUATION**.

STATE OUTPUT EQUATION

STATE OUTPUT EQUATIONS control state-machine outputs.¹¹¹

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS1	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PALCE610	PLS105	PLS167	PLS168	PLS30S16	MACH 1
MACH 2					

SYNTAX

Include output equations in the state segment of state-machine designs after setup and defaults and before the condition equations.

Syntax

Moore machines	Statename.OUTF	=	Output expression
Mealy machines	Statename.OUTF	=	Condition 1 -> Output 1 + Condition 2 -> Output 2 ... + Condition n -> Output n +> Local default

Example

Moore machines	... TWO.OUTF	=	/CNT2 * CNT1 * /CNT0
Mealy machines	TWO.OUTF	=	RUN_UP -> /CNT2 * CNT1 * /CNT0 ... TEST -> CNT2 * CNT1 * CNT0 +> /CNT2 * /CNT1 * /CNT0 ...

Definitions

The construct immediately preceding, and all constructs following, the keyword are defined below. Additional details are provided under Use.

¹¹¹ Refer to .OUTF, in this chapter, for a description of state output equations.

STATE OUTPUT EQUATION

Statename

Statename identifies the name of the state as specified in the state assignments or state transition equations. It must be unique and can have up to 14 alphanumeric characters.

Outputs

Outputs are pin names with appropriate logic sense to create the desired logic values. Outputs are separated by an asterisk, *. In the syntax example, when the Moore machine is in state TWO, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

When the Mealy machine is in state TWO and the inputs match the condition defined as RUN_UP, the output bits CNT2, CNT1 and CNT0 will be 0, 1, and 0, respectively.

You specify the output values regardless of pin polarity. The software adjusts polarity as necessary.

Conditions

In a Mealy machine, the outputs depend on the current state and the current input conditions. This entry specifies the condition under which the specified output will occur. The condition names must be defined in the conditions section of the state-machine design.

If the condition consists of a single input, the input name may be used in place of the condition name. You can use VCC to specify an unconditional output.

Moore machine outputs do not have conditions since their outputs are determined only by the present state.

Local Default

This output is generated if none of the conditions is satisfied. Local defaults are valid only for Mealy machines. Local defaults override global defaults.

STATE OUTPUT EQUATION

USE

You can place output equations anywhere within the state segment. You may prefer to have all of the output equations after all of the state equations or following each state equation with its corresponding output equation.

You can use the following operators in STATE OUTPUT EQUATIONS.

OPERATOR	DEFINITION
->	Conditional output for Mealy machines
+->	Local default for a Mealy machine
=	Combinatorial assignment operator
:=	Registered assignment operator

For Mealy machines, conditions in .OUTF don't have to match conditions in the state-transition equations. Typically, however, these conditions match.

You can use the state bits as outputs by making the output pins the same as the state bits and performing manual state bit assignment.¹¹² In this case, you can omit the output equations. If you do this, don't use the following constructs.

- DEFAULT_OUTPUT
- OUTPUT_HOLD

You can define some outputs with state bits and some outputs with output equations.

If you don't use the state bits as outputs, you must specify output equations. Default output specifications are optional.

¹¹² Refer to Section II, Chapter 4, for additional details regarding assigning state bits.

STATE OUTPUT EQUATION

Registered Mealy machine outputs are valid one clock cycle after reaching the new state. Combinatorial Mealy and Moore machine outputs and registered Moore machine outputs are valid on reaching the new state. Undefined output pins have a don't-care value.¹¹³

¹¹³ Refer to the following topics, in this chapter, for additional details: `CONDITIONS`, `DEFAULT_BRANCH`, `DEFAULT_OUTPUT`, `LOCAL_DEFAULT`, `MEALY_MACHINE`, `MOORE_MACHINE`, `OPERATOR`, `OUTPUT_HOLD`, `STATE`, `STATE ASSIGNMENT EQUATION`, `STATE EQUATIONS`, and `STATE TRANSITION EQUATION`.

STATE TRANSITION EQUATION

For each state, transition equations specify what the next state will be under various conditions.

Devices Supported

PAL10H20EV8	PAL16R4	PAL16R6	PAL16R8	PAL16RP4	PAL16RP6
PAL16RP8	PALCE16V8	PAL18U8	PAL20R4	PAL20R6	PAL20R8
PAL20RS4	PAL20RS8	PAL20RS10	PALCE20V8	PAL20X4	PAL20X8
PAL20X10	PAL22RX8	PAL22V10	PAL23S8	PAL24R10	PAL24R4
PAL24R8	PAL26V12	PALCE29M16	PALCE29MA16	PAL32R16	PAL32VX10
PAL64R32	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

Include state-transition equations in the state segment of state-machine designs.

Syntax

STATE

```
...
Present state      :=      Condition1 -> State1
                    + Condition2 -> State2
                    +-> Local default
```

Example

STATE

```
...
;State Equations
...
TWO                :=      COUNT_UP -> THREE
                    + COUNT_DWN -> ONE
                    +-> TWO
...

```

Definitions

Parameters following the keyword STATE are defined below. Additional details are discussed under Use.

Present State

The present state name is defined in the state diagram.

- Use any combination of up to 14 upper- or lowercase alphanumeric characters, A-Z and 0-9.

STATE TRANSITION EQUATION

- Do not use keywords, reserved words, or logic operators.

Condition

Condition identifies condition names as defined in the conditions section of the state-machine design.

State

State identifies the next state, as defined in the state diagram.

Local Default

Local default defines the state the machine will go to if none of the specified conditions is satisfied.

USE

You can place state transition equations anywhere within the state segment except in the CONDITIONS section.

When you create transition equations, use the state-equation operator, `:=`, to separate the present state from the transition. Use the transition operator, `->`, to identify the condition and corresponding transition state. Use the OR operator (`+`) to indicate additional transitions. Use the default operator, `+>`, to identify the local default.¹¹⁴

In the example, when the state machine is in state TWO, it will transition to state THREE if the input conditions specified for COUNT_UP are satisfied. It will transition to state ONE if the conditions for COUNT_DWN are satisfied. If neither of these conditions is satisfied, it will remain in state TWO.

¹¹⁴ Refer to the following topics, in this chapter, for additional details: CONDITIONS, DEFAULT_BRANCH, DEFAULT_OUTPUT, LOCAL_DEFAULT, STATE, STATE_ASSIGNMENT EQUATION, and STATE EQUATIONS.

STRING

This keyword allows you to assign a name to frequently used character strings such as equations and expressions. You can then use the string name anywhere in the remainder of your design file. The software substitutes the character string for the name during processing.

Devices Supported: All PLD devices.
--

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

STRING	String Name	'String'
--------	-------------	----------

Example

```
...
STRING      IN1      'A1 + /A2 + A3'
STRING      IN2      '(A1 + /A2 + /A3)'
```

...

EQUATIONS

```
...
BANK1 = IN1
O5 = /IN1
O6 = /IN2
...
```

Definitions

Parameters following the keyword STRING are defined below.

String Name

This is the name assigned to a cluster of equations, expressions or other parameters. The name can then be used in the equations or state segments of a design to refer to the entire cluster, without having to list all the characters separately. Follow the rules below.

- Assign a unique name of up to 14 alphanumeric characters.
- Do not use keywords, operators, or reserved words.

- Place the name after the keyword **STRING** and before the **PIN** or **NODE** statements.

String

String identifies the cluster of characters defined by the string name. Single quotes are delimiters and identify the characters to be substituted. The software substitutes these characters literally. However, functional strings, must conform to the rules of the function, such as in an expression. The software does not limit the number of characters you can substitute.

USE

Place at least one blank between the keyword **STRING**, the string name, and the cluster of characters comprising the string. Extra blanks or tabs are reduced to one blank.

In the syntax example, the software processes the string substitution as follows:

```
BANK1 = A1 + /A2 + A3
05 = /(A1 + /A2 +A3)
06 = /(A1 + /A2 +/A3)
```

To DeMorganize an expression when you complement a string name. Use parentheses to enclose the expression. This is only true for expressions. For example, you cannot complement an **.OUTF** statement.

The following table illustrates the effect of complementing string names.

STRING	EXPRESSION
IN1	A1 + /A2 + A3
/IN1	/A1 + /A2 + A3
IN2	(A1 + /A2 + /A3)
/IN2	/A1 + A2 + A3

STRING

Complementing string IN1 causes only the first pin in the string to invert polarity. In contrast, complementing string IN2 DeMorganizes the entire string.

The keywords GROUP and STRING have distinctly different uses. Use GROUP only for clustering pins. Use STRING to substitute any string of characters.¹¹⁵

¹¹⁵ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, DECLARATION SEGMENT, and GROUP.

TEST

This keyword verifies that values at the Q outputs of **registers** are equal to expected values, and creates "T" test vectors per the JEDEC 3B standard.

The Test command changes the simulation results to match the specified signal values, and generates corresponding test vectors in the JEDEC file.¹¹⁶

Devices Supported: MACH-device designs only.

If you use the TEST command with non-MACH PLDs, it is converted to a CHECKQ command automatically.

SYNTAX

You use the TEST command in either the simulation segment of a PDS file or in an auxiliary simulation file for Boolean, state-machine, or schematic-based designs.

Syntax

```
TEST      Prefix_rns
```

Example

```
SIMULATION
TEST      /Q1 Q2
```

Definitions

Because the TEST command verifies signal values at the Q output of registers, you do not need to account for active-low pin declarations. This makes TEST especially useful for verifying states.

Parameters following the command TEST are defined below. Additional details are provided under Use.

¹¹⁶ Refer to the JEDEC *JESD3-B Standard* for additional details regarding test-vector generation.

Prefix

The prefix indicates the logic state of the corresponding register, node, or state. Do not leave a blank between Prefix and rns. There are two prefixes: null and forward slash.

- The null prefix indicates the register or node should be a logical 1. In the syntax example, Q0 has a null prefix.

When used in conjunction with a state name, a null prefix indicates the specified state should be checked. In the syntax example, PLAYING has a null prefix.

- The forward slash indicates that the signal should be a logical 0. In the syntax example, Q1 has a forward-slash prefix.

Note: If the simulated value does not match the expected value, the TEST command forces the expected value. The expected value appears in the test vectors, and a clash is indicated in the simulation results.

Rns

Rns defines the names of the output registers, nodes, or states to be verified. Each value represents both the signal name or state and the expected output value.

- Each signal name can be up to 14 characters in length.
- Include up to 76 characters per line and use as many lines as you need.

The screen displays up to 76 characters per line; however, all information is processed properly even if it extends beyond the 76th character.

TEST

- Include a blank between the keyword and the first register, node, or state in the list.

You can also include **multiple** register and node names. You can also use strings or vector notation to define the signal list.

- Separate multiple prefixed register and node names with a blank.

USE

A conflict occurs when the value of the output register does not match the value defined in the TEST command. Each conflict is identified with a question mark in the simulation output files; a warning is issued and the expected value is reported in the execution log file.

.T EQUATION

This equation defines when to set the T input on T-type flip-flops high.

Devices Supported: PALCE610.

SYNTAX

Use the .T EQUATION in the equations segment of Boolean or state-machine designs.

Syntax

Pn.T	Assignment operator	Expression
------	---------------------	------------

Example

EQUATIONS

...		
Q1.T	=	IN1 * /IN2
...		

Definitions

All parameters are defined below.

Pn.T

Pn.T is the pin or node associated with the T flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.¹¹⁷

Expression

Expression identifies the logic defining when the input on .T-type flip-flops is set high. In the syntax example, when IN1 is true and IN2 is false, the flip-flop associated with the pin or node Q1 is set high.

¹¹⁷ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .T EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin. If you do, the software reports an error during compilation and processing stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.T is not allowed.
- You can use the GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .T EQUATION to several pins.¹¹⁸

¹¹⁸ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, and .S EQUATION.

.T1 EQUATION

This equation defines when to set the T1 input on dual toggle, 2-T, flip-flops high.

Devices Supported: PAL22IP6.

SYNTAX

You use the .T1 equation in the equations segment of Boolean or state-machine designs.

Syntax

Pn.T1	Assignment Operator	Expression
-------	---------------------	------------

Example

EQUATIONS

...		
Q1.T1	=	IN1 * /IN2
...		

Definitions

All parameters are defined below.

Pn.T1

Pn.T1 is the pin or node associated with the dual toggle flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.¹¹⁹

Expression

Expression identifies the logic you define to determine when to set the T1 input on dual toggle flip-flops high. In the example, when IN1 is true and IN2 is false, the T1 input in the dual toggle flip-flop associated with the pin or node Q1 is set high.

¹¹⁹ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .T1 EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.T1 is not allowed.
- You can use GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .T1 equation to several pins.¹²⁰

¹²⁰ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

.T2 EQUATION

This equation defines when to set the T2 input on dual toggle, 2-T, flip-flops high.

Devices Supported: PAL22IP6.

SYNTAX

You use the .T2 EQUATION in the equations segment of Boolean or state-machine designs.

Syntax

	Pn.T2	Assignment Operator	Expression
<i>Example</i>			
	EQUATIONS		
	...		
	Q1.T2	=	IN1 * /IN2
	...		

Definitions

All parameters are defined below.

Pn.T2

Pn.T2 is the pin or node associated with the dual toggle flip-flop. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation as interpreted by the software when processing design files.¹²¹

Expression

Expression identifies the logic you define to determine when to set the T2 input on dual toggle flip-flops high. In the syntax example, when IN1 is true and IN2 is false, the T2 input in the dual toggle flip-flop associated with the pin or node Q1 is set high.

¹²¹ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

USE

You can place the .T2 EQUATION anywhere in the equations segment. Observe the following rules.

- You cannot have multiple equations for the same pin or node. If you do, the software reports an error during compilation and the process stops.
- You cannot use negative polarity on the left side of the equation. For example, /Q1.T2 is not allowed.
- You can use GROUP, STRING, and VECTOR notation to define signals. This is an excellent way to assign a .T2 equation to several pins.¹²²

¹²² Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, GROUP, STRING, and VECTOR.

TITLE

This keyword begins the statement that defines the title of the design. Including the design title is useful for documentation purposes.

Devices Supported: All PLD devices.
--

SYNTAX

You use this keyword in the declaration segment of Boolean and state-machine designs.

Syntax

TITLE	Design Title
-------	--------------

Example

TITLE	Traffic Controller
PATTERN	
REVISION	
AUTHOR	
COMPANY	
DATE	
CHIP	

Definitions

Only the descriptor following the keyword TITLE is discussed.

Design Title

Design title is an optional name that includes any combination of up to 59 alphanumeric characters indicating the company's name.

- You can use other symbols or punctuation; however you cannot use the dollar sign.
- You can use reserved words in this statement.

USE

There are two ways to enter the design title.

- Use the declaration segment form, select the TITLE field, and type the name.
- Type the design title first, followed by the pattern in the PDS file using a text editor.

The following error conditions pertain to the TITLE statement.

- Without a design TITLE statement, a warning is issued and processing continues.
- With multiple design TITLE statements, an error is reported and processing stops.¹²³

¹²³ Refer to the following topics, in this chapter, for additional details: AUTHOR, COMPANY, DATE, DECLARATION SEGMENT, PATTERN, and REVISION.

TRACE_OFF

TRACE_OFF defines the end of the simulation section being traced by TRACE_ON.

Devices Supported: All PLD devices.
--

SYNTAX

Use the reserved word in the simulation segment or auxiliary simulation file of Boolean and state-machine designs.

Syntax

```
TRACE_OFF
```

Example

```
SIMULATION  
TRACE_ON  
TRACE_OFF
```

Definitions

No parameters are required with this keyword.

TRACE_OFF

This reserved word indicates when to conclude the simulation section being traced by TRACE_ON.

USE

If you do not conclude a trace section with TRACE_OFF, the software terminates the trace at the end of the file and displays a warning. If you have multiple traces but do not conclude one, the software does so by assuming a TRACE_OFF before the next TRACE_ON statement.¹²⁴

¹²⁴ Refer to SIMULATION and TRACE_ON, in this chapter, for additional details.

TRACE_ON

TRACE_ON defines which signal values to record in the trace file during simulation. Tracing allows you to reorder and group signals however you wish. Tracing also resolves reverse polarity problems because you can define polarity in the TRACE statement.

Devices Supported: All PLD devices.
--

SYNTAX

Use the keyword in an auxiliary simulation file or the simulation segment of Boolean and state-machine designs.

Syntax

```
TRACE_ON Pn
```

Example

```
SIMULATION  
TRACE_ON 01 02 /IN2 /IN3
```

Definitions

Only the parameter following the keyword TRACE_ON is defined below.

Pin, Node

Pin, node is a list of pin or node names, as defined in the PIN and NODE statements of the declaration segment.

USE

Observe the following usage conventions when using TRACE_ON.

- You can repeat TRACE_ON statements.
- You cannot nest TRACE_ON statements. If the software finds a new TRACE_ON statement, it abandons the previous statement and continues with the new statement.
- You can list multiple pins or nodes with TRACE_ON. Separate pins and nodes with only a blank.

TRACE_ON

- You can reverse pin or node polarity by placing or removing a forward slash before the pin or node name. You cannot use polarity with states.
- You can use strings and groups with TRACE_ON.
- Conclude a trace section with a TRACE_OFF command. If you do not conclude a trace with TRACE_OFF, the software terminates the trace at the end of the file and displays a warning.

During simulation, the software generates a simulation history file. You view this file as either an ASCII table or as a history waveform.

Tracing causes the software to create the trace file, which contains the simulation results of the pins and nodes selected by TRACE_ON. The signals are listed in the same order and with the same polarity as listed in the TRACE_ON statement. If TRACE_ON is not used, then no trace file is created.

The software converts state names to state bits and appends them to the TRACE_ON statement for simulation.¹²⁵

¹²⁵ Refer to SIMULATION and TRACE_ON, in this chapter, for additional details.

.TRST

This reserved word defines when to enable three-state outputs on devices with programmable enable.

Devices Supported

PAL10H20EG8	PAL10H20EV8	PAL16L8	PAL16P8	PAL16R4	PAL16R6
PAL16R8	PAL16RA8	PAL16RP4	PAL16RP6	PALCE16V8	PAL18P8
PALC18U8	PAL20L10	PAL20L8	PAL20R4	PAL20R6	PAL20RA10
PAL20RS4	PAL20RS8	PAL20S10	PAL20X4	PAL20X8	PALCE20V8
PAL22IP6	PAL22RX8	PAL22V10	PAL23S8	PALCE29M16	PALCE29MA16
PAL32VX10	PALCE610	PLS105	PLS167	PLS168	PLS30S16
MACH 1	MACH 2				

SYNTAX

You use this reserved word in a functional equation in the equations segment of Boolean and state designs.

Syntax

Pn.TRST	Assignment Operator	Expression
---------	---------------------	------------

Example

EQUATIONS

Q0	=	/Q0
Q1.TRST	=	I1 * /I2
...		

Definitions

All parameters are defined below.

Pn.TRST

Pn.TRST is associated with the three-state buffer. The name must be defined in an earlier PIN or NODE statement in the declaration segment.

Assignment Operator

The assignment operator is a symbol that defines a specific operation, as interpreted by the software when processing design files.¹²⁶

¹²⁶ Refer to ASSIGNMENT OPERATOR, in this chapter, for additional details.

Expression

Expression defines the logic conditions that determine when to enable three-state outputs on devices with programmable enable.

USE

Multiple .TRST statements for the same pin or node are automatically ORed together into one statement. This can result in an error during either assembly or fitting.

You can use more than one product term for a three-state buffer depending on the device. Some devices using product terms also have a pin for controlling the buffer which overrides the logic.

You **cannot** complement the pin name. For example, if a signal is defined as Q1 in the PIN statement, /Q1.TRST is not permitted. You **can** complement the Boolean expression, if supported by the device, as follows:

```
Q1.TRST = I1 * I2 * I3 for active high
Q1.TRST = /(I1 * I2 * I3) for active low
```

.TRST provides several logical means of enabling the outputs on some devices, such as the PALCE29M16. Examples follow.¹²⁷

- A dedicated output:
Q[1..4].TRST = VCC
- A dedicated input:
O[5..8].TRST = GND

¹²⁷ Refer to Chapter 11, in this section, for additional details regarding devices that support programmable enable.

.TRST

- A product-term enable, XOR:
 $O[9..12].TRST = I1 * I2 :*: I3$
- OE pin enable, where there is a choice between a product term and a dedicated output enable pin:
 $O[1..4].TRST = IOE$
- Unconditional high, where an output equation exists for a pin and no .TRST equation exists:
 $Q.TRST = VCC$
- Unconditional low, where no output equations or .TRST equation is defined:
 $Q.TRST = GND$

Many PAL devices have dedicated enable pins to control some or all three-state outputs. For these outputs, no .TRST equation is needed.¹²⁸

¹²⁸ Refer to the following topics, in this chapter, for additional details: BOOLEAN EQUATION, EXPRESSION, FUNCTIONAL EQUATIONS, GROUP, STRING, and VECTOR.

VCC

You can include this reserved word in an equation to hold a pin, node, or functional equation unconditionally high.

Devices Supported: All PLD devices.
--

SYNTAX

You use the reserved word, VCC, in the equations segment of Boolean and state-machine designs.

Syntax

Pn or Functional Equation	Assignment Operator	VCC
------------------------------	---------------------	-----

Example

EQUATIONS

...

OUT5	=	VCC
OE1.TRST	=	VCC

...

Definitions

The element preceding the reserved word is described below.

Pn or Functional Equation

Pn or functional equation defines the element to be held high.

- The pin or node name defined in the PIN or NODE statement of the declaration segment
- The pin or node function defined in an earlier functional equation.¹²⁹

¹²⁹ Refer to the following topics, in this chapter, for additional details: FUNCTIONAL EQUATIONS, NODE, and PIN.

USE

VCC is normally used for functional equations. You can use 1 instead of VCC anywhere you want an unconditional high value.¹³⁰

Important: You must define the VCC pin in a PIN statement.

¹³⁰ Refer to GND, in this chapter, for additional details.

VECTOR

VECTOR notation allows you to assign a set of descriptions to a set of pins or nodes. It also allows you to compare the value of a set of pins or nodes to a radix in a CASE statement.

Devices Supported: All PLD devices.
--

SYNTAX

Use vector notation in Boolean and state-machine designs.

Syntax

PIN	x1..xn	NUM[y1..yn]
-----	--------	-------------

Example

DECLARATION

PIN 1 INA COMB

...

PIN 5..2 OUT[4..1]

PIN 6, 10..8 DATA[4..1]

PIN 18..11 ADD[7..0]

...

EQUATIONS

...

CASE (ADD[7..0])

BEGIN

#hOF:

BEGIN

...

END

END

...

Definitions

Parameters following the keyword PIN are defined below. Additional details are discussed under Use.

x1..xn

This notation specifies the user-defined pin number range. You must have the same number of pin numbers as pin names.

y1..yn

This notation specifies the user-defined pin name range. You must have the same number of pin names as pin numbers.

VECTOR is essentially a dual range syntax: "for this range of pins use this range of names."¹³¹

USE

To use VECTOR notation, you must do the following.

- Declare all pins in the vector in one PIN statement.
- Use subscripted pin or node names with the format NAME[1] rather than NAME 1.

You can include input and output pins in the same vector if your application calls for it, but you cannot include pins and nodes in the same vector.

In the syntax example, the software converts the pin definitions as follows.

```
PIN 2  OUT[1]
PIN 3  OUT2]
...
PIN 18 ADD[7]
```

In the CASE statement, the software tests the value on pins ADD[7] through ADD[0]. If they match the hex value 0F, the instructions between the BEGIN and END statements are executed.

¹³¹ Refer to OPERATOR, in this chapter, for additional details.

WHILE-DO

This is a simulation construct that looks at a logical condition and performs a specified task as long as the condition is true.

Devices Supported: All PLD devices.
--

SYNTAX

Use the WHILE-DO construct in an auxiliary simulation file or the simulation segment of Boolean and state-machine designs.

Syntax

```
WHILE (Condition) DO
    BEGIN
                                Task
    END
```

Example

```
SIMULATION
...
    WHILE (/BIT2 * /BIT3) DO
        BEGIN
                                CLOCKF CLOCK
        END
...

```

Definitions

The following structures are part of the WHILE-DO construct.

Condition

Condition is any Boolean expression. You can use more than one condition if you separate them by commas; the software ANDs multiple conditions together. If the WHILE-DO construct is nested in a FOR-TO-DO construct, the condition can also be the index variable of the FOR-TO-DO construct. You cannot use an index variable outside its defining FOR-TO-DO construct.

Use parentheses to enclose the WHILE condition. However, you cannot nest parentheses.

WHILE-DO

Task The simulation task the software performs during the WHILE-DO loop. Use BEGIN and END statements to enclose the task; indent the statements to make your program easier to follow.

USE You can nest WHILE-DO constructs within CASE, FOR-TO-DO, IF-THEN-ELSE, and other WHILE-DO constructs. There is no limit to the number of constructs you can include in your design. However, using a minimal number of nests may make your program easier to follow and faster to compile.

The condition can be any Boolean expression of logic signals or mathematical equality: =, >, <, >=, <=, and <>. ¹³²

¹³² Refer to the following topics, in this chapter, for additional details: FOR-TO-DO, IF-THEN-ELSE, and SIMULATION.

CHAPTER 11

DEVICE PROGRAMMING REFERENCE

CONTENTS

DEVICE PROGRAMMING REFERENCE	1
11.1 PLD INTRODUCTION	2
11.1.1 PLD NAMING CONVENTIONS.....	2
11.1.2 STANDARD PLD DEVICES VERSUS NON-STANDARD PLD DEVICES	3
11.2 DEVICE FEATURE CROSS-REFERENCE	4
11.3 GENERAL PLD LANGUAGE SYNTAX	6
11.3.1 OUTPUT-ENABLE CONTROL.....	7
11.3.1.1 Common External Output-Enable Pin	7
11.3.1.2 Individual Product Term Control	7
11.3.1.3 Common External Pin or Individual Product Term Control.....	8
11.3.2 CLOCK CONTROL.....	10
11.3.2.1 Common External Clock Control	10
11.3.2.2 Individual Product Term Clock Control	11
11.3.3 PRESET CONTROL.....	11
11.3.3.1 Individual Product Term Control	12
11.3.3.2 Global Product Term Control	12
11.3.4 RESET CONTROL	13
11.3.4.1 Individual Product Term Control	13
11.3.4.2 Global Product Term Control	14
11.3.5 DEVICE POLARITY	15
11.3.5.1 Active-Low Polarity	15
11.3.5.2 Active-High Polarity	16
11.3.5.3 Programmable Polarity	18
11.3.6 COMBINATORIAL LOGIC	19
11.3.7 REGISTERED OR LATCHED LOGIC.....	20
11.3.7.1 D Flip-Flop	20
11.3.7.2 SR Flip-Flop.....	21
11.3.7.3 Latch.....	22
11.3.8 FEEDBACK	23
11.3.8.1 Programmable Feedback	23
Output with I/O Feedback	23
Output with /Q Feedback.....	25

	Output with I/O and /Q (Dual) Feedback.....	26
	Buried Register with /Q Feedback.....	28
	Buried Register with Q Feedback	30
	Registered Input with /Q Output.....	31
11.3.8.2	Non-Programmable Feedback.....	32
	Combinatorial or Registered Output with I/O Feedback.....	32
	Registered Output with /Q Feedback	33
11.3.9	PRELOAD CONTROL.....	33
11.3.9.1	Supervoltage	33
11.3.9.2	Product Term Control.....	33
11.3.10	OBSERVABILITY PRODUCT TERM CONTROL.....	34
11.3.11	COMPLEMENT ARRAY	35
11.3.12	ELECTRONIC SIGNATURE.....	36
11.4	PLD DEVICE SYNTAX DATASHEETS.....	38
11.4.1	PIN AND NODE DESCRIPTIONS.....	38
11.4.2	BLOCK AND MACROCELL DIAGRAM(S).....	39
11.4.3	SPECIAL PROGRAMMING FEATURES	39
	105	40
	167/168	44
	16RA8	48
	16V8HD.....	52
	20EG8	62
	20EV8	66
	20RA10	70
	22IP6	74
	22V10.....	80
	23S8.....	84
	26V12.....	92
	29M16	96
	29MA16	104
	30S16.....	110
	32VX10	130
	610	142
11.5	MACH 1 AND MACH 2 SERIES DEVICES.....	156
11.5.1	OVERVIEW	156
11.5.1.1	Device Features.....	156
11.5.1.2	Pin and Node Descriptions	157
11.5.1.3	PALASM Programming Features	157
11.5.2	SAMPLE EQUATIONS.....	168

11.5.2.1	I/O Cell and Macrocell	169
11.5.2.2	Pin and Node Feedback	170
11.5.2.3	Registered and Latched Inputs.....	171
MACH 110 Device.....		174
MACH 120 Device.....		178
MACH 130 Device.....		182
MACH 210 Device.....		186
MACH 220 Device.....		190
MACH 230 Device.....		194
MACH 215 Device.....		198

This chapter provides datasheets on the PALASM language syntax, examples of language use, and information related to programming PLD and MACH devices.

- The PLD introduction, 11.1, discusses the purpose of, and guidelines for using, this device programming reference.
- The PLD cross-reference table, 11.2, tabulates programming features for each device and indicates where you can find a language syntax example for each feature.
- The general PLD language syntax, 11.3, explains general device features and shows the language syntax needed to use them.
- The PLD device syntax datasheets, 11.4, provide information relating to the PALASM language for devices with special features.

Important: Standard devices do not have a PLD device syntax datasheet.

- The MACH 1 and MACH 2 series, 11.5., furnishes device, language, and programming information.

11.1 PLD INTRODUCTION

This chapter is organized so you can quickly and easily familiarize yourself with PALASM language constructs and the syntax required for designing with different programmable devices.

- Review the feature cross-reference table for programming features that apply to the device you want to design with.
- Read the general syntax for programming features marked with an X in the cross-reference table.
- Read the corresponding device syntax for features marked with an asterisk, *, in the cross-reference table.
- For features marked with an ampersand, @, in the device feature cross-reference table, read the general syntax datasheets for the language syntax and the corresponding device syntax datasheet for node locations and descriptions.¹

11.1.1 PLD NAMING CONVENTIONS

The naming of devices follows the convention used in the PAL Device Data Book. For clarity, only the number of a device is addressed. For example, 22V10 includes the devices PAL22V10, AmPAL22V10, and PALCE22V10; 16R8 includes the family of devices PAL16L8, PAL16R8, PAL16R6, and PAL16R4. The actual devices are listed under the alphanumeric device reference index. For each device syntax datasheet, the actual devices are listed under the device number heading.

¹ Refer to Chapter 10, in this section, for an in-depth discussion of particular language elements.

11.1.2 STANDARD PLD DEVICES VERSUS NON- STANDARD PLD DEVICES

For ease of reference, devices whose features are covered by the **general** syntax data sheet are grouped as standard programmable devices.² Devices that require specific language syntax are grouped as non-standard programmable devices.³

105	PLS105 / PLSCE105	11-42
167/168	PLS167 / PLSCE167	11-46
16RA8	PAL16RA8	11-50
16V8HD	PAL16V8HD	11-54
20EG8	PAL10H20EG8 / PAL10020EG8	11-64
20EV8	PAL10H20EV8 / PAL10020EV8	11-68
20RA8	PAL20RA8	11-72
22IP6	PAL20IP6	11-76
22V10	PAL22V10 / AmPAL22V10 / PALCE22V10	11-82
23S8	AmPAL23S8	11-86
26V12	PALCE26V12	11-92
29M16	PALCE29M16	11-98
29MA16	PALCE29MA16	11-106
30S16	PLS30S16	11-112
32VX10	PAL32VX10	11-132
610	PALCE610	11-144

² Standard programmable devices include 16R8 family, 16V8, 18P8, 20R8 family, 20V8, 20X10/20L10 family, 22P10, 24R10 family, and 24V10.

³ Non-standard programmable devices include 105, 167/168, 16RA8, 16V8HD, 20EG8, 20EV8, 20RA8, 22IP6, 22V10, 23S8, 26V12, 29M16, 29MA16, 30S16, 32VX10, and 610.

11.2 DEVICE FEATURE CROSS-REFERENCE

The following table cross references the features for each programmable device. It also identifies whether a particular device requires special language constructs and where to look for that information.

PLD Device Feature Cross-Reference Table	Standard Programmable Devices										Non-Standard Programmable Devices																		
Features	16L8/20L8	16R4/20R4	16R6/20R6	16R8/20R8	16V8/20V8	18P8	20L10/24L10	20X4/20X8	20X10	22P10	24R4/24R8	24R10	24V10	105	167/168	16RAB	16V8HD	20EG8	20EV8	20RA10	22IP6	22V10	23S8	26V12	29M16	29MA16	30S16	32VX10	610
Output-Enable Control																													
Common		X	X	X			X	X		X	X																		
Individual					X				X								X	X		X	X		X					X	
Common / individual					X							X				X										X			
Others														*	*	*			*			*		*		*	*	*	*
Clock Control																													
Common	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X		X					X	
Individual															X				X										
Others																*								*	*	*	*	*	*
Preset Control																													
Individual																X			X							X			
Global																	X	X		X	X	X	X	X	X			X	
Others														*	*				*			*		*	*	*	*	*	*
Reset Control																													
Individual															X			X								X			
Global																				X	X	X	X	X	X			X	
Others																			*		*		*		*	*	*	*	*
Device Polarity																													
Active low	X	X	X	X			X	X	X	X	X																		
Active high													X	X														X	
Programmable					X	X			X		X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Output Logic Types																													
Combinatorial	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
D flip-flop		X	X	X	X		X	X	X	X	X			X	X		X	X		X	X	X	X	X	X	X	X	X	*
SR flip-flop													X	X					*								X	*	
Latch																X								X	X				
T flip-flop																			*									*	
JK flip-flop																												*	
Macrocells - different configs.																													
Non-programmable Feedback																													
Comb. output - I/O feedback	X	X	X	X	X	X	X	X	X	X	X			X				X	X	X	X								
Reg. output - I/O feedback														X				X	X	X									
Reg. output - /Q feedback	X	X	X	X			X	X	X	X	X									X									

Notes: *1 PAL10H20G8, PAL10020G8
 *2 PAL10H20V8, PAL10020V8
 X See the general PLD L language Syntax for general language syntax; if necessary, see also the corresponding PLD Device Syntax datasheet for node descriptions and locations.
 * See the corresponding PLD Device Syntax datasheet for device dependent features.

PLD Device Feature Cross-Reference Table (Continued)	Standard Programmable Devices										Non-Standard Programmable Devices																		
Features	16L8/20L8	16R4/20R4	16R6/20R6	16R8/20R8	16V8/20V8	18P8	20L10/24L10	20X4/20X8	20X10	22P10	24R4/24R8	24R10	24V10	105	167/168	16R8	16V8HD	20EG8	20EV8	20RA10	22IP6	22V10	23S8	26V12	29M16	29MA16	30S16	32VX10	610
Programmable Feedback																													
Output with I/O feedback																		X	X			X	X	X	X		*	*	
Output with /Q feedback																						X	X	X	X		*	*	
Output with Q feedback																											*	*	
Output with I/O and /Q feedback																								X	X		*	*	
Output with I/O and Q feedback																											*	*	
Output with I/O and latch feedback																											*	*	
Buried register with /Q feedback																						X	X	X			*	*	
Buried register with Q feedback													X	X													X	*	
Register input with /Q																								X	X				
Latch input																	*												
Preload Control																													
Supervoltage				X						X							X	X	X		X	X	X	X	X	X	X	X	
Product term																*			*					X	X				
Others																*													
Observability Product Term Control																													
Miscellaneous																													
Complement Array													X	X													X		
State Bits													*	*													*		
Electronic Signature				X						X						X												*	
Bypassable Register																												*	
Input Latch																	*												
Input Register																											*		
Open-Collector Output																	*										*		
Notes:	*1 PAL10H20G8, PAL10020G8 *2 PAL10H20V8, PAL10020V8 X See the general PLD Language Syntax for general language syntax; if necessary, see also the corresponding PLD Device Syntax datasheet for node descriptions and locations. * See the corresponding PLD Device Syntax datasheet for device dependent features.																												

11.3 GENERAL PLD LANGUAGE SYNTAX

Each discussion below explains a feature commonly found in most programmable devices.⁴

- 11.3.1, Output-Enable Control
- 11.3.2, Clock Control
- 11.3.3, Preset Control
- 11.3.4, Reset Control
- 11.3.5, Device Polarity
- 11.3.6, Combinatorial Logic
- 11.3.7, Registered or Latched Logic
- 11.3.8, Feedback
- 11.3.9, Preload Control
- 11.3.10, Observability Product Term Control
- 11.3.11, Complement Array
- 11.3.12, Electronic Signature

Each discussion consists of the following information.

- Description of the feature
- Pertinent, standard PALASM language syntax
- Example(s) of the language as it is used

All language elements required to design with standard programmable-device features and those elements needed for certain general features of non-standard programmable devices are included. Use this information in conjunction with the device programming syntax datasheet when designing with non-standard programmable devices.

Note: In each syntax example, *italicized* information is provided as an explanation and is not part of the actual syntax.

⁴ Refer to the Device Programming Feature Cross-Reference Table for more information about individual device features.

11.3.1 OUTPUT-ENABLE CONTROL

Three major types of output-enable control are identified below and discussed next.

- Common external output
- Individual product term control
- Common external or individual product term control

11.3.1.1 Common External Output-Enable Pin

A common external output-enable pin controls all three-state buffers within the device. Since the three-state buffers are externally controlled by an input pin, **no language syntax is required.**

Devices Supported: Devices with common external output-enable control include the 16R8, 20R8, 20X10/20L10, and 24R10.

11.3.1.2 Individual Product Term Control

Each three-state buffer is controlled individually by a product term. To use these product terms, define each product term control individually in the language syntax and example, as shown next.

Devices Supported: Devices with individual product term output-enable control include the 20EG8, 20EV8, 22IP6, 26V12, and 32VX10.

Note: For the 20EG8 and 20EV8, even though the output buffer is two-state instead of three-state, the syntax you use to define the product term control is the same as shown next.

Syntax

Pin Statement(s) :
 PIN Output_pin_location Output_pin_name Storage_type
 :

Equation(s) Output_pin_name.TRST = Boolean expression using one product term

Example

```
:  
CHIP EXAMPLE PALCE16V8  
:  
PIN      2    I1    COMB      ;input  
PIN      3    I2    COMB      ;input  
PIN     12    O1    COMB      ;output  
:  
O1.TRST = I1 * I2                ;output enable controlled by product term (I1 * I2)
```

11.3.1.3 Common External Pin or Individual Product Term Control

Each three-state buffer can be controlled using either of the following methods.

- An external, common output-enable pin
- An individual product term

In addition, these three-state buffers can be either

- permanently enabled or
- permanently disabled.

Use the syntax shown in the following examples to define the three-state buffer control.

Devices Supported: Devices that have output enable with common external or individual product term are the 16V8, 16V8HD, 20V8, 24V10, and 29MA16.

Note: The output enable defaults are as follows.

- If the pin is defined as an output,
pin.TRST = VCC
- If the pin is not defined as an output,
pin.TRST = GND

Syntax

```
Pin Statement(s) :
    PIN Output_pin_location      Output_pin_name      Storage_type
    PIN Output_enable_pin_location Output_enable_pin_name Storage_type
:
```

```
Pin Statement(s) :
    Output_pin_name.TRST
    external common output-enable pin
    = Output_enable_pin_name
    or
    individual product term
    = Boolean expression using one product term
    or
    permanently enabled
    = VCC      ;default for pins defined as outputs
    or
    permanently disabled
    = GND      ;default for pins not defined as outputs
```

Example 1

External common-enable pin control

```
:
CHIP EXAMPLE PALCE16V8
:
PIN    11  IOE  COMB      ;input
PIN    12  01  COMB      ;output
:
01.TRST = IOE              ;output controlled by input pin IOE
```

Example 2

Individual product term output-enable control

```
:
CHIP EXAMPLE PALCE16V8
:
PIN    2   I1  COMB      ;input
PIN    3   I2  COMB      ;input
PIN    12  01  COMB      ;output
:
01.TRST = I1 * I2         ;output enable controlled by product term (I1 * I2)
```

Example 3

Output enable permanently enabled

```
:
CHIP EXAMPLE PALCE16V8
:
PIN      12   01   COMB      ;output
:
01.TRST = VCC                ;output permanently enabled
```

Example 4

Output enable permanently disabled

```
:
CHIP EXAMPLE PALCE16V8
:
PIN      12   01   COMB      ;output
:
01.TRST = GND                ;output permanently disabled
```

11.3.2 CLOCK CONTROL

In general, there are two types of clock control for registered and latched programmable devices.

- Common external clock control
- Individual product term clock control

The two clock control types are discussed below.

11.3.2.1 Common External Clock Control

A dedicated clock pin is used to clock all registers in the device. Since the clock configuration is fixed for these devices, no special language syntax is required.

<p>Devices Supported: Devices that have a common external clock pin include the 105, 167/168, 16R8, 16V8, 18P8, 20EG8, 20EV8, 20R8, 20V8, 20X10, 22P10, 22V10, 23S8, 24R10, 24V10, and 32VX10.</p>

11.3.2.2 Individual Product Term Clock Control

The clock input for each register is individually controlled by a product term. You must use the syntax below to program each product term for the clock control.

Devices Supported: Devices that have clock control through an individual product term are the 16RA8 and 20RA10.

Syntax

```
Pin Statement(s) :  
                  PIN Output_pin_location      Output_pin_name      Storage_type  
                  :  
Equation(s)      Output_pin_name.CLKF = Boolean expression using one product term
```

Example

Individual product term clock control

```
:  
CHIP EXAMPLE 16RA8  
:  
PIN    2    I1    COMB    ;input  
PIN    3    I2    COMB    ;input  
PIN    12   01    REG     ;registered output  
:  
01.CLKF = I1 * /I2      ;clock input to the register of output 01 is controlled  
                        ; by the product term (I1* /I2)
```

11.3.3 PRESET CONTROL

There are two types of preset control.

- Individual product term control
- Global product term control ⁵

⁵ Refer to discussion 11.4 for node locations and information.

11.3.3.1 Individual Product Term Control

Each preset input is individually controlled by a programmable product term. To program the preset product terms, you use the language syntax to define each preset input, as shown next.

Devices Supported: Devices that have preset/reset control with an individual product term are the 16RA8, 20RA10, and 29MA16.

Syntax

```
Pin Statement(s) :  
                PIN Output_pin_location      Output_pin_name      Storage_type  
                :  
Equation(s)     Output_pin_name.SETF = Boolean expression using one product term
```

Example

Preset with individual product term control

```
:  
CHIP EXAMPLE PAL16RA8  
:  
PIN      2      I1      COMB      ;input  
PIN      3      I2      COMB      ;input  
PIN      12     01      REG       ;registered output  
:  
01.SETF = I1 * /I2                ;preset input to the register of output 01 is controlled  
;      by the product term (I1 * /I2)
```

11.3.3.2 Global Product Term Control

The preset inputs of all registers within the device are controlled by a common global preset product term. To program this global product term, you must declare the global node in the pin statement segment, as shown next.

Devices Supported: Devices that have preset with global product term control are the 20EG8, 20EV8, 22V10, 23S8, 26V12, 29M16, and 32VX10.

Syntax

```
Pin Statement(s) :  
                NODE Global_node_location   Global_node_name  
                :  
Equation(s)     Global_node_name.SETF = Boolean expression using one product term
```

Example

Preset with global product term control

```
:  
CHIP EXAMPLE  
:  
PIN    2    I1    COMB    ;input  
PIN    3    I2    COMB    ;input  
:  
NODE   1    GLOBAL    ;internal global node  
:  
GLOBAL.SETF = I1 * /I2    ;preset inputs to all registers are globally controlled  
                        ; by the product term (I1 * /I2)
```

11.3.4 RESET CONTROL

There are two types of reset control.

- Individual product term control
- Global product term control⁶

11.3.4.1 Individual Product Term Control

Each reset input is individually controlled by a programmable product term. To program the preset product terms, you use the language syntax to define each preset input, as shown next.

Devices Supported: Devices that have reset control with an individual product term are the 16RA8, 20RA10, 29MA16, and 610.

⁶ Refer to discussion 11.4 for node locations and information.

Syntax

```
Pin Statement(s) :  
                PIN Output_pin_location      Output_pin_name      Storage_type  
                :  
Equation(s)     Output_pin_name.RSTF = Boolean expression using one product term
```

Example

Reset with individual product term control

```
:  
CHIP EXAMPLE PAL16RA8  
:  
PIN      2    I1    COMB      ;input  
PIN      3    I2    COMB      ;input  
PIN     12    01    REG      ;registered output  
:  
01.RSTF = I2                ;reset input to the register of output 01 is controlled  
                            ; by the product term (I2)
```

11.3.4.2 Global Product Term Control

The reset inputs of all registers within the device are controlled by a common global reset product term. To program this global product term, you must declare the global node in the pin statement segment, as shown below.

Devices Supported: Devices that have preset/reset with global product term control are the 22V10, 23S8, 26V12, 29M16, and 32VX10.
--

Syntax

```
Pin Statement(s) :  
                NODE Global_node_location  Global_node_name  
                :  
Equation(s)     Global_node_name.RSTF = Boolean expression using one product term
```

Example

Reset with global product term control

```
:  
CHIP EXAMPLE  
:  
PIN      2      I1      COMB      ;input  
PIN      3      I2      COMB      ;input  
:  
NODE     1      GLOBAL      ;internal global node  
:  
GLOBAL.RSTF = I2      ;reset inputs to all registers are globally controlled  
                        ; by the product term (I2)
```

11.3.5 DEVICE POLARITY

There are three types of device polarity.

- Active low
- Active high
- Programmable

You must be careful about both the pin and equation output polarities when you write equations for each type of device.

11.3.5.1 Active-Low Polarity

Active-low devices are those programmable devices whose outputs pass through the output pins inverted.

For these devices, the output of the equations always has the opposite polarity of the output pin. Examples below show how you can write the same equation for active-low or active-high outputs using active-low devices.

Devices Supported: Active-low devices are the 16R8, 20R8, 20X10/20L10, and 24R10.
--

Syntax

```
Pin Statement(s) :  
                PIN Output_pin_location      Output_pin_name      Storage_type  
                :
```

```
Equation(s)      Output_pin_name* = Boolean expression
```

* *Output_pin_name in output equation must have the opposite polarity of the pin statement.*

Example 1

Active-low output using active-low devices

```
:  
CHIP EXAMPLE PAL16R8  
:  
PIN    2    I1    COMB    ;input  
PIN    3    I2    COMB    ;input  
PIN    4    I3    COMB    ;input  
PIN    12   /O1   REG     ;active-low output pin  
:  
O1 = I1 * I2 + I3          ;equation with the opposite polarity of the pin  
; statement
```

Example 2

Active-high output using active-low devices

```
:  
CHIP EXAMPLE PAL16R8  
:  
PIN    2    I1    COMB    ;input  
PIN    3    I2    COMB    ;input  
PIN    4    I3    COMB    ;input  
PIN    12   O1    REG     ;active-high output pin  
:  
/O1 = /I1 * /I3           ;same equation with active-low output using DeMorgan's  
+ /I2 * /I3              ; theorem /O1 = /((I1*I2) + I3)
```

11.3.5.2 Active-High Polarity

Active-high devices are those programmable devices whose outputs pass through the output pins without inversion. For these devices the pin output and the equation output are always the same polarity.

Devices Supported: Active-high devices are the 105, 167/168, and 30S16.

Syntax

Pin Statement(s) :
 PIN Output_pin_location Output_pin_name Storage_type
 :
Equation(s) Output_pin_name* = Boolean expression

* *Output_pin_name* in output equation must have the opposite polarity of the *pin* statement.

Example 1

Active-high output using active-high devices

```
:
CHIP EXAMPLE PLS105
:
PIN     2     I1     COMB       ;input
PIN     3     I2     COMB       ;input
PIN     4     I3     COMB       ;input
PIN     12    O1     COMB       ;active-high output pin
:
O1 = I1 * I2 + I3               ;equation with the same polarity as the pin statement
```

Example 2

Active-low output using active-high devices

```
:
CHIP EXAMPLE PLS105
:
PIN     2     I1     COMB       ;input
PIN     3     I2     COMB       ;input
PIN     4     I3     COMB       ;input
PIN     12    /O1    COMB       ;active-low output pin
:
/O1 = /I1 * /I3                 ;same equation with active-low output using DeMorgan's
     + /I2 * /I3               ;   theorem /O1 = /((I1*I2) + I3)
```

11.3.5.3 Programmable Polarity

You can individually program the polarity of each output.

- If you want an active-high output pin, use the same output polarity for both the pin statement and the equation.
- If you want an active-low output pin, use the opposite polarity for the pin statement and the equation.

Devices Supported: Devices with programmable polarity are the 16RA8, 16V8, 16V8HD, 18P8, 20EG8, 20EV8, 20RA10, 20V8, 22IP6, 22P10, 22V10, 23S8, 24V10, 26V12, 29M16, 29MA16, 32VX10, and 610.

For consistency, all examples use an active-high polarity in each pin statement for each output pin. In this case, the output equation controls the polarity of the pin.⁷

Syntax

<i>Pin Statement(s)</i>	:	PIN	Output_pin_location	Output_pin_name*	Storage_type
	:				
<i>Equation(s)</i>		Output_pin_name*	= Boolean expression		

* *Output_pin_name* can be active high or active low.

⁷ Refer to Section II, Chapter 4, for details on polarity.

Example 1

Active-high output from a device with programmable polarity

```
:
CHIP EXAMPLE PAL16R8
:
PIN      2      I1      COMB      ;input
PIN      3      I2      COMB      ;input
PIN      4      I3      COMB      ;input
PIN      12     01      COMB      ;active-high output pin
:
/01 = I1 * I2 + I3                ;equation with same the polarity as the output pin
```

Example 2

Active-low output from a device with programmable polarity

```
:
CHIP EXAMPLE PAL16R8
:
PIN      2      I1      COMB      ;input
PIN      3      I2      COMB      ;input
PIN      4      I3      COMB      ;input
PIN      12     01      COMB      ;active-low output pin
:
/01 = I1 * I2 + I3                ;equation with the opposite polarity of the pin
;      statement
```

11.3.6 COMBINATORIAL LOGIC

For outputs that use only combinatorial logic, you define the storage type as COMBINATORIAL or use the abbreviation, COMB, in the pin declaration segment of the PDS file. See the syntax description and example shown next.

Note : The default storage type is combinatorial, which is used if you do not define the type.

Syntax

```
Pin Statement(s)  :  
                  PIN  Output_pin_location      Output_pin_name      COMB  
                  :  
Equation(s)      Output_pin_name = Boolean expression
```

Example

Combinatorial output

```
:  
CHIP EXAMPLE PAL16R6  
:  
PIN    2    I1    COMB    ;input  
PIN    3    I2    COMB    ;input  
PIN    4    I3    COMB    ;input  
PIN   12    O1    COMB    ;output  
:  
O1 = I1 * I2 + I3          ;Boolean expression
```

11.3.7 REGISTERED OR LATCHED LOGIC

For outputs that use registered or latched logic, you must define the corresponding output type in the pin declaration segment. On some programmable devices the registers can be programmed into different types, while others have a fixed hardware configuration. In general, there are three types of registers. Their corresponding language syntax is described below.

- D Flip-flop
- SR Flip-flop
- Latch

11.3.7.1 D Flip-Flop

To use registered logic with D Flip-flops, you must define the output type as REGISTERED or REG and write the corresponding equation for the registered output.

Devices Supported: Devices that use D flip-flops are the 16RA8, 16R8, 16V8, 16V8HD, 20EV8, 20RA9, 20R8, 20V8, 20X10/20L10, 22V10, 24R10, 24V10, 26V12, 29M16, 29MA16, 32VX10, and 610.

Syntax

```

Pin Statement(s) :
                  PIN Output_pin_location      Output_pin_name      REG
                  :

```

```

Equation(s)      Output_pin_name = Boolean expression

```

Example

Registered output with D flip-flop

```

CHIP EXAMPLE PAL16R8
:
PIN    2    I1    COMB    ;input
PIN    3    I2    COMB    ;input
PIN    4    I3    COMB    ;input
PIN    12   O1    REG     ;registered output
:
O1 = I1 * I2 + I3        ;equation for registered output O1

```

11.3.7.2 SR Flip-Flop

To use registered logic with SR flip-flops, define the output type as REGISTERED or REG and provide one corresponding equation for each of the R and S inputs.

Devices Supported: Devices that use SR flip-flops are the 105, 167/168, 30S16, and 610.
--

Syntax

```

Pin Statement(s) :
                  PIN Output_pin_location      Output_pin_name      REG
                  :

```

```

Equation(s)      Output_pin_name.R = Boolean expression
                  Output_pin_name.S = Boolean expression

```

Example

Registered output with SR flip-flop

```
:
CHIP EXAMPLE PLS167
:
PIN      2    I1    COMB      ;input
PIN      3    I2    COMB      ;input
PIN      4    I3    COMB      ;input
PIN      9    O1    REG       ;registered output
:
O1.R = I1 * I2 + I3          ;equation for the R input of the register O1
O1.S = /I1                   ;equation for the S input of the register O1
```

11.3.7.3 Latch

To use latched logic, define the output type as LATCHED, or LAT, and write the corresponding equation for the latched output.

Devices Supported: Devices that use latched logic are the 20EG8, 29M16, and 29MA16.
--

Syntax

```
Pin Statement(s) :
                    PIN Output_pin_location      Output_pin_name      LAT
                    :


---


Equation(s)      Output_pin_name = Boolean expression
```

Example

Latch output

```
:
CHIP EXAMPLE PALCE29M16
:
PIN      2    I1    COMB      ;input
PIN      14   I2    COMB      ;input
PIN      23   I3    COMB      ;input
PIN      9    O1    LAT       ;latched output
:
O1 = I1 * I2 + I3          ;equation for latched output O1
```

11.3.8 FEEDBACK

You specify feedback of logic signals by defining the signals in the pin segment and specifying the logical construction in the equations or state segments. You may then use the defined signal names in other equations to accomplish feedback. This section details feedback specifications by syntax category.

11.3.8.1 Programmable Feedback

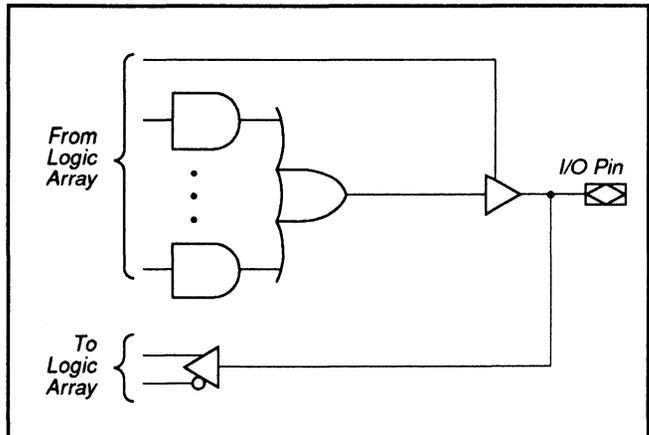
For those programmable devices that have a programmable feedback configuration, there are six possible configuration types.

- Output with I/O feedback
- Output with /Q feedback
- Output with I/O and /Q (dual) feedback
- Buried register with /Q feedback
- Buried register with Q feedback
- Registered input with /Q output.

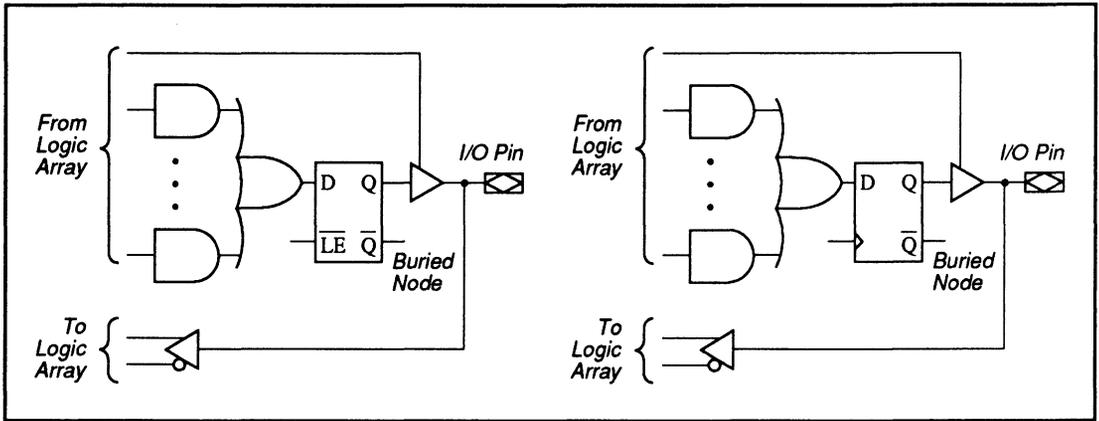
Output with I/O Feedback

The language syntax for combinatorial and registered output with the I/O feeding back to the arrays is shown next.

Devices Supported: Devices that have output with I/O feedback configuration are the 20EG8, 20EV8, 23S8, 26V12, 29M16, 29MA16, 32VX10, and 610.



Combinatorial Output with I/O Feedback



Registered/Latched Output with I/O Feedback

Syntax

Pin Statement(s) :

```
PIN I/O_pin_location I/O_pin_name <I/O_storage_type*>
```

:

Equation(s) I/O_pin_name = Boolean expression

:

< Output equations(s) using I/O_pin_name as feedback >

* Use the following table for the appropriate storage types.

<u>Output Type</u>	<u><I/O_storage_type></u>
Combinatorial	COMB
D flip-flop	REG
Latch	LAT

Example 1

Output with I/O feedback

```

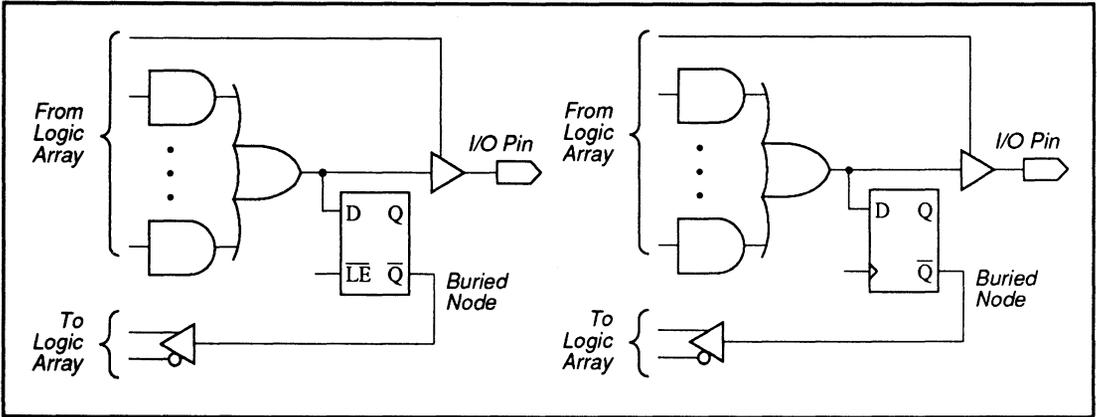
:
CHIP EXAMPLE PALCE29M16
:
PIN 2 I1 COMB ;input
PIN 14 I2 COMB ;input
PIN 3 IOF0 COMB ;I/O, combinatorial
PIN 15 IOF4 REG ;I/O, registered
IOF0 = I1 * I2 * IOF4 ;equation for IOF0 with feedback from registered
; I/O, IOF4
IOF4 = IOF0 * I1 ;equation with feedback from combinatorial I/O, IOF0

```

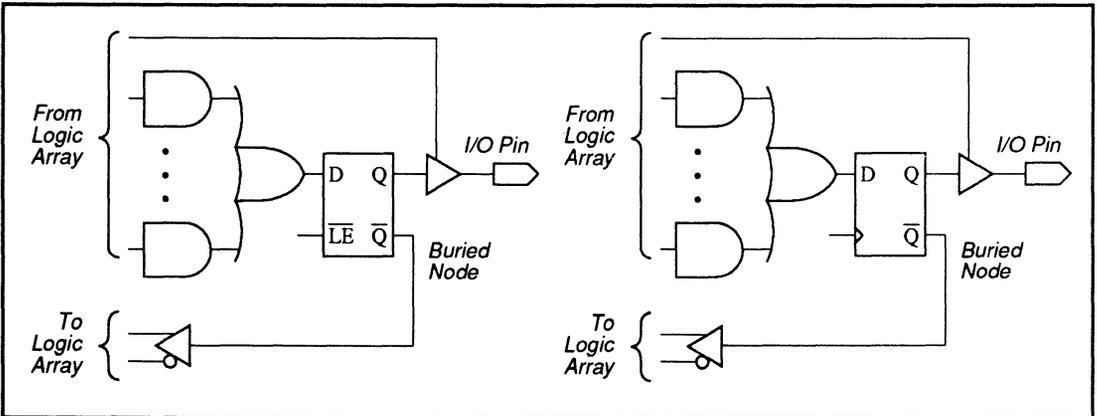
Output with /Q Feedback

The syntax for combinatorial and registered or latched output with the /Q register output feeding back to the arrays is shown below.

Devices Supported: Devices that have output with /Q feedback configuration are the 20EG8, 20EV8, 23S8, 26V10, 29M16, 29MA16, and 32VX10.



Combinatorial Output with /Q Feedback



Registered/Latched Output with /Q Feedback

Syntax

```
Pin Statement(s) :
    PIN I/O_pin_location I/O_pin_name <I/O_storage_type*>
    :
    NODE Buried_node_location Buried_node_name < Node_storage_type*>
    :
```

```
Equation(s)      I/O_pin_name = Boolean expression
                  /Buried_node_name = { I/O_pin_name** }
                  :
                  < Output equation(s) using either I/O_pin_name or Buried_node_name as
                    feedback(s) >
```

* Use the following table for the appropriate storage types.

<u>Storage element</u>	<u><I/O_storage_type></u>	<u><Node_storage_type></u>
D Flip-flop	COMB or REG	REG

** I/O_pin_name inside curly brackets must use the same polarity as defined on the left side of the output equation.

Example

Combinatorial output with /Q feedback

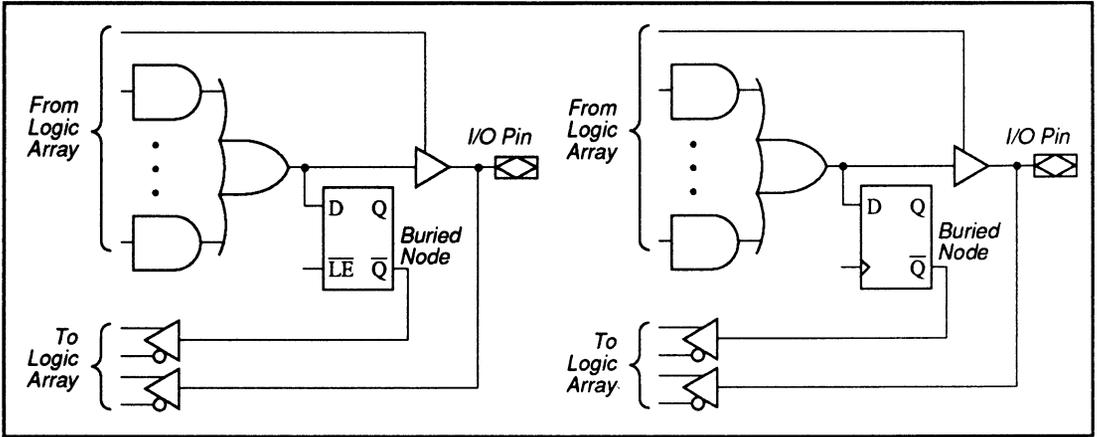
```
CHIP EXAMPLE PALCE29M16
:
PIN      2    IO    COMB      ;input
PIN      14   I1    COMB      ;input
PIN      9    IOF2  COMB      ;I/O, combinatorial
PIN      15   IOF4  COMB      ;I/O, combinatorial
NODE     9    RF2   ;buried node
:
/IOF2 = IO * I1          ;equation for IOF2
/RF2 = { /IOF2 }        ;define buried node /RF2 as /Q of IOF2's register
IOF4 = /RF2 * I1        ;equation with feedback from buried node RF2
```

Output with I/O and /Q (Dual) Feedback

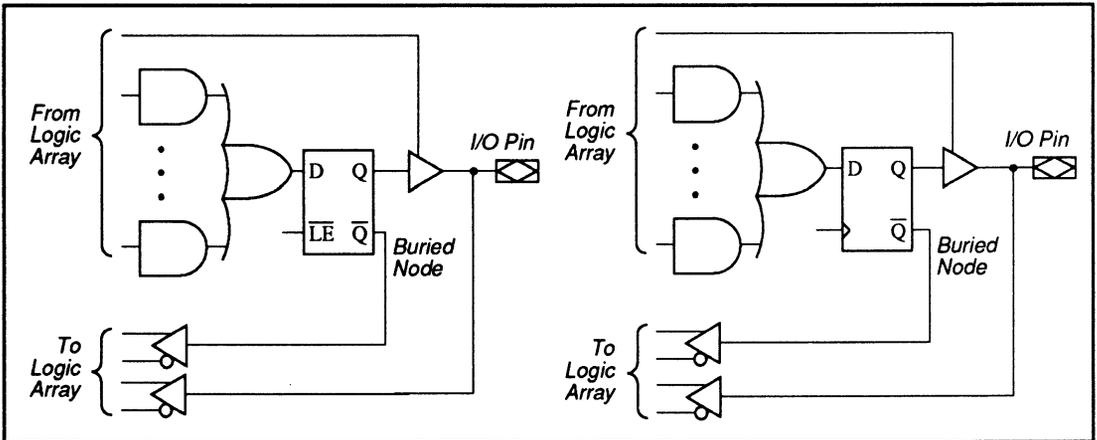
The language syntax for combinatorial and registered or latched output with both the I/O and the /Q register output feeding back to the array is shown below.

<p>Devices Supported: Devices that have output with both I/O and /Q configuration are the 29M16 and 29MA16.</p>
--

Note: This configuration is only possible in macrocells with dual feedback capability.



Combinatorial Output with I/O and /Q Feedback (Dual Feedback)



Registered/Latched Output with I/O and /Q Feedback (Dual Feedback)

Syntax

Pin Statement(s) :

```
PIN I/O_pin_location I/O_pin_name <I/O_storage_type*>
:
NODE Buried_node_location Buried_node_name <Node_storage_type*>
:
```

Equation(s)

```
I/O_pin_name = Boolean expression
/Buried_node_name = { I/O_pin_name** }
:
< Output equation(s) using either I/O_pin_name or buried_node_name,
or both, as feedback(s)>
```

* Use the following table for the appropriate storage types.

<u>Storage element</u>	<u><I/O storage type></u>	<u><Node storage type></u>
D Flip-flop	COMB or REG	REG

** I/O_pin_name inside curly brackets must use the same polarity as defined on the left side of the output equation.

Example

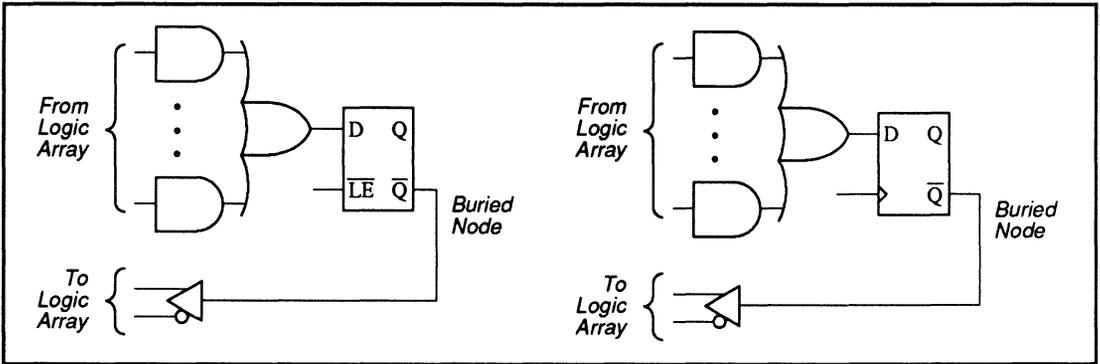
Registered output with I/O and /Q feedback

```
CHIP EXAMPLE PALCE29M16
:
PIN 2 IO COMB ;input
PIN 14 I1 COMB ;input
PIN 3 IOF0 COMB ;I/O, combinatorial
PIN 15 IOF4 REG ;I/O, registered
PIN 16 IOF5 REG ;I/O, registered
NODE 3 RFO ;buried node, of pin 3's register
:
/IOF0 = IO * I1 ;equation for combinatorial I/O /IOF0
/RFO = { /IOF0 } ;define /RFO as /Q output of IOF0
IOF4 = IOF0 * IO ;equation with I/O feedback IOF0
IOF5 = /RFO * I1 ;equation with buried node feedback /RFO
```

Buried Register with /Q Feedback

To use the /Q feedback of registers or latches that are buried, you must write an equation for the buried node. The language syntax for the buried /Q output feedback is illustrated below.

Devices Supported: Devices that have buried registers with /Q feedback configuration are the 23S8, 29M16, and 29MA16.



Buried Register with /Q Feedback

Syntax

Pin Statement(s) :
 NODE Buried_node_location Buried_node_name REG or LAT
 :

Equation(s) /Buried_node_name = Boolean expression
 :
 < Output equation(s) using buried_node_name as feedback >

Example

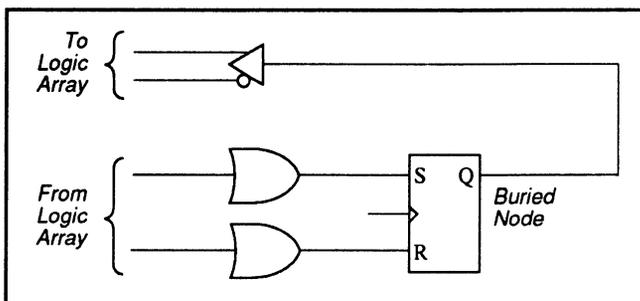
Buried register with /Q feedback

```
CHIP EXAMPLE PALCE29M16
:
PIN    2    IO    COMB    ;input
PIN    14   I1    COM     ;input
PIN    15   IOF4  REG     ;I/O, registered
NODE   6    R1    REG     ;Buried node
:
/R1 = IO * /I1           ;equation for buried node, R1, with negative polarity
IOF4 = /R1 * IO         ;equation using R1 as feedback
```

Buried Register with Q Feedback

To use the Q feedback of registers that are buried, you must write an output equation for the buried node of the Q output. The language syntax for the buried Q feedback is illustrated on the next page.

Devices Supported: Devices that have buried registers with Q feedback configuration are the 105, 167/168, and 30S16.



Buried Register with Q Feedback

Syntax

Pin Statement(s) :
 NODE Buried_node_location Buried_node_name REG
:

Equation(s)
 Buried_node_name.S = Boolean expression
 Buried_node_name.R = Boolean expression
:
 < Output equation(s) using buried_node_name as feedback >

Example

Buried registered with Q feedback

```

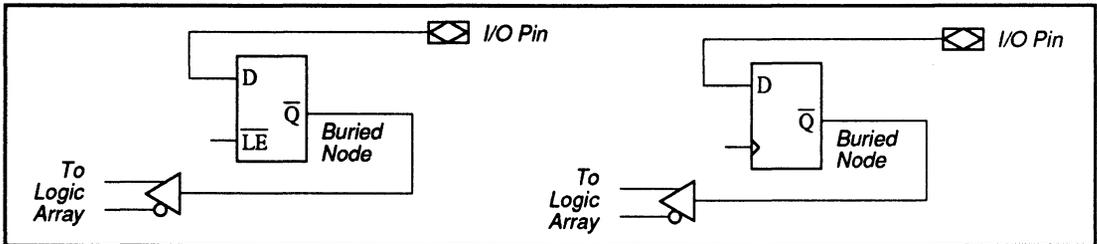
CHIP EXAMPLE PLS167
:
PIN      8   IO   COMB      ;input
PIN      7   I1   COMB      ;input
PIN      6   I3   COMB      ;input
PIN     14   P0   REG        ;output, registered
NODE     1   S0   REG        ;Buried node
:
S0.S = IO * /I1              ;equation for S input of the SR flip-flop with buried
                               ;   node S0
S0.R = I3 * I1               ;equation for R input of the SR flip-flop with buried
                               ;   node S0
P0 = S0 * IO                 ;equation using S0

```

Registered Input with /Q Output

The output macrocell can also be configured as an input register or latch with /Q output.

Devices Supported: Devices that have registered input with /Q output configuration are the 29M16 and 29MA16.



Registered/Latched Input with /Q Output

Syntax

```

Pin Statement(s) :
    PIN I/O_pin_location      I/O_pin_name      Storage_type
:
    NODE Buried_node_location  Buried_node_name  REG

```

Equation(s) < Output equation(s) using buried_node_name as feedback* >

* Pin name cannot appear on the left side of any equation.

Example

Registered input with /Q input

```
:  
CHIP EXAMPLE PALCE29M16  
:  
PIN      2    IO    COMB    ;input  
PIN     14    I1    COMB    ;input  
PIN      7    IO2   REG     ;I/O, registered  
PIN     15   IOF4   COMB    ;I/O, combinatorial  
NODE     7    R2    REG     ;Buried node  
:  
IOF4 = /R2 * IO + R2 * I1    ;equation using R2
```

11.3.8.2 Non-Programmable Feedback

Some programmable devices have feedbacks that are not programmable, that is, their feedback paths are fixed. There are two types of non-programmable feedback.

- Combinatorial or registered output with I/O feedback
- Registered output with /Q feedback

Combinatorial or Registered Output with I/O Feedback

If an output is configured to be combinatorial or registered, the I/O can be used to feedback directly into the logic array. To use the combinatorial output as feedback, no special language syntax is necessary. Simply write the output equation for that I/O pin, then use the I/O pin name in the Boolean expression, as required for other output equations.

Devices Supported: Devices that have combinatorial output with I/O feedback are the 16R8, 16RA8, 16V8, 18P8, 20R8, 20RA10, 20V8, 20X10/20L10, 22IP6, 22P10, 22V10, 23S8, 24R10, and 24V10.

Registered Output with /Q Feedback

If an output is configured to be registered, the /Q output of the register can be used to feedback directly into the logic array. To use the /Q output as feedback, no special language syntax is necessary. Simply write the output equation for that I/O pin, then use the I/O pin name in the Boolean expression, as required for other output equations.

Devices Supported: Devices that have registered output with /Q feedback are the 16R8, 16V8, 20R8, 20V8, 20X10/20L10, 22V10, 24R10, and 24V10.

11.3.9 PRELOAD CONTROL

There are two types of preload control.

- Supervoltage
- Product term control

11.3.9.1 Super-voltage

The supervoltage preload control allows any arbitrary state value to be loaded into the registers or latches under supervoltage.⁸ No special language syntax is required for this type of supervoltage-enabled preload.

Devices Supported: Devices that use supervoltage preload are the 16V8, 16V8HD, 20EG8, 20EV8, 20V8, 22V10, 23S8, 24V10, 26V12, 29M16, 29MA16, 30S16, 32VX10, and 610.

11.3.9.2 Product Term Control

The global preload product term is used to control the preload function. To use the preload product term, you must use the language syntax described below.

⁸ Refer to the PAL Device Data Book for details.

Devices Supported: Devices that have both super-voltage enabled and preload product term control are the 29M16 and 29MA16.⁹

Syntax

```
Pin Statement(s) :  
                NODE Global_node_location      Global_node_name  
                :
```

```
Equation(s)     Global_node_name.PRLD = Boolean expression using one product term
```

Example

Preload control with product term

```
:  
CHIP EXAMPLE PALCE29M16  
:  
PIN    2    I1    COMB    ;input  
PIN    11   I2    COMB    ;input  
:  
NODE   1    GLOBAL    ;internal global node  
:  
GLOBAL.PRLD = I1 * /I2    ;preload the registers when (I1 * /I2) is true
```

11.3.10 OBSERVABILITY PRODUCT TERM CONTROL

The global observability product term is used to control the observability function. To use this product term, you must use the language syntax described below.

Devices Supported: Devices that have observability product term control are the 23S8, 29M16, 29MA16, and 30S16.¹⁰

⁹ Refer to 11.4 for specific device syntax datasheets, which provide the assigned node location.

¹⁰ Refer to the individual datasheets for the assigned node location.

Syntax

```
Pin Statement(s) :  
                NODE Observe_node_location   Observe_node_name  
                :
```

```
Equation(s)     Observe_node_name = Boolean expression using one product term
```

Example

```
:  
CHIP EXAMPLE PALCE29M16  
:  
PIN      2    I1    COMB      ;input  
PIN      11   I2    COMB      ;input  
:  
NODE     2    OBSERVE      ;internal global node  
:  
OBSERVE = I1 * /I2          ;observe buried registers when ( I1 * /I2 ) is true
```

11.3.11 COMPLEMENT ARRAY

Complement arrays are used in PLS devices as extra logic resources. To use a complement array, you must first define the output of the complement array as a node in the pin statement, then write the equation for the opposite output polarity. To use the complement array in an equation, you must use the same polarity as defined in the pin statement. The syntax and example for the complement array are described next.

Devices Supported: Devices that have complement arrays are the 105, 167/168, and 30S16.
--

Syntax

```
Pin Statement(s) NODE Complement_array_node_location Complement_array_node_name  
Equation(s)     Complement_array_node_name* =Boolean expression with one product term  
                :  
                < Output equation(s) using either Complement_array_node_name** >
```

* *Complement_array_node_name must have the opposite polarity as defined in the node statement.*

** *The complement_array_node_name used in other output equations must have the same polarity as defined in the node statement.*

Example

Complement array

```
CHIP EXAMPLE PLS167
:
PIN      7    I1    COMB      ;input
PIN      6    I2    COMB      ;input
PIN      5    I3    COMB      ;input
PIN     13    Q3    REG        ;registered output
:
NODE     13    /CA          ;complement array node
:
CA = I1 * /I2              ;write the equation for CA output with opposite polarity
                           ; as defined in the pin statement
Q3.S = /CA * I3            ;use same polarity, /CA as defined in the pin statement
```

11.3.12 ELECTRONIC SIGNATURE

The electronic signature word contains 64 bits of programmable memory that can contain user-defined data. You can program the signature using the Signature statement, shown next.

Devices Supported: Devices that have electronic signature capability are the 16V8, 20V8, and 26V12.
--

Syntax

Pin declaration segment

```
:
SIGNATURE* = < Base(radix) number > or an alphanumeric character string
```

* *The signature can be specified in any of the following formats.*

<u>Base (Radix) Number</u>	<u>Syntax</u>	<u>Max Number of Digits</u>
Binary	#B or #b	64
Decimal (Default)	#D or #d	15
Hexadecimal	#H or #h	16
Octal	#O or #o	21

Example 1

Signature using a base number

SIGNATURE = 144350

Example 2

Signature using an alphanumeric character string

SIGNATURE = V12_6

11.4 PLD DEVICE SYNTAX DATASHEETS

The information here is divided into datasheets for the following non-standard programmable devices.¹¹

- 105
- 167/168
- 16RA8
- 16V8HD
- 20EG8
- 20EV8
- 20RA10
- 22IP6
- 22V10
- 23S8
- 26V12
- 29M16
- 29MA16
- 30S16
- 32VX10
- 610

Each datasheet consists of the following information for a particular device.

- Pin and Node Descriptions
- Block and Macrocell Diagrams
- Special Programming Features

11.4.1 PIN AND NODE DESCRIPTIONS

The pin and node descriptions provide the locations and names for each pin and node in the specified device. This information is needed to program specific features, such as global preset and reset, preload with product term control, observability, and feedback with buried nodes.

¹¹ Standard devices are not listed here and are discussed only in 11.3. MACH devices are discussed in 11.5.

11.4.2 BLOCK AND MACROCELL DIAGRAM(S)

The block diagram shows the relationship between the macrocells and pins, and the locations of the macrocells. The macrocell diagram(s), if any, show logic and fuse information of each type of macrocell, as well as node locations.

11.4.3 SPECIAL PROGRAMMING FEATURES

These discussions identify the language syntax required to program each special feature not covered in the earlier general language syntax discussion. Information here is organized into subtopics that cover each special programming feature. The corresponding language syntax and an example that illustrates its use are also included.

PLS105: PLS105 / PLSCE105

PIN AND NODE DESCRIPTIONS

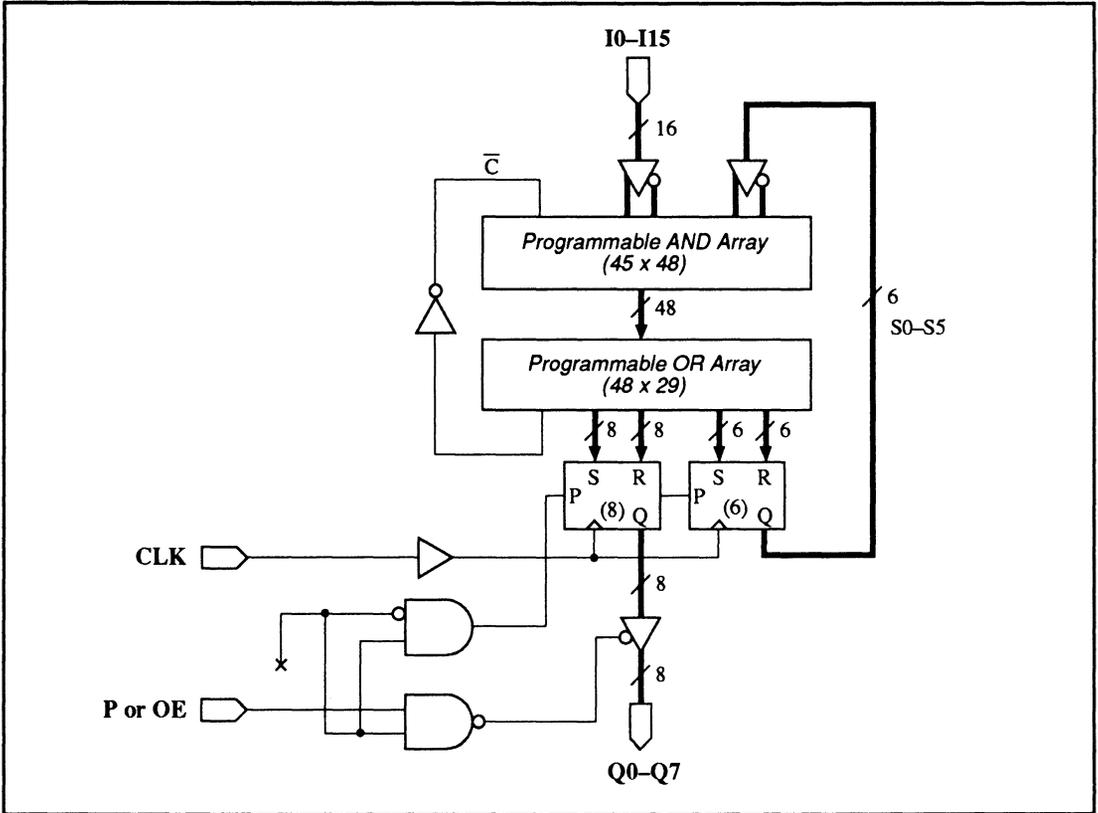
- 28 pins
- 6 buried nodes
- 1 complement array node name

105

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK	-	-	-
2 - 9	I7 - I0	-	-	-
10 - 13	Q7 - Q4	-	-	-
14	GND	-	-	-
15 - 18	Q3 - Q0	-	-	-
19	P or OE	-	-	-
20 - 27	I15 - I8	-	-	-
28	VCC	-	-	-
-	-	1 - 6	S0 - S5	Buried feedback
-	-	7	/C	Complement array

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



105: Block Diagram Showing Pin and Node Locations

SPECIAL PROGRAMMING FEATURES

- Programmable preset/output-enable pin
- State bits

Programmable Pre-set/Output Enable Pin

Pin 19 controls either preset or output enable for the entire device. You define the global function for pin 19 by writing either a .TRST or a .SETF functional equation for one or more outputs. If you write equations for more than one output, each must contain the same sequence of literals and operators.

105: PLS105 / PLSCE105

To eliminate confusion and reduce errors, it is a good idea to write functional equations for a group of vectors rather than for each pin; the preset or enable function controls all output pins.

Syntax 1

Using an output vector

```
Pin Statement(s) :
    PIN Input_pin_location      Input_pin_name
    PIN < Output_pin_number(s) > Output_vector_name
    :
Equation(s) using Pin 19 to control output enable
    Output_vector_name.TRST = Input_pin_name
or using Pin 19 to control preset
    Output_vector_name.SETF = Input_pin_name
```

Example 1

Output enable using an output vector

```
:
PIN    19    OE                ;output-enable control input
PIN    10    13, 15.18 Q[7..0] ;output vector, Q[7..0]
:
Q[7..0].TRST = OE                ;pin OE controls the buffers of outputs Q7 to Q0
```

Syntax 2

Using group outputs

```
Pin Statement(s) :
    PIN Input_pin_location      Input_pin_name
    :
    GROUP Group_name < Output_pin/Node_name(s) >
Equation(s) using Pin 19 to control output enable
    Group_pin_name.TRST = Input_pin_name
or using Pin 19 to control preset
    Group_pin_name.SETF = Input_pin_name
```

Example 2

Preset enable using group outputs

```

:
PIN    19  PRESET                ;preset input
PIN    18  Q7    REG              ;output
:
PIN    10  Q0    REG              ;output
:
NODE   1    S0                    ;buried node
:
NODE   6          S5              ;buried node
GROUP OUTPUTS
    Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
    S5 S4 S3 S2 S1 S0            ;group all outputs and buried registers as OUTPUTS
:
OUTPUTS.SETF = PRESET            ;pin PRESET controls the preset of output registers
;    Q7 to Q0

```

State Bits

Device 105 has six buried-state registers where bits can be stored: S0 to S5. Do not assign names to the buried-state registers when you want to assign state bits automatically.

167/168: PLS167 / PLSCE167 / PLS168 / PLSCE168

PIN AND NODE DESCRIPTIONS

- 24 pins
- 6 buried nodes
- 1 complement array node name

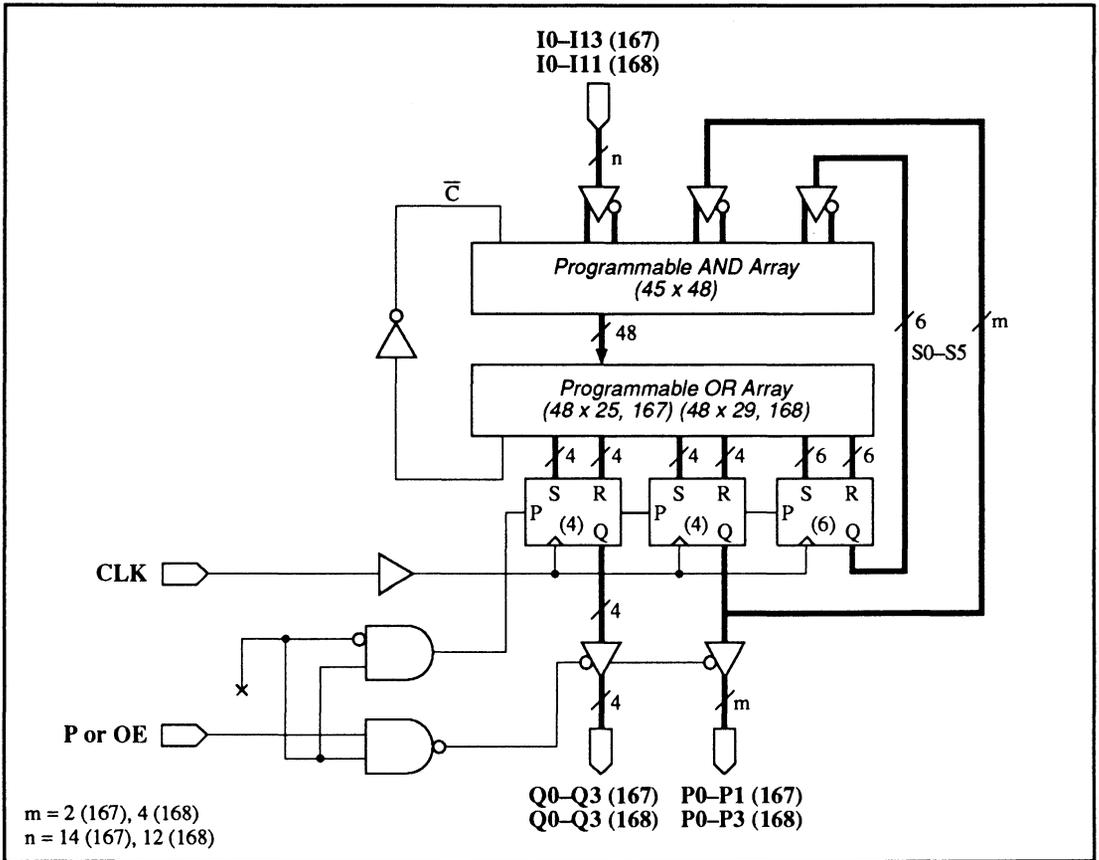
167/168

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK	-	-	-
2 - 8	I6 - I0	-	-	-
9 - 11	Q0 - Q2	-	-	-
12	GND	-	-	-
13	Q3	-	-	-
14 - 15	P0 - P1	-	-	-
16	P or OE	-	-	-
17 - 23	I13 - I7	-	-	-
24	VCC	1 - 6	S0 - S5	Buried feedbacks
-	-	7	/C	Complement array

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

167/168: PLS167 / PLSCE167 / PLS168 / PLSCE168



167/168: Block Diagram Showing Pin and Node Locations

167/168: PLS167 / PLSCE167 / PLS168 / PLSCE168

SPECIAL PROGRAMMING FEATURES

- Programmable preset/output-enable pin
- State bits

Programmable Pre-set/Output-Enable Pin

Pin 16, P or OE, controls either the preset or output enable globally. You define the function for pin 16 by writing either a .TRST or a .SETF functional equation for one or more outputs. If you write equations for more than one output, each must contain the same sequence of literals and operators.

To eliminate confusion and reduce errors, it's a good idea to write functional equations for a group of vectors instead of for each pin; since the preset or enable function controls all output pins.

Syntax 1

Programmable output enable/preset using an output vector

```
Pin Statement(s) :
    PIN    Input_pin_location      Input_pin_name
    PIN < Output_vector_number(s) > Output_vector_name  Storage_type
    :
```

Equation(s) *using Pin 16 to control output enable*
 Output_vector_name.TRST = Input_pin_name
or using Pin 16 to control preset
 Output_vector_name.SETF = Input_pin_name

Example 1

Output enable using an output vector

```
:
PIN    16    OE                      ;output-enable input
PIN    14    15, 9..11  Q[0..7]      ;output vector, Q[0..7]
:
Q[0..7].TRST = OE                    ;pin OE controls the outputs Q0 to Q7
```

167/168: PLS167, PLSCE167, PLS168, PLSCE168

Syntax 2

Programmable output enable/preset using group outputs

```
Pin Statement(s) :
    PIN Input_pin_location      Input_pin_name
    :
    GROUP Group_name    < Output_pin/node_name(s) >
```

Equation(s) using Pin 16 to control output enable
Group_pin_name.TRST = Input_pin_name

or using Pin 16 to control preset
Group_pin_name.SETF = Input_pin_name

Example 2

Preset using group outputs

```
:
PIN    16  PRESET          ;preset input
PIN    9   Q0    REG      ;output
:
PIN    11  Q2    REG      ;output
PIN    14  P0    REG      ;output
PIN    15  P1    REG      ;output
:
NODE    1   S0    REG      ;buried node
:
NODE    6   S5    REG      ;buried node
GROUP OUTPUTS
    P0 P1  Q2  Q1  Q0
    S5 S4  S3  S2  S1  S0      ;group all outputs and buried registers as OUTPUTS
:
OUTPUTS.SETF = PRESET        ;pin PRESET controls the preset of output registers Q2
                             ; to Q0, P0 and P1, and buried registers S5 to S0
```

State Bits

Devices 167/168 have six buried state registers for storing bits: S0 to S5. Do not assign names to the buried-state registers when you want to assign state bits automatically.

16RA8: PAL16RA8

PIN AND NODE DESCRIPTIONS

- 20 pins
- No internal nodes

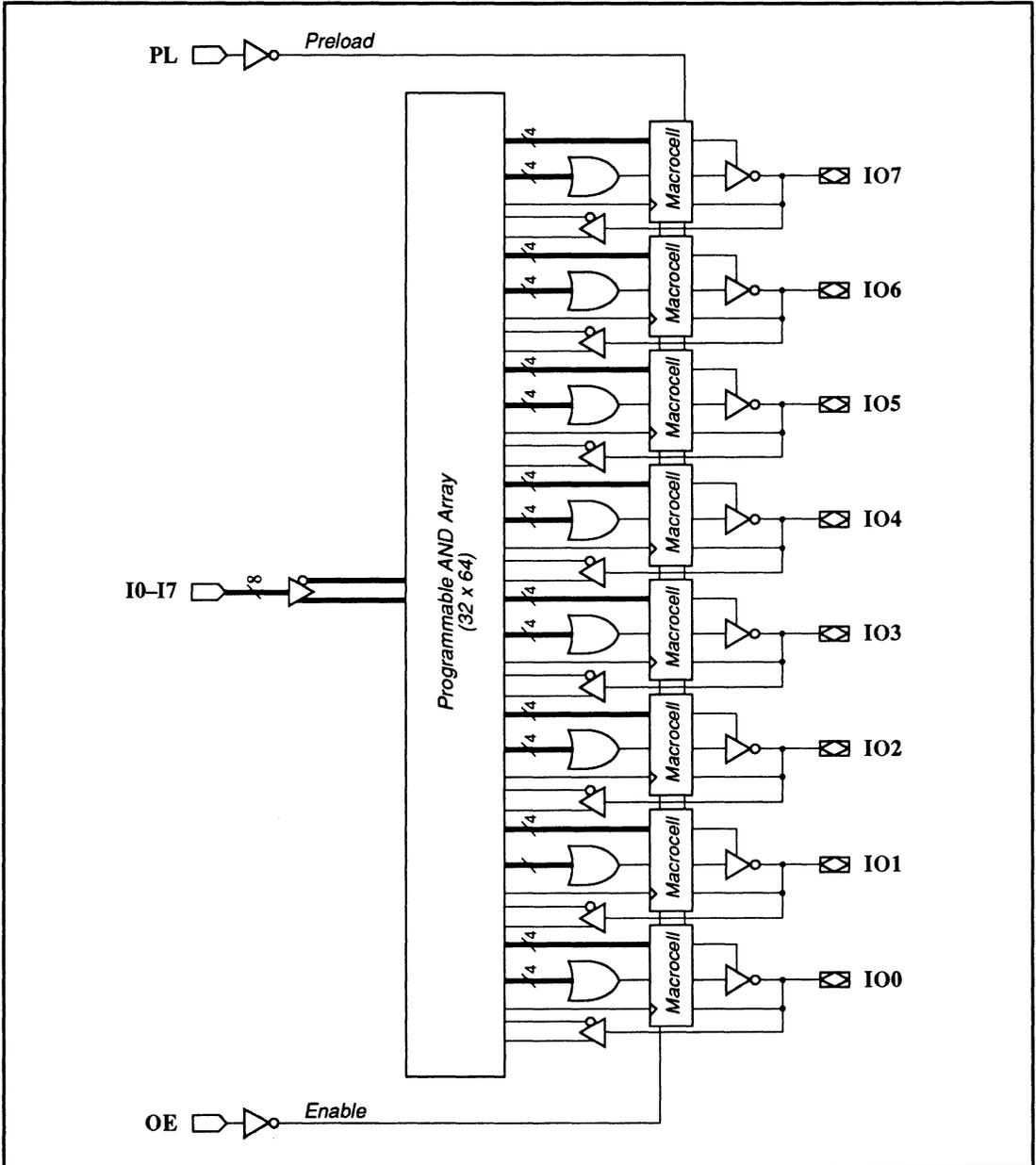
16RA8

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	PL	—	—	—
2 – 9	I0 – I7	—	—	—
10	GND	—	—	—
11	OE	—	—	—
12 – 19	IO0 – IO7	—	—	—
20	VCC	—	—	—

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

16RA8: PAL16RA8



16RA8: Block Diagram Showing Pin and Macrocell Locations

16RA8: PAL16RA8

To program the product term, you write a .TRST equation for the corresponding output. Otherwise, just control the output buffer using the external output-enable pin. To program the individual product term use the syntax below.

Syntax

<i>Pin Statement(s)</i>	:	PIN I/O_pin_location Output_pin_name Storage_type
	:	
<i>Equations)</i>	:	I/O_pin_name .TRST = Boolean expression with one product term

Example

```
:
PIN    11    OE           ;output enable input
PIN    2     IO           ;input
PIN    3     I1           ;input
PIN    12    I00    COMB  ;output, combinatorial
:
I00.TRST = I0 * I1       ;output buffer I00 is only enabled if OE is LOW and
                        ;   (I0*I1) is HIGH
```

External Preload Control

Register preload is controlled by a TTL-level signal through an external preload pin, pin 1. No special language syntax is required.

16V8HD: PAL16V8HD

PIN AND NODE DESCRIPTIONS

- * 24 Pins
- * 16 Buried Nodes

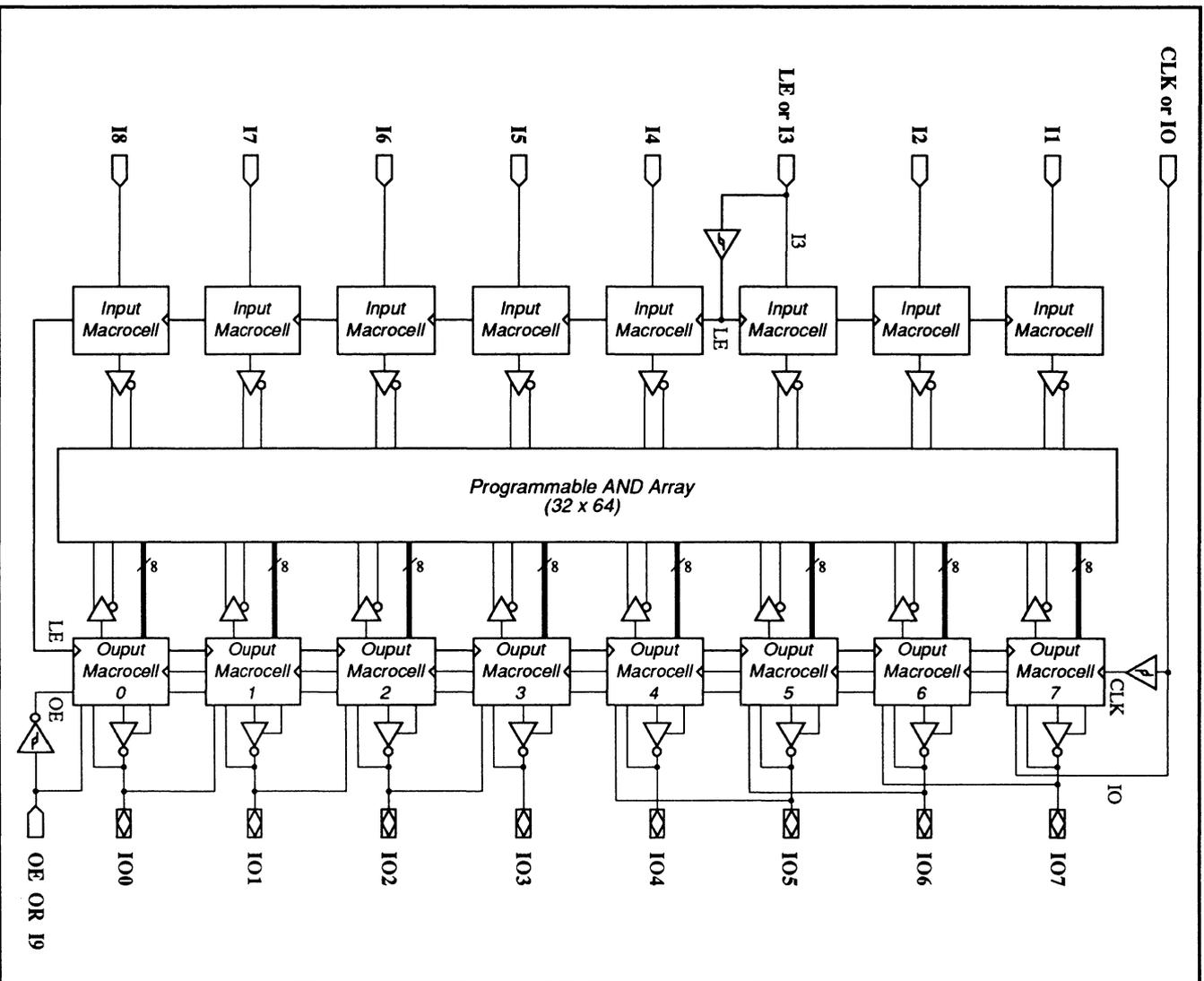
16V8HD

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK or I0	-	-	-
2 - 3	I1 - I2	1 - 2	IBN1 - IBN2	Buried nodes for input latches
4	LE or I3	3	IBN3	Buried nodes for input latches
5 - 9	I4 - I8	4 - 8	IBN4 - IBN8	Buried nodes for input latches
10	OE or I9	-	-	-
11	GND	-	-	-
12	VCC	-	-	-
13 - 16	IO0 - IO3	9 - 12	IOBN0 - IOBN3	Buried nodes for feedback latches
17	GND	-	-	-
18	VCC	-	-	-
19 - 20	IO4 - IO5	13 - 14	IOBN4 - IOBN5	Buried nodes for feedback latches
21	GND	-	-	-
22 - 23	IO6 - IO7	15 - 16	IOBN6 - IOBN7	Buried nodes for feedback latches
24	VCC	-	-	-

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

16V8HD: PAL16V8HD



16V8HD: Block Diagram

SPECIAL PROGRAMMING FEATURES

- * Latch and clock controls
- * Output with latched feedback
- * Feedback with latched input
- * Input latch
- * Output buffer with open collector output

Latch and Clock Controls

The clocks of all output registers are controlled by a dedicated clock input, pin 1, which can also be used as an input. The latch enable inputs of all input and feedback latches are controlled by a dedicated latch input, pin 4, which can also be used as an input. To use pin 1 as clock control, write a .CLKF equation for any I/O pin. To use pin 4 as latch control, write a .CLKF equation for any latch node name, as shown below.

Syntax 1

For clock control

```
Pin Statement(s) :
    PIN 1 Clock_input_name
    PIN I/O_pin_number    I/O_pin_name    REG
    :
```

```
Equation(s)      I/O_pin_name.CLKF = Clock_input_name
```

Syntax 2

For latch control

```
Pin Statement(s) PIN 4 Latch_enable_name
    Node Buried_node_number    Buried_node_name    LAT
    :
```

```
Equation(s)      I/O_pin_name.CLKF = Latch_enable_name
```

Note: If you do not include any .CLKF equations, then the clock pin is routed to all registers by default and the latch enable pin is routed to all latches by default.

Note: If you write no .CLKF equations, the clock pin is routed to all registers by default and the latch enable pin is routed to all latches by default.

16V8HD: PAL16V8HD

Example

Latch and clock control

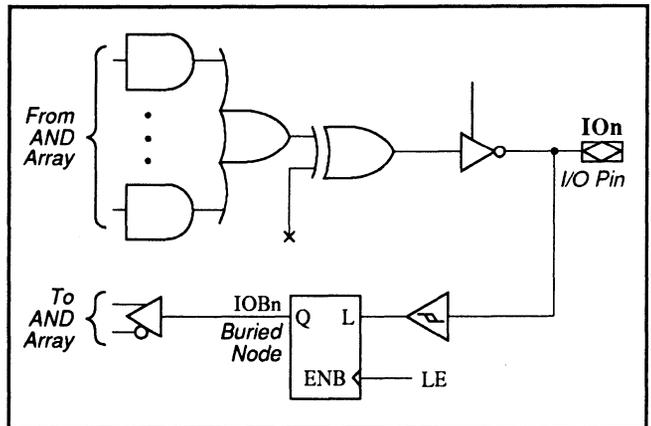
```

:
PIN    1    Clock                ;clock input
PIN    4    LE                    ;latch enable input
PIN    13   IOO    REG           ;output, registered
:
NODE    9    IOBNO LAT           ;buried node of feedback latch IOO
:
IOO.CLKF = Clock                ;assign clock input
IOBNO.CLKF = LE                  ;assign latch enable input

```

Output with Latched Feedback

For each output macrocell, the feedback from each I/O can be programmed to be a latched input that feeds back to the AND array. Each output macrocell can be configured to have either combinatorial or registered output with latched feedback. The PALASM syntax for both cases is shown below.



16V8HD: Combinatorial Output with Latched Feedback

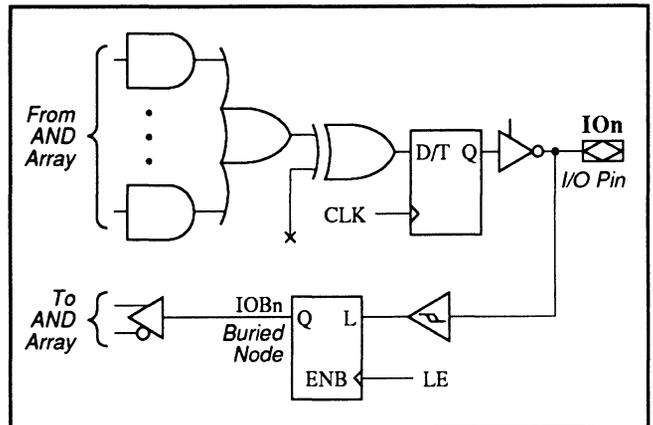
Syntax

For combinatorial output with latched feedback

```

Pin Statement(s) :
    PIN I/O_pin_number    I/O_pin_name    COMB
    :
    NODE Buried_node_number    Buried_node_name    LAT
    :
Equation(s)      I/O_pin_name = Boolean expression
                  Buried_node_name = I/O_pin_name*
    :
                  < Output equation(s) using Buried_node_name as feedback >
    
```

* *I/O_pin_name must use the same polarity as defined in the pin statement.*



16V8HD: Registered Output with Latched Feedback

16V8HD: PAL16V8HD

Syntax

For registered output with latched feedback

```
Pin Statement(s)  PIN  I/O_pin_number  I/O_pin_name  REG
                  :
                  NODE  Buried_node_number  Buried_node_name  LAT
                  :
Equation(s)       for D flip-flop
                  I/O_pin_name = Boolean expression
                  for T flip-flop
                  I/O_pin_name.T = Boolean expression
                  :
                  Buried_node_name = I/O_pin_name*
                  :
                  < Output equation(s) using Buried_node_name as feedback >
```

* I/O_pin_name must use the same polarity as defined in the pin statement.

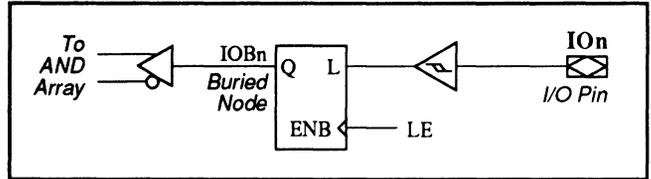
Example

Output with latched feedback

```
:
PIN    2    I1           ;input, combinatorial
PIN    3    I2           ;input, combinatorial
PIN    13   I00  REG     ;I/O, registered
PIN    14   /I01  COMB   ;I/O, combinatorial
PIN    15   I02  COMB   ;I/O, combinatorial
:
NODE   9    IOBNO  LAT    ;buried node, feedback latch
NODE   10   IOBN1  LAT    ;buried node, feedback latch
:
I00 = I1 * I2           ;output equation for I00
I01 = I1 * /I2          ;output equation for I01
IOBNO = I00             ;assign node IOBNO to latch the feedback from I00
IOBN1 = /I01           ;assign node IOBN1 to latch the feedback from /I01
```

Feedback with Latched Input

Each output macrocell can be configured to be used just as an input, with no output. The PALASM syntax for this configuration is shown below.



16V8HD: Feedback with Latched Input Only

Syntax

```
Pin Statement(s)  PIN  I/O_pin_number  I/O_pin_name
                  :
                  NODE Buried_node_number  Buried_node_name  LAT
                  :
```

```
Equation(s)      Buried_node_name = I/O_pin_name*
                  :
                  < Output equation(s) using Buried_node_name as input >
```

* *I/O_pin_name must use the same polarity as defined in the pin statement.*

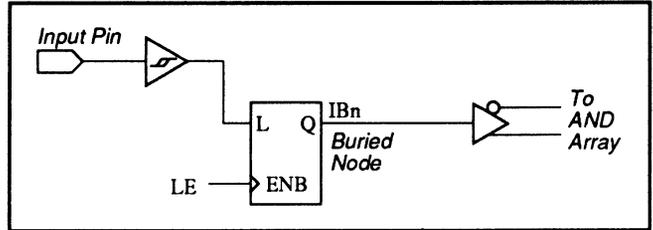
Example

```
:
PIN    2    I1                ;input, combinatorial
PIN    3    I2                ;input, combinatorial
PIN    13   IO0               ;input only
PIN    15   IO2    COMB       ;I/O, combinatorial
:
NODE    9    IOBNO  LAT        ;buried node, feedback latch
:
IOBNO = IO0                    ;assign node IOBNO to IO0
IO2 = I1 * /I2 * /IOBNO        ;output equation for IO2 using IOBNO as input
```

16V8HD: PAL16V8HD

Input Latch

Each input macrocell can be configured as a latched input. The PALASM syntax for this configuration is shown below.



16V8HD: Input Latch

Syntax

```
Pin Statement(s)  PIN  Input_pin_number  Input_pin_name
                  :
                  NODE Buried_node_number  Buried_node_name  LAT
```

```
Equation(s)      Buried_node_name = Input_pin_name*
                  :
                  Output equation(s) using Buried_node_name as input
```

** Input_pin_name must use the same polarity as defined in the pin statement.*

Example

Input latch

```
:
PIN    2    I1                ;input, combinatorial
PIN    3    I2                ;input, combinatorial
PIN    15   I02  COMB         ;I/O, combinatorial
:
NODE   1    IBN1  LAT         ;buried node, latched
:
IBN1 = I1                    ;assign node IBN1 to input I1
I02 = I1 * IBN1              ;output equation for I02 using node IBN1 as input
```

Output Buffer with Open Collector Output

Each output buffer can be programmed to have an open-collector output by blowing the open-collector fuse, SL5. An 8-bit mask is used to control the eight open-collector fuses of the eight outputs. The MSB of the 8-bit mask controls output IO7, while the LSB controls output IO0, shown below.

Outputs	I07	I01	I02	I03	I04	I05	I06	I00
Eight-bit Mask	X	X	X	X	X	X	X	X
	MSB							LSB

X = 0 or 1
 0 = No Open-collector
 1 = Open-collector

If the bit for the corresponding output is set, that open-collector fuse is blown, resulting in an open-collector output. For example, the mask 00000111 means that IO0 to IO2 are open-collector outputs. In order to use the 8-bit mask to program the outputs, use the PALASM syntax below.

Syntax

```
Declaration segment      :
                          COLLECTOR = < Base(radix) number >
```

<i>Base(radix) number</i>	<i>Syntax</i>	<i>Max. No. of Digits</i>
<i>Binary</i>	#B or #b	8
<i>Decimal</i>	#D or #d	3
<i>Hexadecimal</i>	#H or #h	2
<i>Octal</i>	#O or #o	3

Example

Output buffer with open-collector output

```
:
;Declaration Segment
:
COLLECTOR = #B10100001      ;Outputs IO0, IO5 and IO7 are programmed to be open-
;                            ; collector outputs.
```

20EG8: PAL10H20EG8 / PAL10020EG8

PIN AND NODE DESCRIPTIONS

- 24 pins
- 1 global preset/reset node

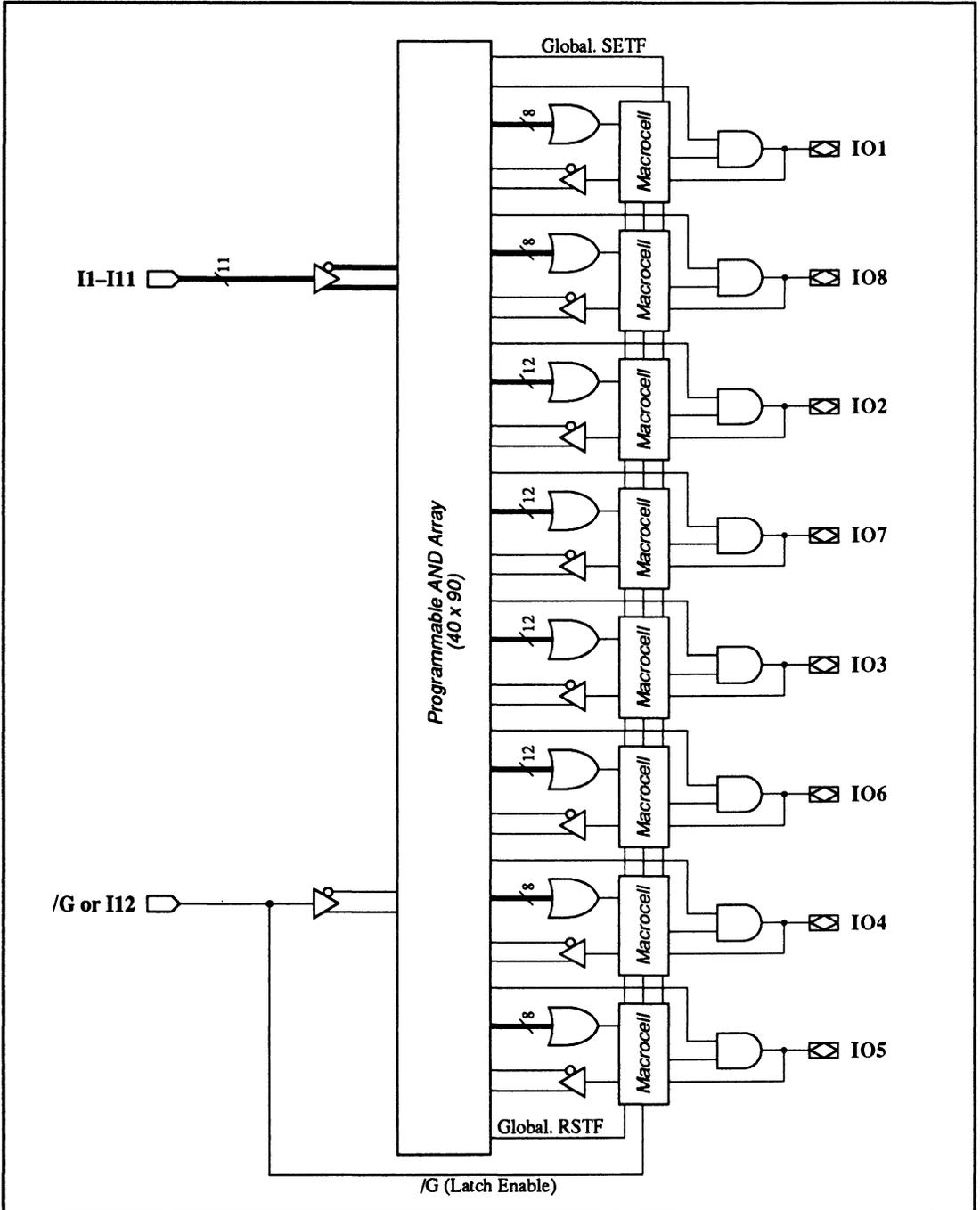
20EG8

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1 - 2	I1 - I2	-	-	-
3	/G or I12	-	-	-
4 - 5	IO1 - I2	-	-	-
6	VCO1	-	-	-
7 - 8	IO3 - I4	-	-	-
9 - 11	I3 - I5	-	-	-
12	VEE	-	-	-
13 - 16	I6 - I9	-	-	-
17 - 18	IO5 - IO6	-	-	-
19	VCO2	-	-	-
20 - 21	IO7 - I8	-	-	-
22 - 23	I10 - I11	-	-	-
24	VCC	-	-	-
-	-	1	GLOBAL	Global preset and reset

BLOCK AND MACROCELL DIAGRAMS

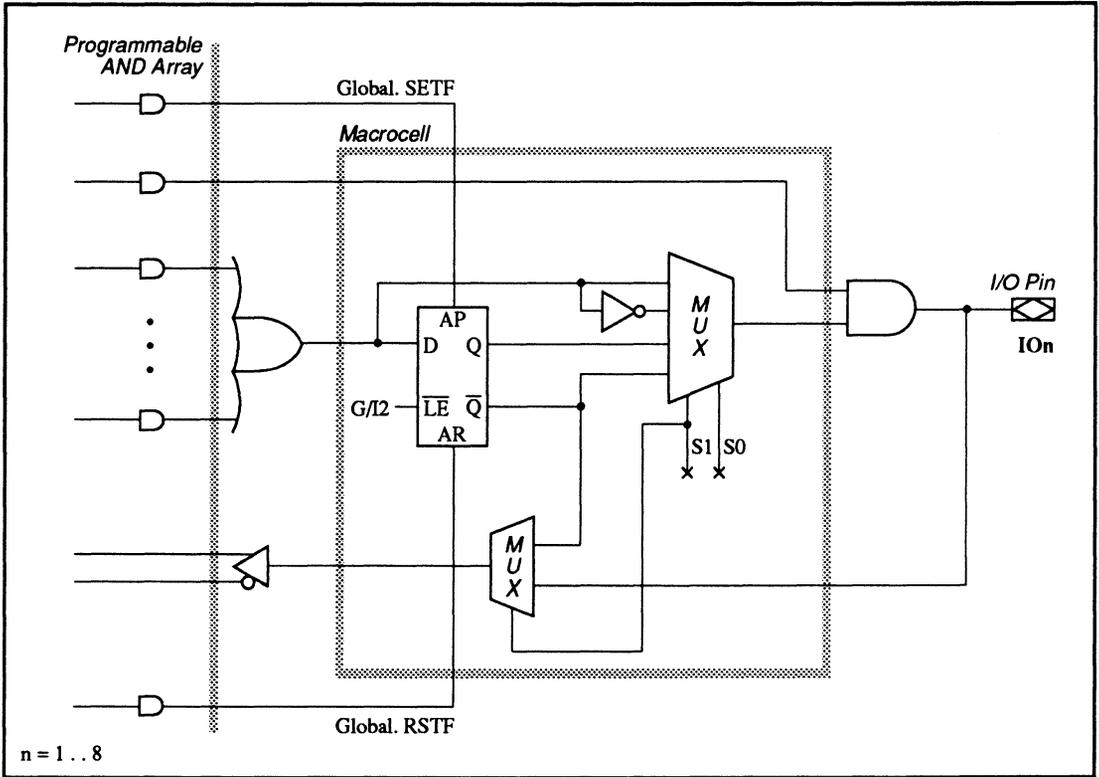
Block and macrocell diagrams follow.

20EG8: PAL10H20EG8 / PAL10020EG8



20EG8: Block Diagram Showing Pin and Node Locations

20EG8: PAL10H20EG8 / PAL10020EG8



20EG8: Macrocell Diagram

20EG8: PAL10H20EG8 / PAL10020EG8

SPECIAL PROGRAMMING FEATURES

These devices have no additional programming features other than those discussed under 11.3.

20EV8: PAL10H20EV8 / PAL10020EV8

PIN AND NODE DESCRIPTIONS

- 24 pins
- 1 global preset/reset node

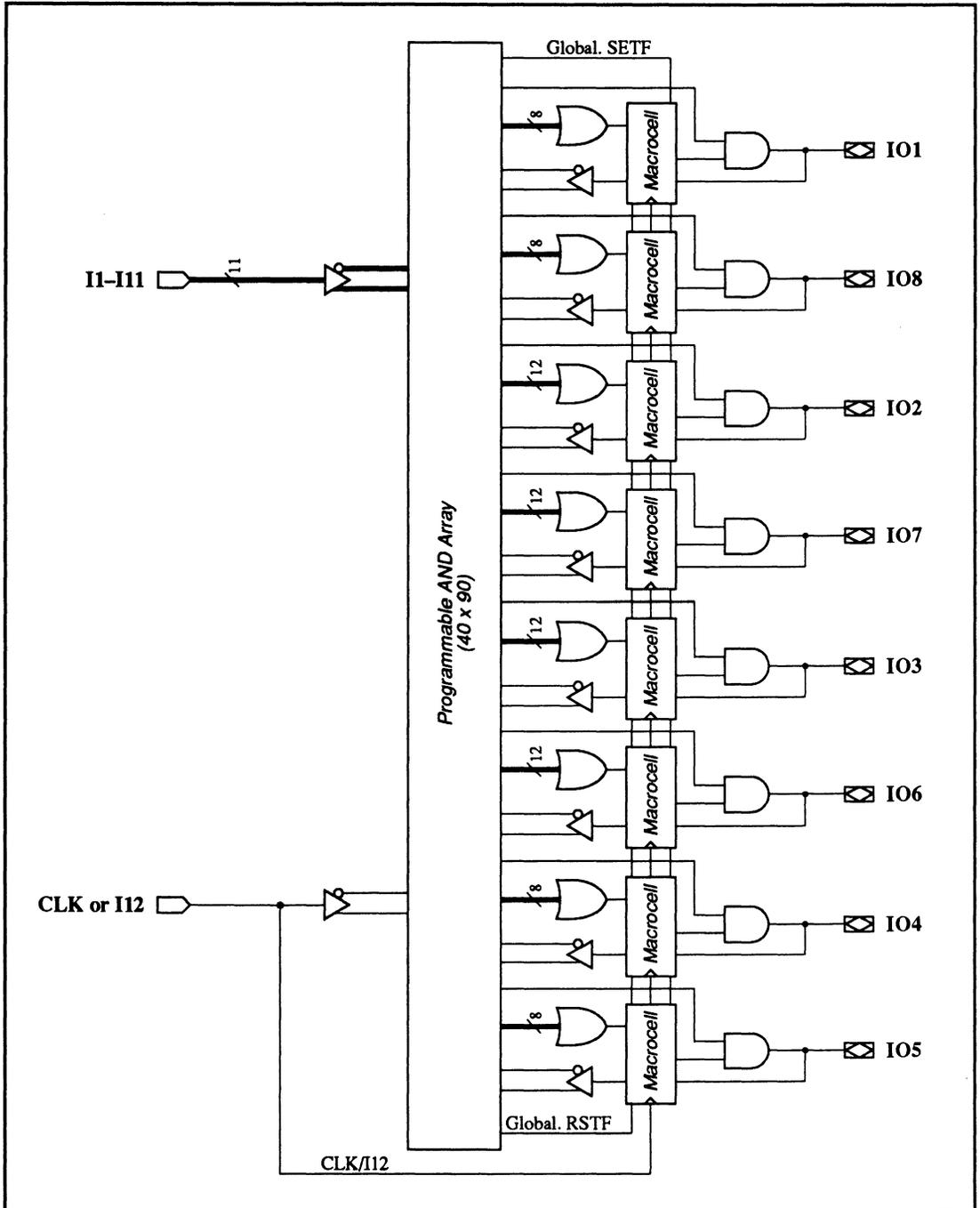
20EV8

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1 - 2	I1 - I2	-	-	-
3	CLK or I12	-	-	-
4 - 5	IO1 - I2	-	-	-
6	VCO1	-	-	-
7 - 8	IO3 - I4	-	-	-
9 - 11	I3 - I5	-	-	-
12	VEE	-	-	-
13 - 16	I6 - I9	-	-	-
17 - 18	IO5 - IO6	-	-	-
19	VCO2	-	-	-
20 - 21	IO7 - I8	-	-	-
22 - 23	I10 - I11	-	-	-
24	VCC	-	-	-
-	-	1	GLOBAL	Global preset and reset

BLOCK AND MACROCELL DIAGRAMS

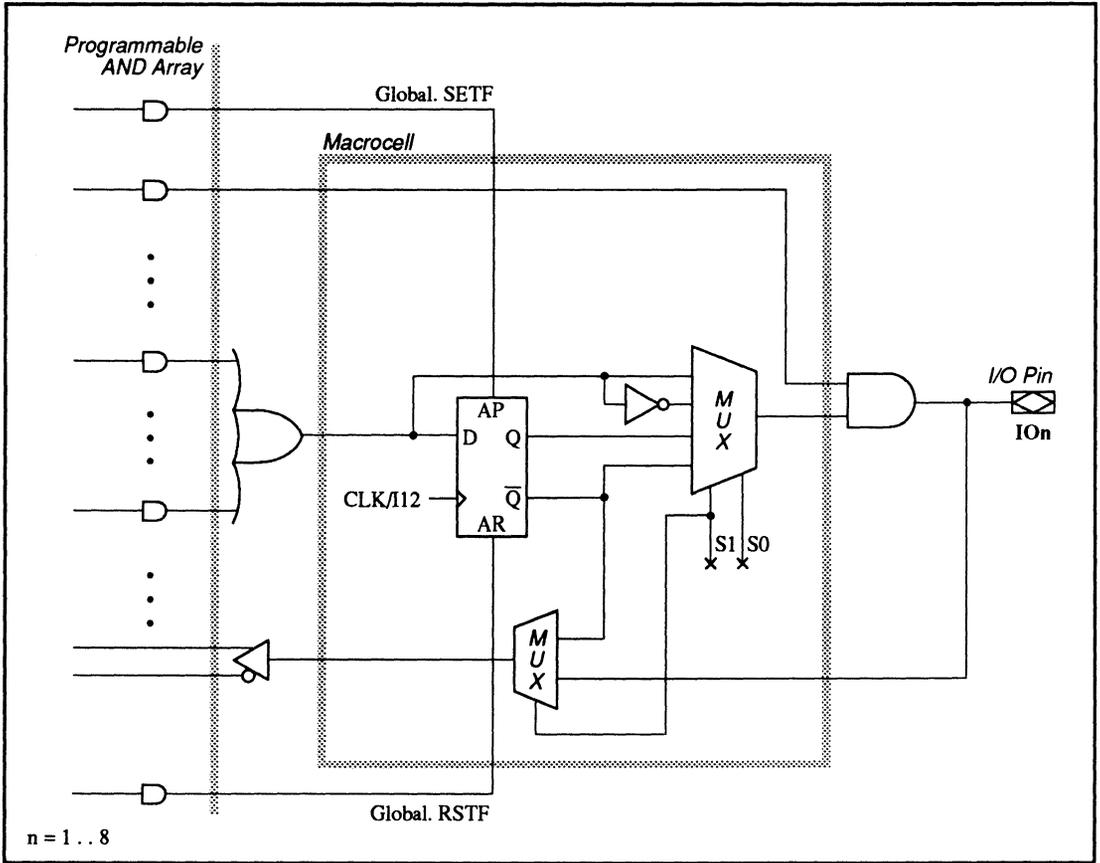
Block and macrocell diagrams follow.

20EV8: PAL10H20EV8 / PAL10020EV8



20EV8: Block Diagram Showing Pin and Node Locations

20EV8: PAL10H20EV8 / PAL10020EV8



20EV8: Macrocell Diagram

20EV8: PAL10H20EV8 / PAL10020EV8

SPECIAL PROGRAMMING FEATURES

These devices have no additional programming features other than those discussed under 11.3.

20RA10: PAL20RA10

PIN AND NODE DESCRIPTIONS

- 20 pins
- No internal nodes

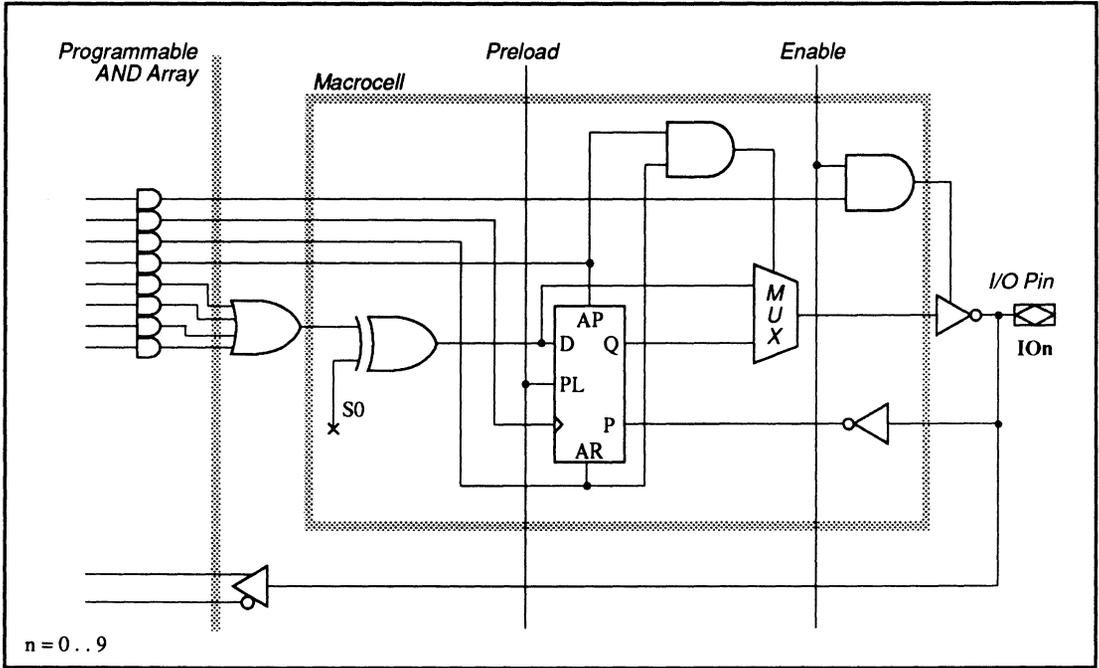
20RA10

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	PL	—	—	—
2 – 11	IO – I9	—	—	—
12	GND	—	—	—
13	OE	—	—	—
14 – 23	IO0 – IO9	—	—	—
24	VCC	—	—	—

BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

20RA10: PAL20RA10



20RA10: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Common external and individual product term output-enable control
- External preload control

Common External and Individual Product Term Output-Enable Control

Each three-state output buffer is controlled by both the common external output-enable pin and an individual product term. If the individual product term is used, an output buffer will be enabled **only if the external output-enable pin is low and the output-enable product term is true.**

To program the product term, you write a .TRST equation for the corresponding output. Otherwise, you can control the output buffer with the external output-enable pin.

Syntax

```
Pin Statement(s) :  
    PIN I/O_pin_location I/O_pin_name Storage_type  
    :  
Equation(s)      I/O_pin_name .TRST = Boolean expression with one product term
```

Example

```
:  
PIN    13    OE                ;output-enable input  
PIN    2     IO                ;input  
PIN    3     I1                ;input  
PIN    14    I00    COMB       ;output, combinatorial  
:  
I00.TRST = IO * I1            ;output buffer I00 is only enabled if OE is LOW and  
;    (IO*I1) is HIGH
```

External Preload Control

Register preload is controlled by a TTL-level signal through an external preload pin, pin 1. No special language syntax is required.

22IP6: PALCE22IP6

PIN AND NODE DESCRIPTIONS

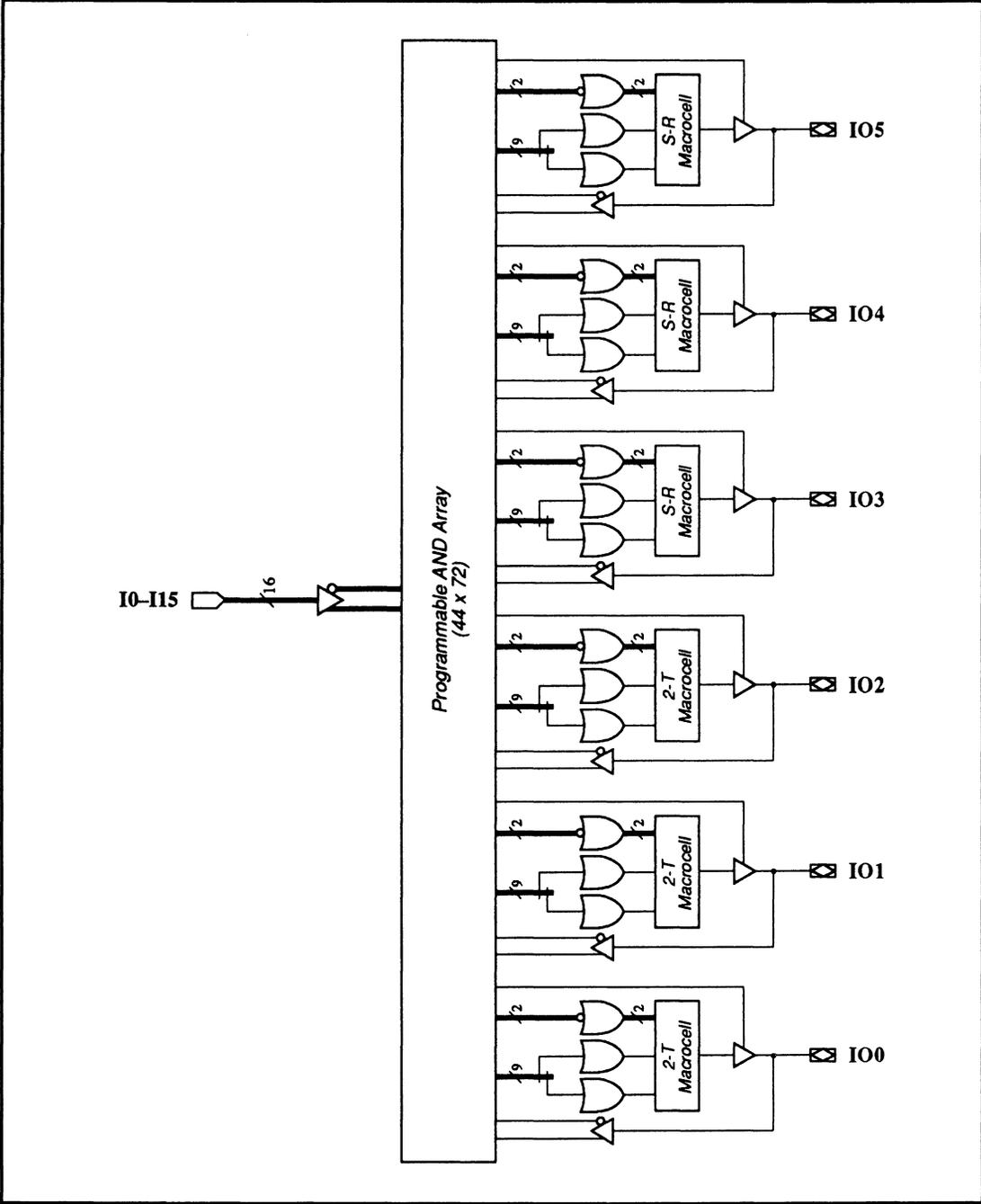
- 24 pins
- No internal nodes

22IP6

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1 - 5	I0 - I4	-	-	-
6	VCC	-	-	-
7 - 12	I5 - I0	-	-	-
13 - 14	I15 - I14	-	-	-
15 - 17	IO5 - IO3	-	-	-
18	GND	-	-	-
19 - 21	IO2 - IO0	-	-	-
22 - 24	I13 - I11	-	-	-

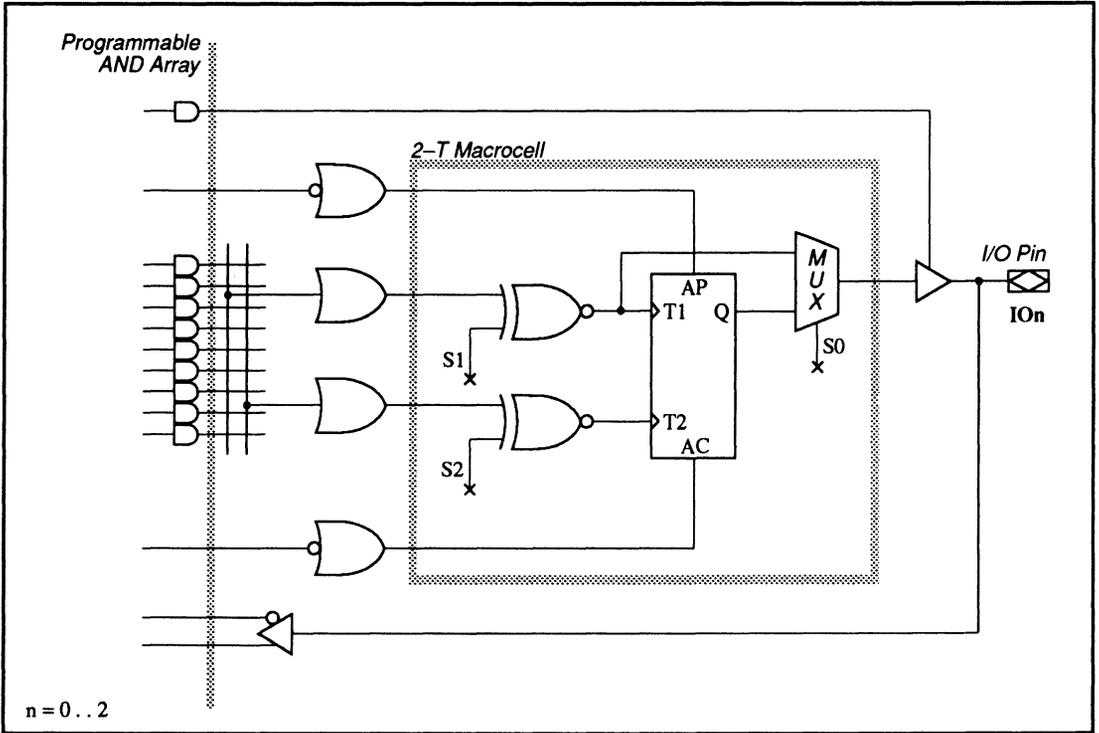
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

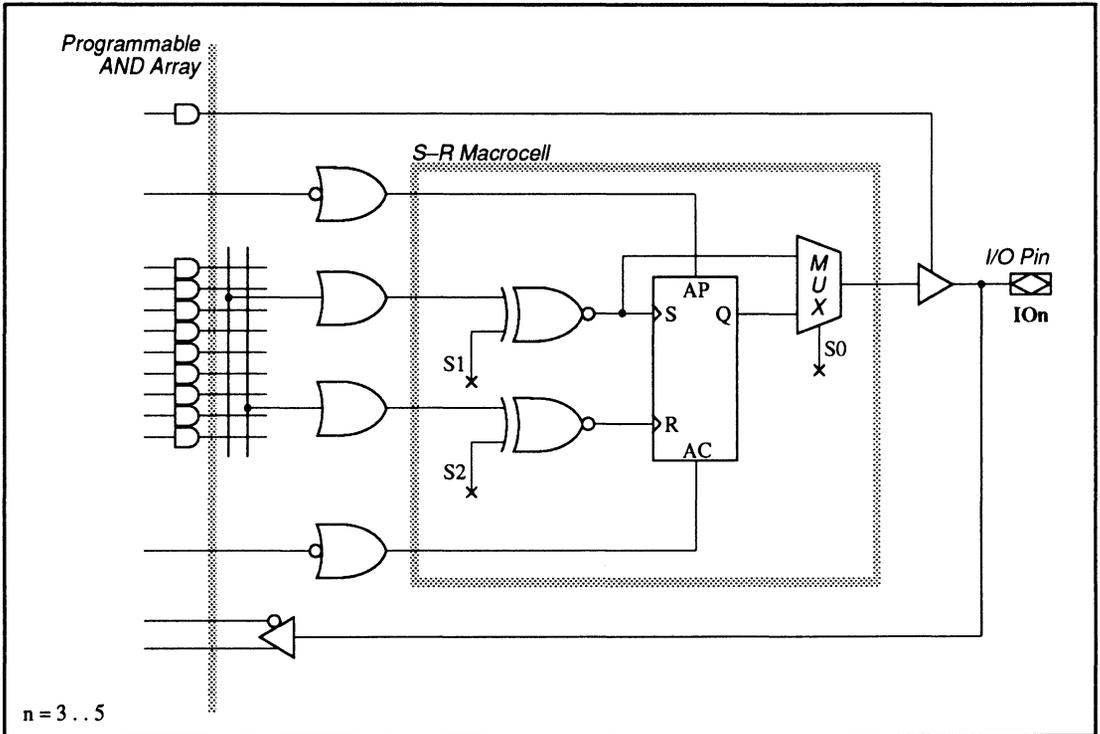


22IP6: Block Diagram Showing Pin and Macrocell Locations

22IP6: PALCE22IP6



22IP6: 2-T Macrocell Diagram



22IP6: S-R Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- 2-T flip-flops with programmable edge-activated input polarity
- SR flip-flops with programmable edge-activated input polarity
- Preset/reset controls with individual sum terms

2-T Flip-Flops

Three of the flip-flops in the 22IP6 are 2-T flip-flops. Each 2-T flip-flop has two independent inputs. Each input can be defined as rising- or falling-edge triggered. The language syntax is shown next.

22IP6: PALCE22IP6

Syntax

<i>Pin Statement(s)</i>	:	PIN	I/O_pin_location	I/O_pin_name	REG
	:				
<i>Equation(s)</i>			$I/O_Pin_name.T1^*$	= Boolean expression	
			$I/O_Pin_name.T2^*$	= Boolean expression	

* Use active-high *output_pin_name* for rising-edge triggered; use active-low *output_pin_name* for falling-edge triggered.

Example

```
:
PIN    1    IO                ;input
PIN    2    I1                ;input
PIN    15   IO5   REG        ;output, registered
:
  IO5.T1 = I1 * I2            ;T1 is asserted when I1 * I2 is high (rising edge)
/I05.T2 = I1 * I2            ;T2 is asserted when I1 * I2 is low (falling edge)
```

SR Flip-Flops

Three of the flip-flops in the 22IP6 are SR flip-flops. Each SR flip-flop has two inputs. Each input can be defined as rising- or falling-edge triggered. The language syntax is shown below.

Syntax

<i>Pin Statement(s)</i>	:	PIN	I/O_pin_location	I/O_pin_name	REG
	:				
<i>Equation(s)</i>			$I/O_Pin_name.S^*$	= Boolean expression	
			$I/O_Pin_name.R^*$	= Boolean expression	

* Use active-high *output_pin_name* for rising-edge triggered; use active-low *output_pin_name* for falling-edge triggered.

Example

```

:
PIN    1   IO           ;input
PIN    2   I1           ;input
PIN    15  IO5  REG     ;output, registered
:
  IO5.S = I1 * I2       ;S is asserted when I1 * I2 is high (rising edge)
  /IO5.R = I1 * I2      ;R is asserted when I1 * I2 is low (falling edge)

```

Preset/Reset Control with Individual Sum Term Syntax

On each flip-flop, each preset and reset function is controlled by a programmable sum term. The language syntax is shown below.

Syntax

```

Pin Statement(s) :
    PIN Output_pin_location  I/O_pin_name  Storage_type
:

```

```

Equation(s) for preset
    Output_Pin_name.SETF = Boolean expression using one sum term
                        or NAND product term

for reset
    Output_Pin_name.RSTF = Boolean expression using one sum term
                        or NAND product term

```

Example

```

:
PIN    1   IO           ;input
PIN    2   I1           ;input
PIN    15  IO5  REG     ;output, registered
:
  IO5.SETF = I1 + /I2    ;set is asserted when I1 + I2 is true
  IO5.RSTF = /(I1 * I2) ;reset is asserted when /(I1* I2) is true

```

22V10: PAL22V10 / AMPAL22V10 / PALCE22V10

PIN AND NODE DESCRIPTIONS

- 24 pins
- 1 global preset and reset node

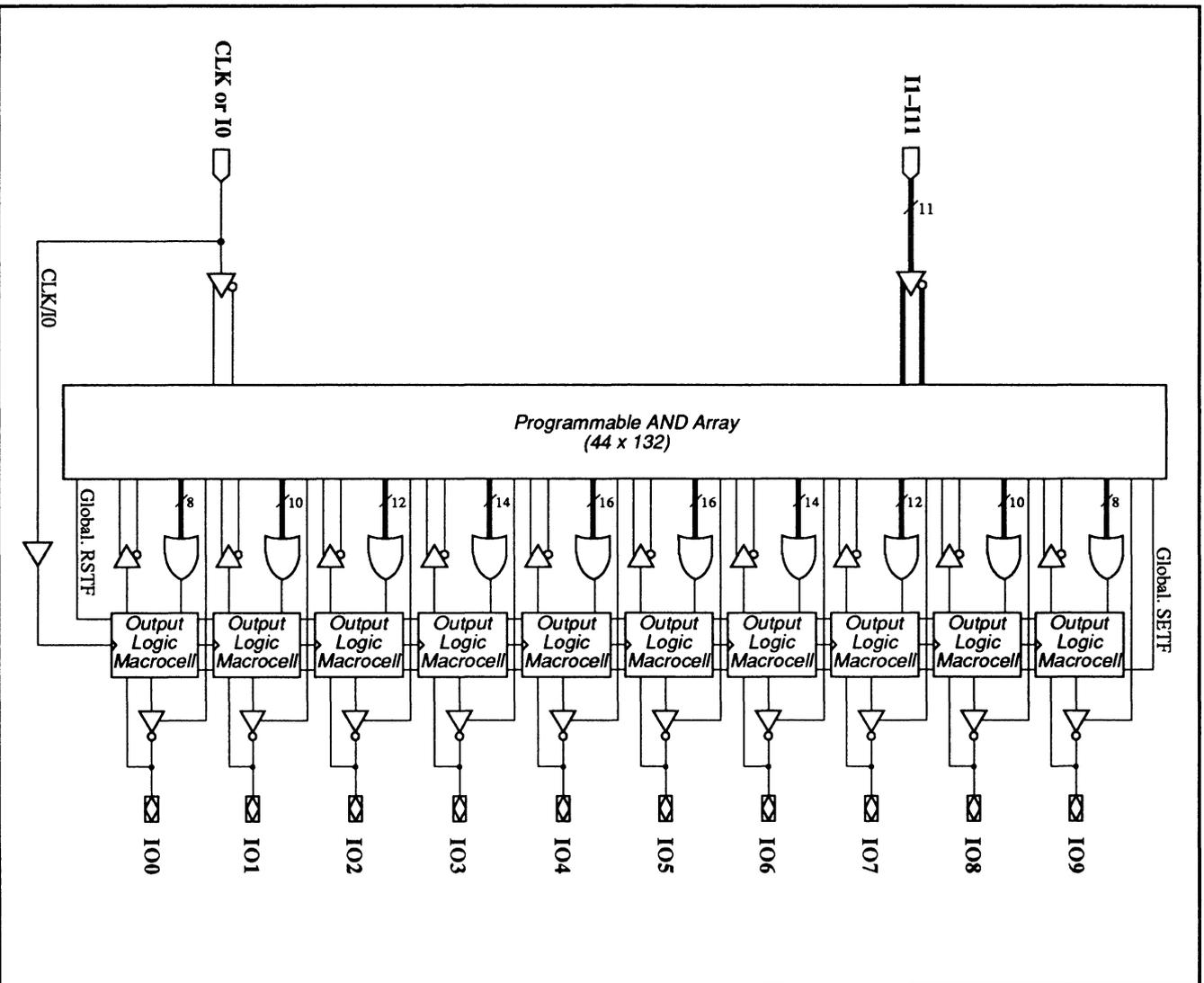
22V10

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK or I0	-	-	-
2 - 11	I1 - I10	-	-	-
12	GND	-	-	-
13	I1	-	-	-
14 - 23	IO0 - IO9	-	-	-
24	VCC	-	-	-
-	-	1	GLOBAL	Global preset and reset

BLOCK AND MACROCELL DIAGRAMS

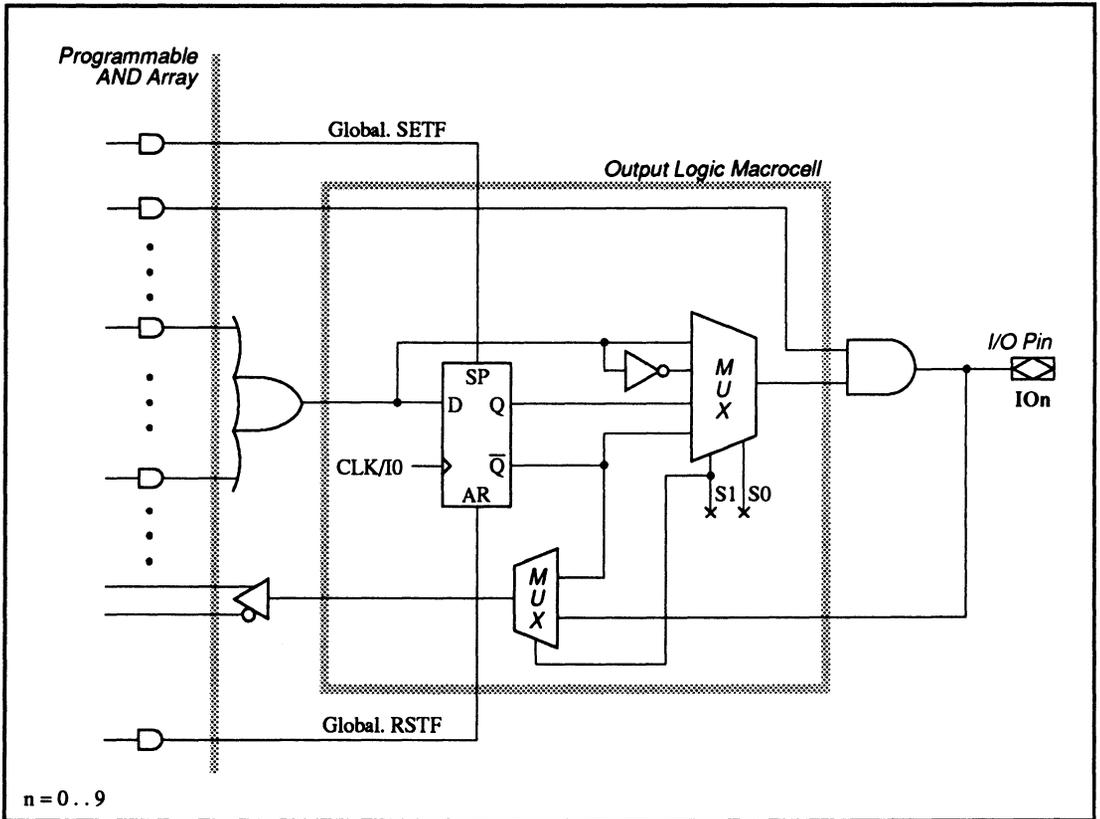
Block and macrocell diagrams follow.

22V10: PAL22V10 / AMPAL22V10 / PALCE22V10



22V10: Block Diagram Showing Pin and Node Locations

22V10: PAL22V10 / AMPAL22V10 / PALCE22V10



22V10: Macrocell Diagram

22V10: PAL22V10 / AMPAL22V10 / PALCE22V10

SPECIAL PROGRAMMING FEATURES

These devices have no additional programming features other than those discussed under 11.3.

23S8: PALCE23S8

PIN AND NODE DESCRIPTIONS

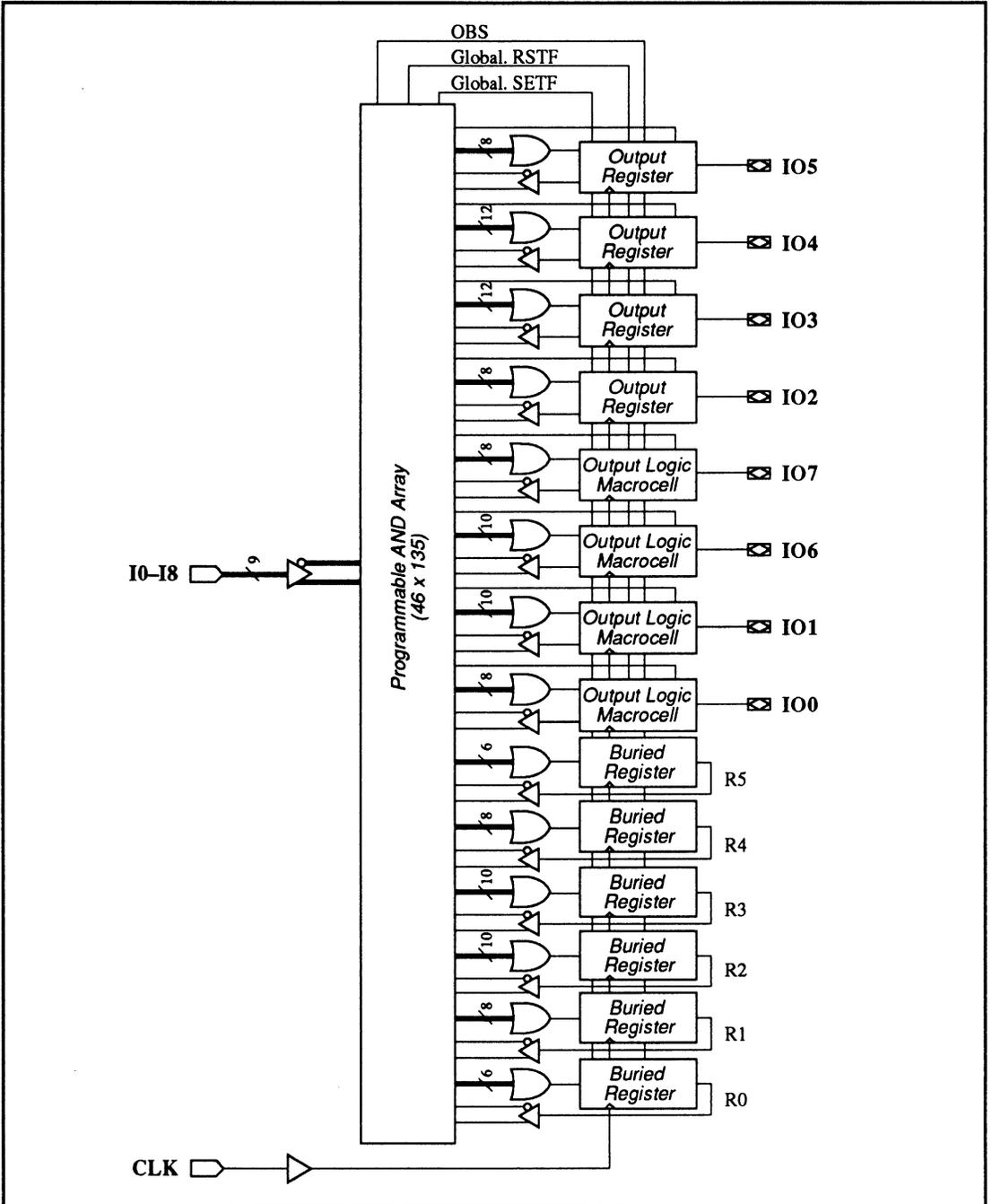
- 20 pins
- 1 global preset and reset node
- 1 observability node
- 10 buried nodes

23S8

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK	-	-	-
2 - 9	I1 - I7	-	-	-
10	GND	-	-	-
11	I8	-	-	-
12 - 13	IO0 - IO1	3 - 4	RIO0 - RIO1	Register feedbacks
14 - 17	IO2 - IO5	-	-	-
18 - 19	IO6 - IO7	11 - 12	RIO6 - RIO7	Register feedbacks
20	VCC	-	-	-
-	-	1	GLOBAL	Global preset and reset
-	-	2	OBS	Observability
-	-	5 - 10	R0 - R5	Buried registers

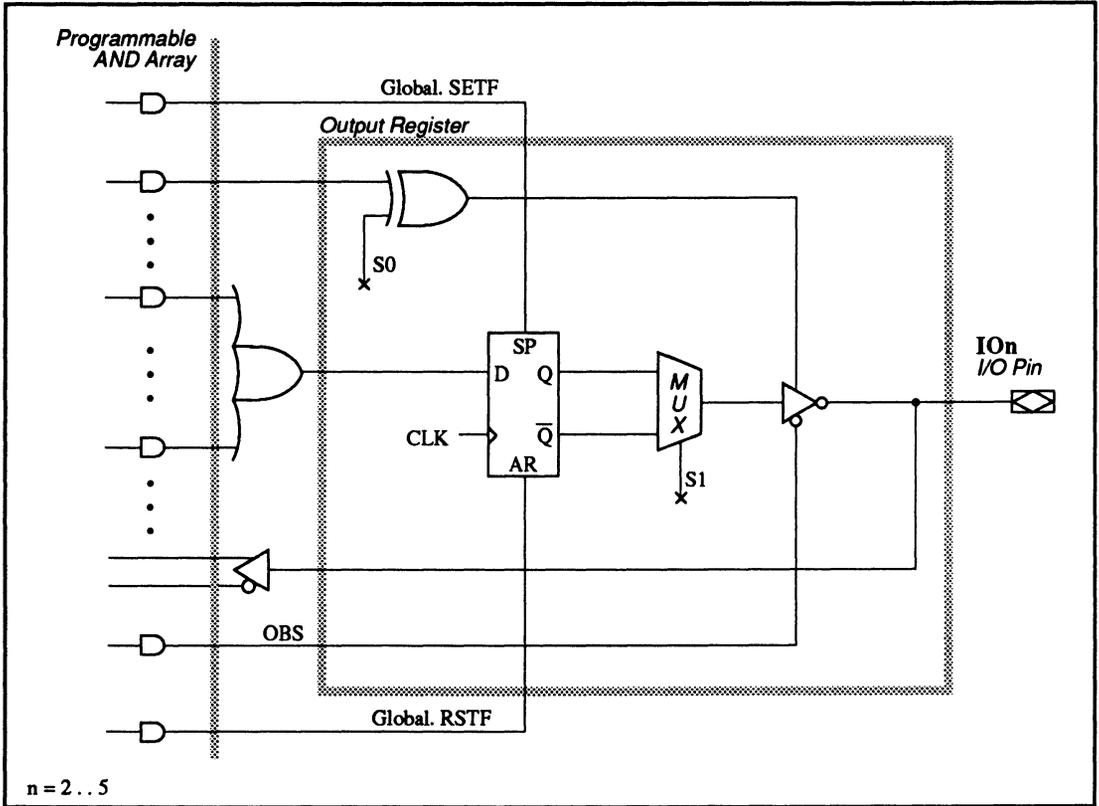
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

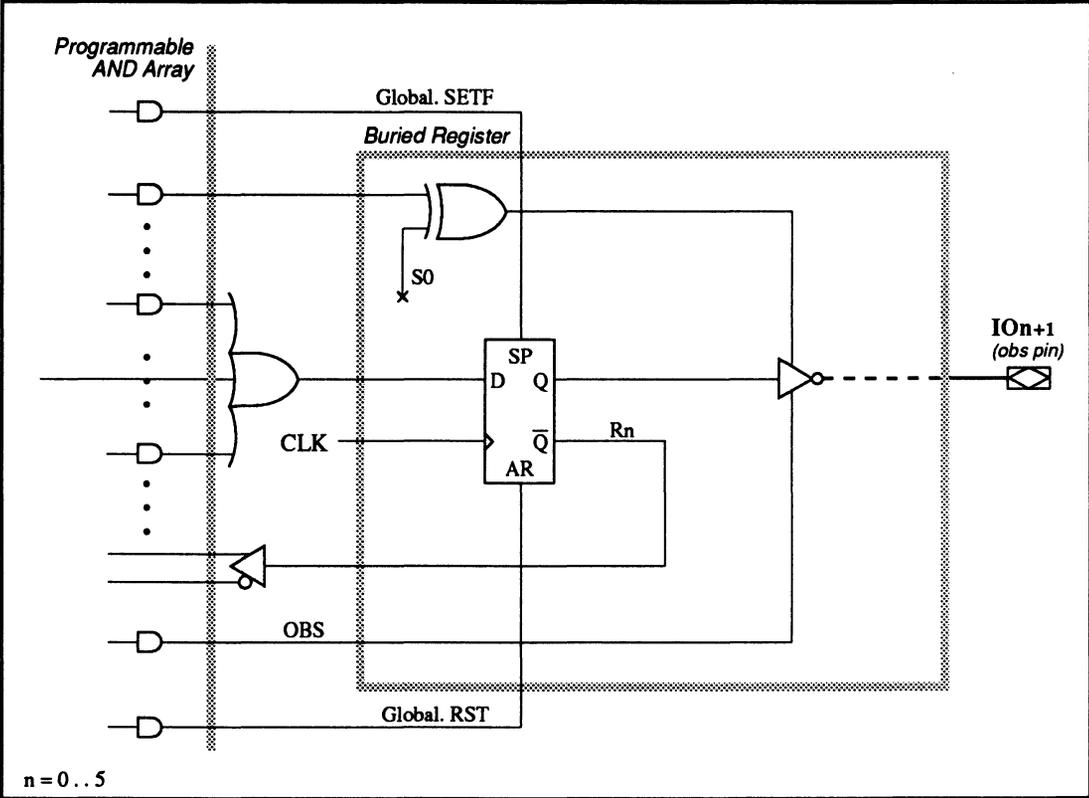


23S8: Block Diagram Showing Pin and Node Locations

23S8: PALCE23S8

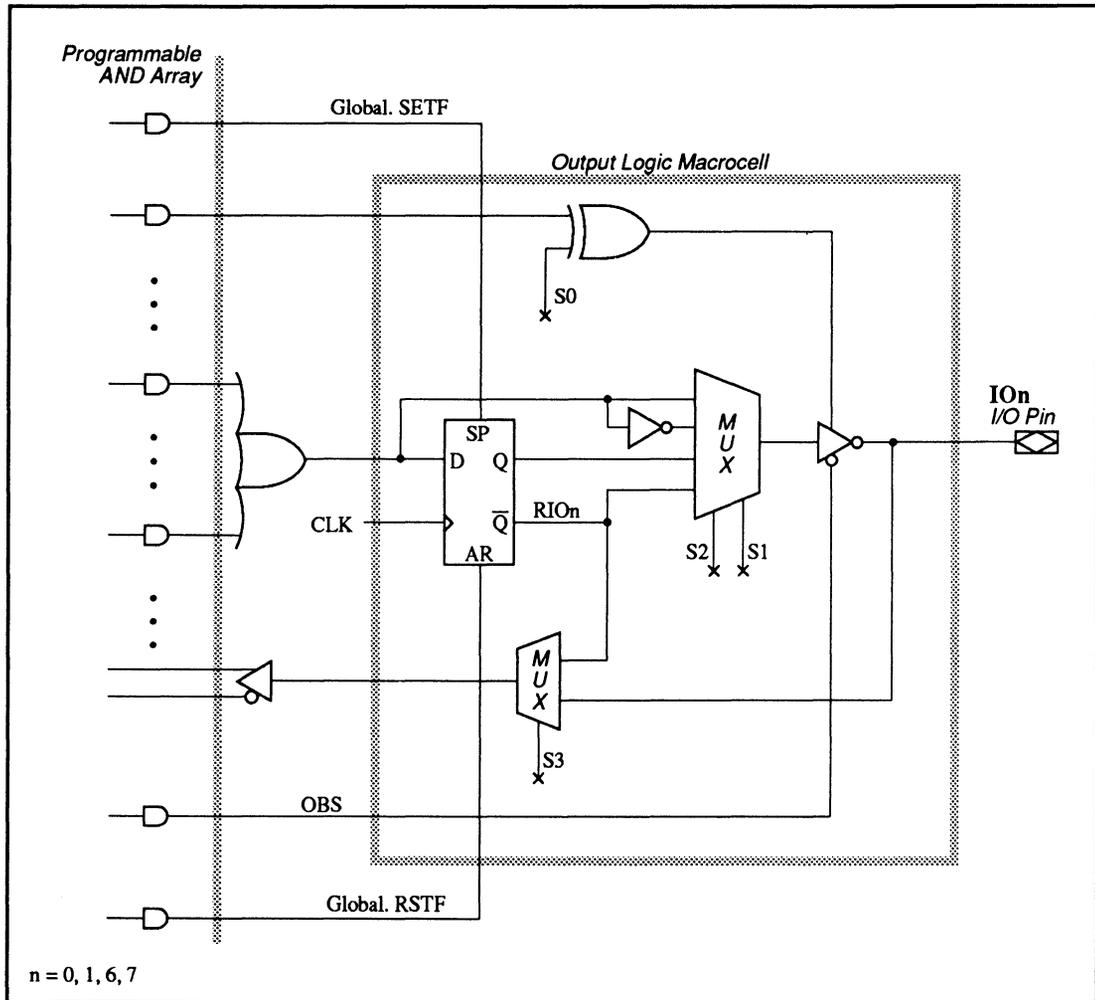


23S8: Output Register Macrocell Diagram



23S8: Buried Register Macrocell Diagram

23S8: PALCE23S8



23S8: Output Logic Macrocell Diagram

23S8: PALCE23S8

Example

Individual output-enable product term control with programmable polarity

```
:
PIN    2    I0           ;input
PIN    3    I1           ;input
PIN    12   I00  REG     ;output, registered
PIN    13   I01  REG     ;output, registered
PIN    14   I02  REG     ;output, registered
:
I00.TRST = I1 * I2       ;active-high output-enable control using AND product
;      term
I01.TRST = /(I1 * I2)    ;active-low output-enable control using NAND product
;      term
I02.TRST = /I1 + /I2     ;active-low output-enable control using SUM term
```

Macrocells with Different Configurations

The 23S8 has three different types of macrocells: output register, output logic macrocell, and buried register.

Each type of macrocell provides a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	OUTPUT REGISTER	OUTPUT LOGIC MACROCELL	BURIED REGISTER
Non-Programmable Feedback			
Combinatorial Output with I/O feedback	X	N/A	N/A
Registered Output with I/O feedback	X	N/A	N/A
Programmable Feedback			
Output with I/O feedback	N/A	X	N/A
Output with /Q feedback	N/A	X	N/A
Buried register with /Q feedback	N/A	X	X

26V12: PALCE26V12

PIN AND NODE DESCRIPTIONS

- 28 Pins
- 12 Buried nodes
- 1 Global preset, reset, and preload node

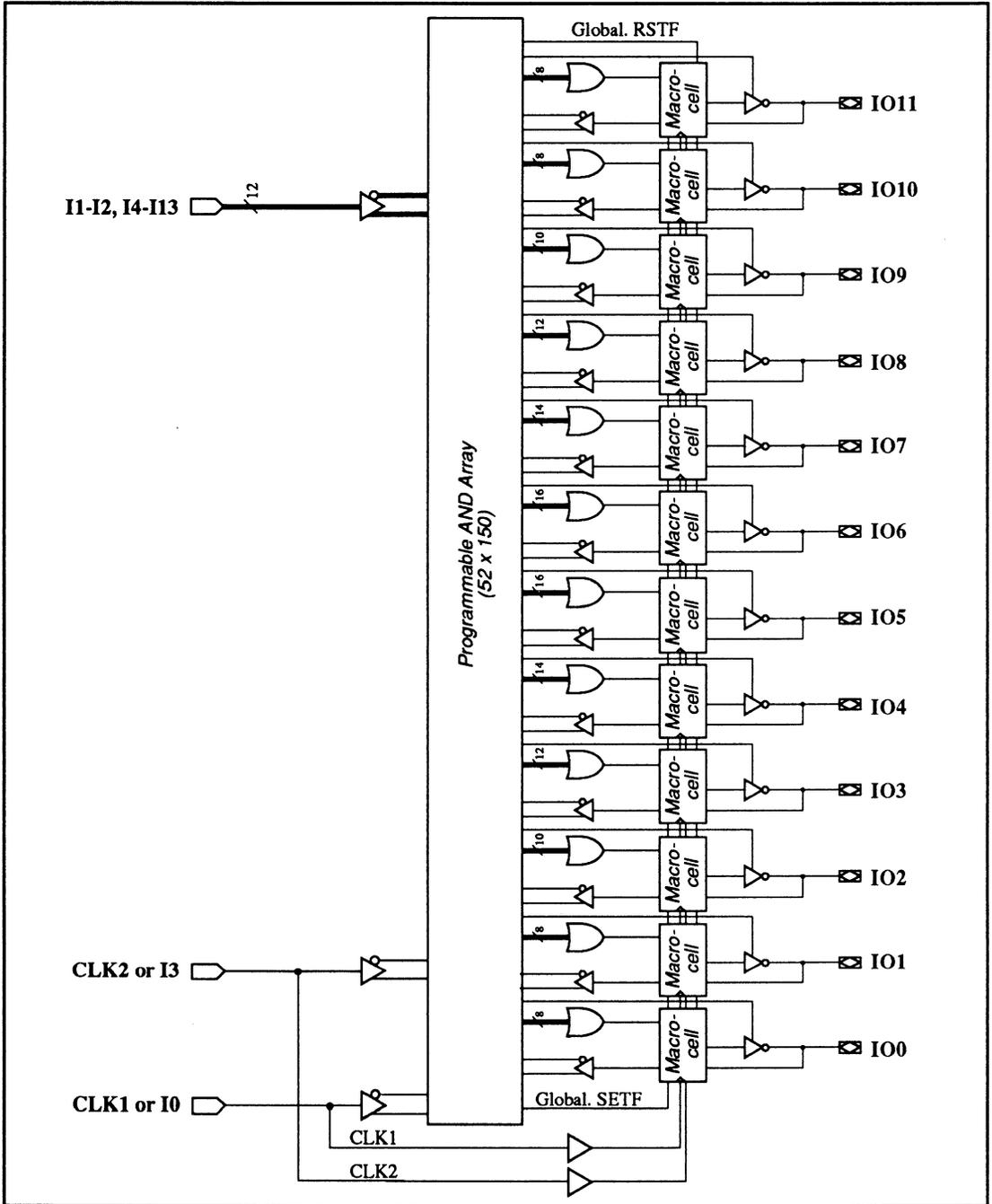
26V12

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK1 or I0	–	–	–
2 – 3	I1 – I2	–	–	–
4	CLK2 or I3	–	–	–
5 – 6	I4 – I5	–	–	–
7	VCC	–	–	–
8 – 14	I6 – I12	–	–	–
15 – 20	IO0 – IO5	2 – 7	R0 – R5	Register feedback
21	GND	–	–	–
22 – 27	IO6 – IO11	8 – 13	R6 – R11	Register feedback
28	I13	–	–	–
–	–	1	GLOBAL	Global preset and reset

BLOCK AND MACROCELL DIAGRAMS

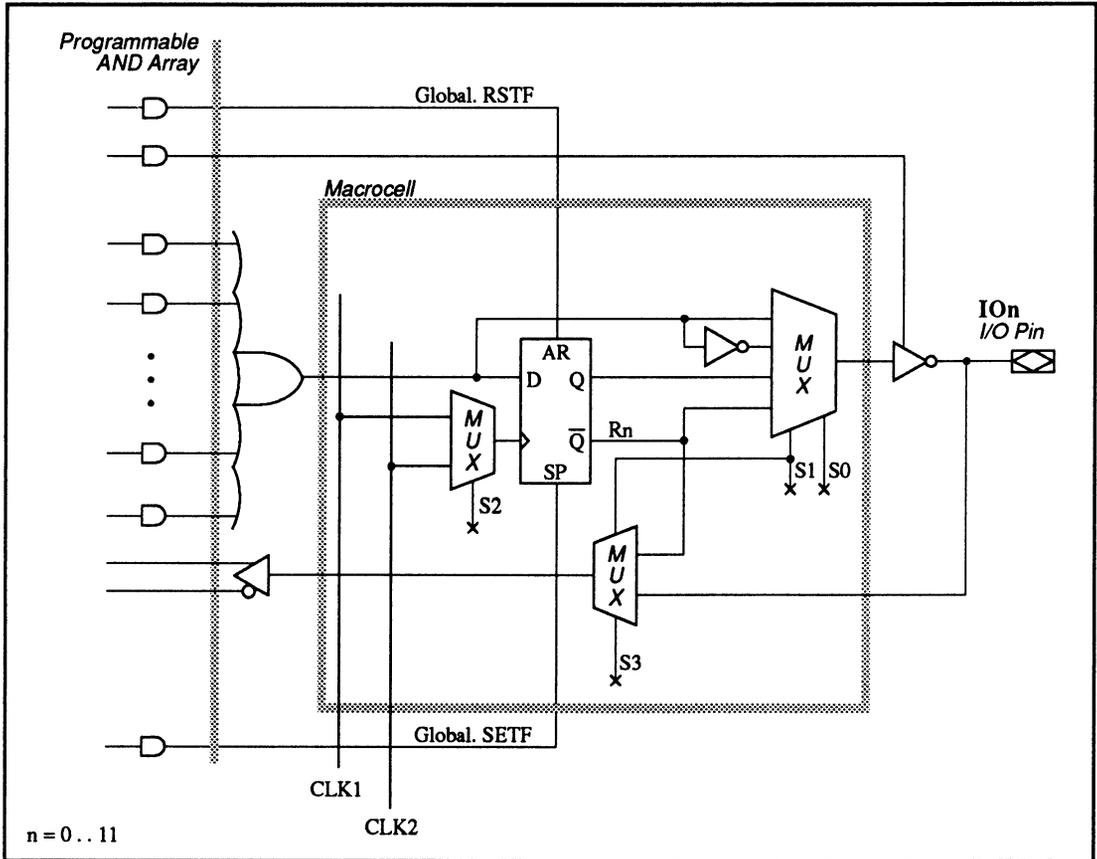
Block and macrocell diagrams follow.

26V12: PALCE26V12



26V12: Block Diagram Showing Pin and Node Locations

26V12: PALCE26V12



26V12: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Two external clock pins

Two External Clock Pins

The 26V12 allows you to individually select one of two external clock pins as the the clock input for each flip-flop. You can write a .CLKF functional equation, as shown next.

Syntax

```

Pin Statement(s) :
    PIN Clock_input_pin_location  Clock_input_pin_name
    PIN Output_pin_location       Output_pin_name           Storage_type
    :
Equation(s)      Output_pin_name.CLKF = Clock_input_pin_name

```

Example

```

:
PIN 1 CLK1 ;clock input
PIN 4 CLK2 ;clock input
PIN 14 I2 ;input
PIN 15 I00 REG ;output, registered
PIN 16 I01 REG ;output, registered
:
I00.CLKF = CLK1 ;I00 flip-flop uses CLK1 as clock signal
I01.CLKF = CLK2 ;I01 flip-flop uses CLK2 as clock signal

```

Equations written for the 22V10 are mapped to the 26V12 exactly as if they were implemented on the 22V10. This means that feedback is taken from /Q by default, rather than from the I/O pin. If you want feedback from the I/O pin, you must define the corresponding buried node in the pin statement portion of the design file. However, you should not write any equations for that node. If you use the pin name on the right side of an equation, feedback will be routed from the I/O pin rather than from /Q.

29M16: PALCE29M16

PIN AND NODE DESCRIPTIONS

- 24 pins
- 16 buried nodes
- 1 global preset, reset, and preload node
- 1 observability node

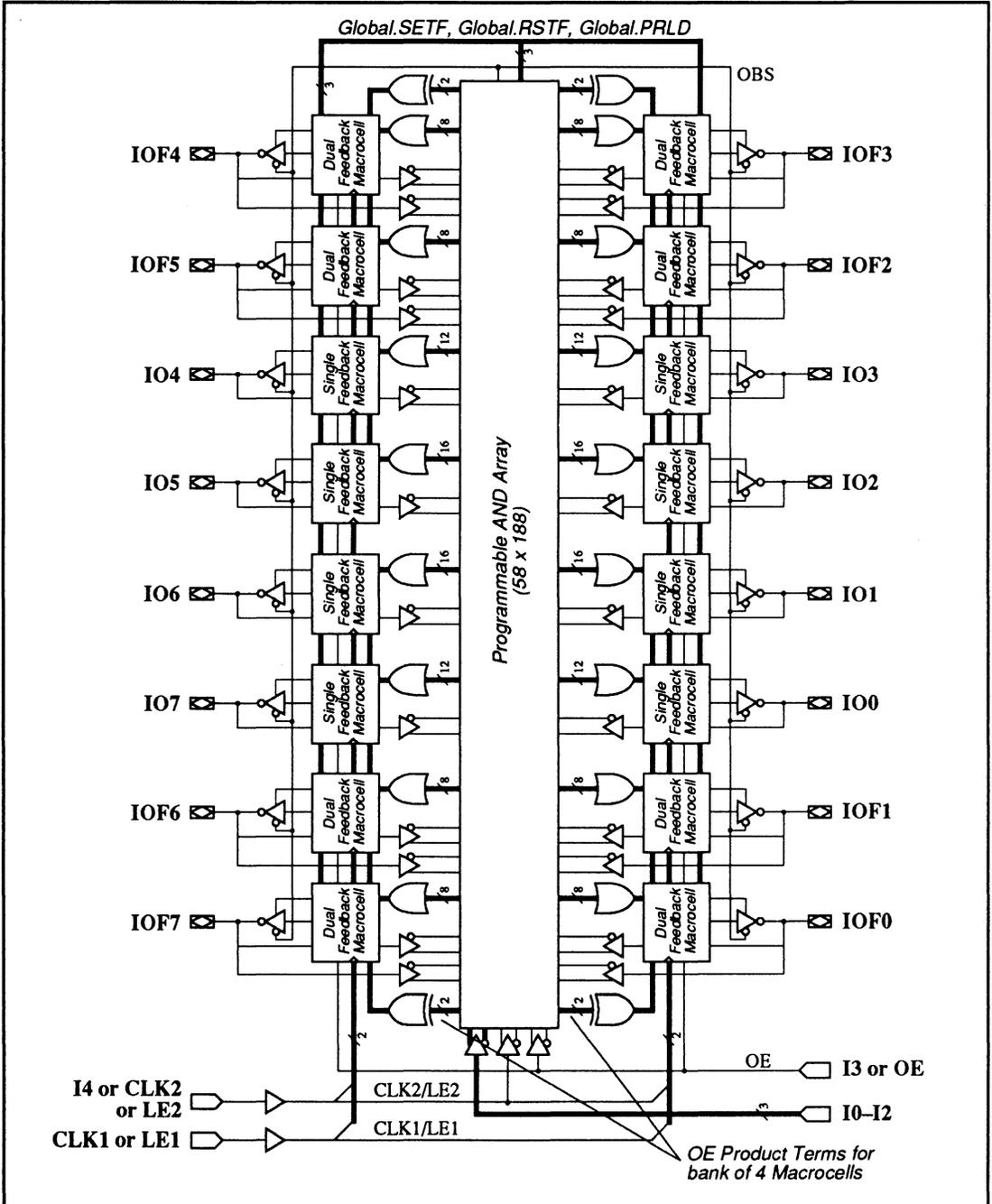
29M16

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK1 or LE	–	–	–
2	I0	–	–	–
3 – 4	IOF0 – IOF1	3 – 4	RF0 – RF1	Buried feedback of dual feedback macrocell
5 – 8	IO0 – IO3	5 – 8	R0 – R3	Buried feedback of single feedback macrocell
9 – 10	IOF2 – IOF3	9 – 10	RF2 – RF3	Buried feedback of dual feedback macrocell
11	I3 or OE	–	–	–
12	GND	–	–	–
13	I4 or CLK2 or LE	–	–	–
14	I1	–	–	–
15 – 16	IOF4 – IOF5	11 – 12	RF4 – RF5	Buried feedback of dual feedback macrocell
17 – 20	IO4 – IO7	13 – 16	R4 – R7	Buried feedback of single feedback macrocell
21 – 22	IOF6 – IOF7	17 – 18	RF6 – RF7	Buried feedback of dual feedback macrocell
23	I2	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preset, reset, and preload
–	–	2	OBS	Observability node

BLOCK AND MACROCELL DIAGRAMS

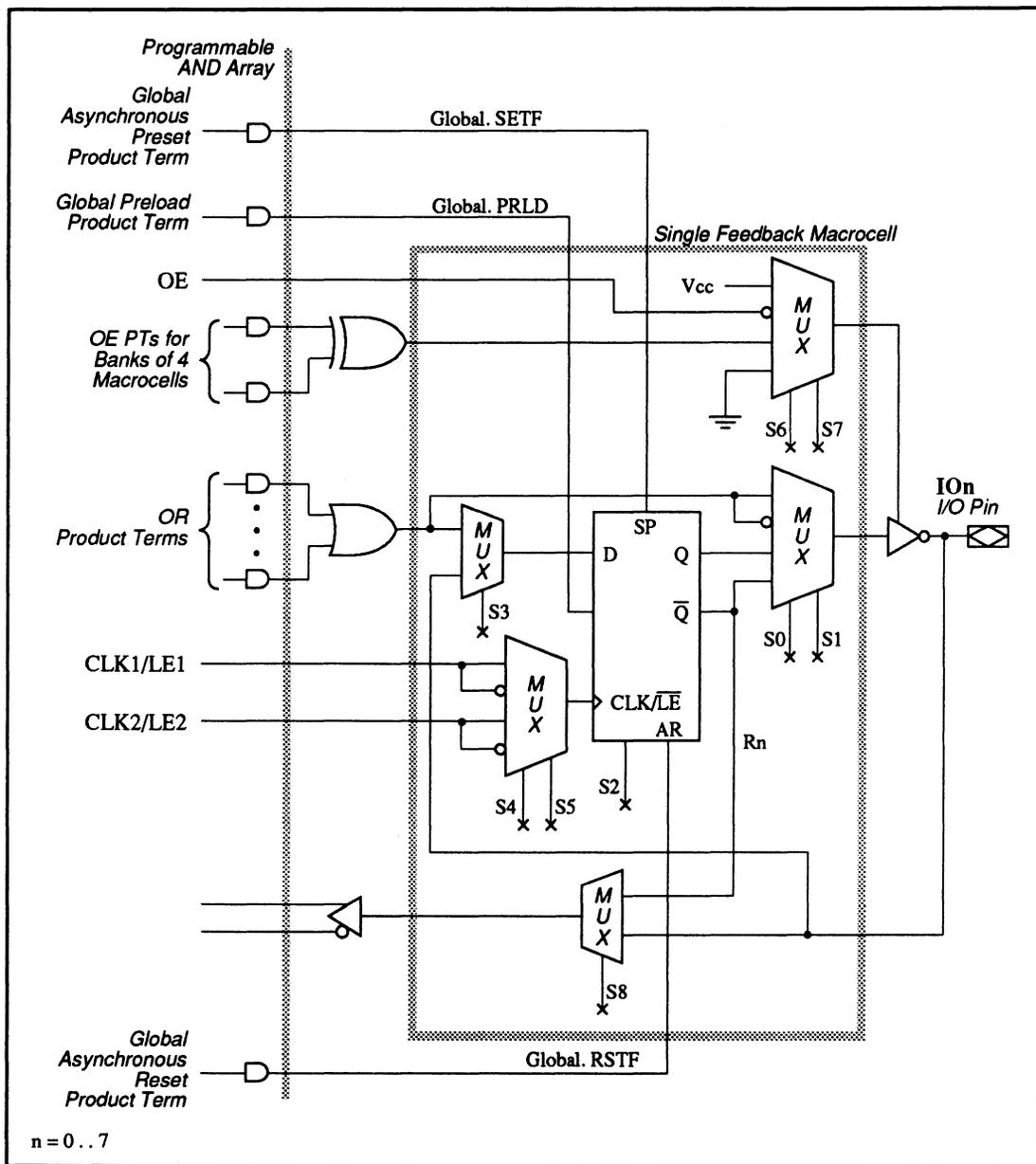
Block and macrocell diagrams follow.

29M16: PALCE29M16



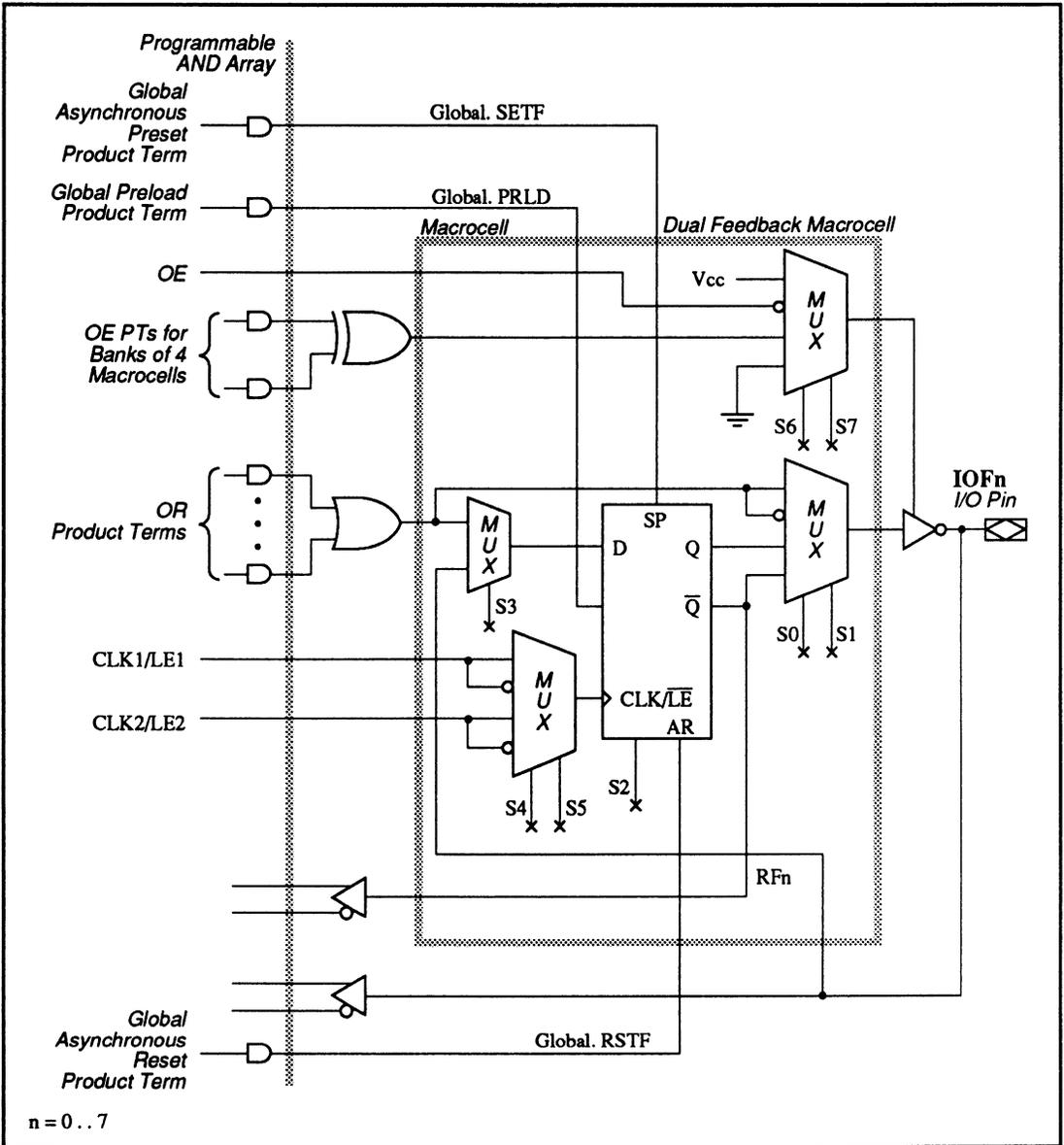
29M16: Block Diagram Showing Pin and Node Locations

29M16: PALCE29M16



29M16: Single Feedback Macrocell Diagram

29M16: PALCE29M16



29M16: Dual Feedback Macrocell Diagram

29M16: PALCE29M16

SPECIAL PROGRAMMING FEATURES

- Common external or grouped XOR output-enable control
- Programmable clock polarity
- Macrocells with different configurations

Common External or Grouped XOR Output-Enable Control

Each group of four outputs shares a common output-enable control, which can be selected from a common external output pin, or an XOR function with two product terms, or permanently enabled, or permanently disabled. The four groups of outputs are arranged as follows.

group 1: IOF0, IOF1, IO0, IO1
group 2: IOF2, IOF3, IO2, IO3
group 3: IOF4, IOF5, IO4, IO5
group 4: IOF6, IOF7, IO6, IO7

To program the output-enable functions, you can write a .TRST functional equation for any output within a group, or for an output vector using the language syntax shown below.¹²

¹² Refer to VECTOR in Chapter 10, in this section, for details on output vector.

Syntax 1

```
Pin Statement(s)  PIN  Input_pin_name
                  PIN  Output_pin_location      I/O_pin_name      Storage_type
                  :
```

```
Equation(s)  for external output-enable pin
              I/O_pin_name.TRST
              = Input_pin_name
or  grouped XOR function
    = Boolean expression using XOR function with two product terms*
or  permanently enabled
    = VCC
or  permanently disabled
    = GND
```

* If you write separate .TRST equations for each output sharing the same XOR function, each equation must list the same literals and operators in the same sequence. Otherwise, an error message occurs if used in the same design.

Example

```
:
PIN      11      OE                ;OE input
PIN      2       IO                ;input
PIN      14      I1                ;input
PIN      23      I2                ;input
PIN      3..4, 5..6      0[1..4]    ;grouped output pins 3,4,5,8 into an
                                   ;   output vector 0[1..4]
PIN      9..10, 7..8      0[5..8]
PIN      15..16, 17..18  0[9..12]
PIN      21..22, 19..20  0[13..16]
:
0[1..4].TRST = OE                ;output buffers 01 to 04 are controlled
                                   ;   by external OE pin
0[5..8].TRST = IO * I1           ;output buffers 05 to 08 are controlled
                                   ;   by the XOR function
0[9..12].TRST = VCC              ;output buffers 09 to 012 are permanently
                                   ;   enabled
0[13..16] = GND                  ;output buffers 013 to 016 are permanently
                                   ;   disabled
```

29M16: PALCE29M16

Programmable Clock Polarity

The 29M16 allows you to select one of two clock or latch-enable pins for each flip-flop using the .CLKF functional equation. An active-low equation produces either a falling-edge triggered clock or an active-low latch enable. An active-high equation produces either a rising-edge triggered clock or an active-high latch enable. To select which clock input, simply write a .CLKF functional equation for any output.

Syntax

```
Pin Statement(s) :  
    PIN Clock_input_pin_location  
    Clock_input_pin_name Input_type  
    PIN Output_pin_location I/O_pin_name Storage_type  
    :
```

```
Equation(s)      Output_pin_name.CLKF*  
    For common external clock input  
    = Clock_input_pin_name  
    or for individual clock product term  
    = Boolean expression with one product term
```

* For rising-edge-triggered clock or active-high latch enable, use an active-high output_pin_name; for falling-edge-triggered clock or active-low latch enable, use an active-low output_pin_name.

Example

```
:  
PIN 1 CLK1 ;input  
PIN 3 I00 REG ;output, registered  
:  
/I00 .CLKF = CLK1 ;clocks I00 register on the falling edge of CLK1 clock  
; signal
```

Macrocells with Different Configurations

The 29M16 has two types of macrocells, a single-feedback macrocell and a dual-feedback macrocell.

Each type of macrocell allows a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	SINGLE-FEEDBACK MACROCELL	DUAL-FEEDBACK MACROCELL
Programmable Feedback		
Output with I/O feedback	X	X
Output with /Q feedback	X	X
Output with /Q and I/O feedback	N/A	X
Buried register with /Q feedback	X	X
Register input with /Q	X	X

29MA16: PALCE29MA16

PIN AND NODE DESCRIPTIONS

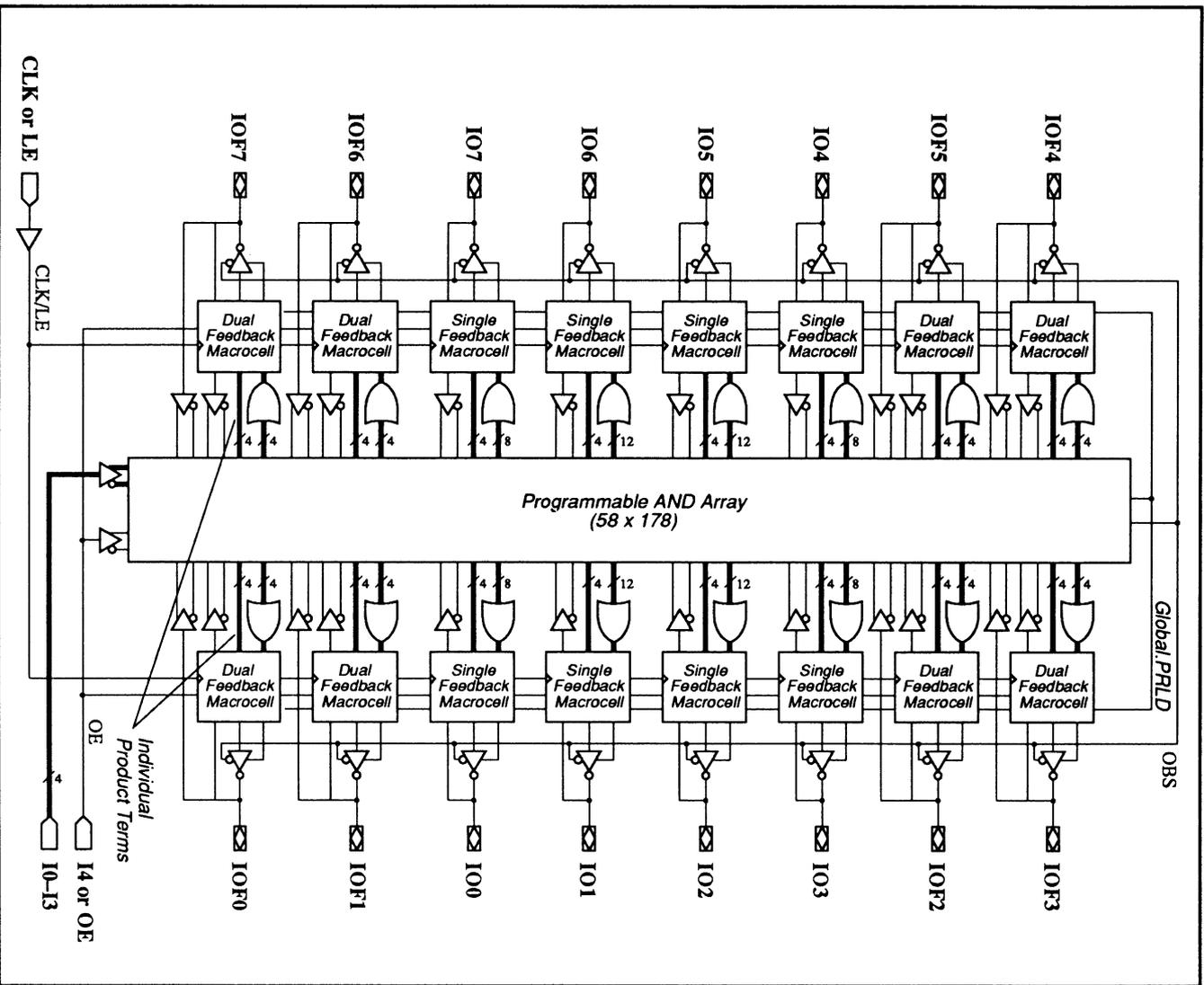
- 24 pins
- 16 buried nodes
- 1 global preset, reset, and preload node
- 1 observability node

29MA16

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK or LE	–	–	–
2	I0	–	–	–
3 – 4	IOF0 – IOF1	3 – 4	RF0 – RF1	Buried feedback of dual feedback macrocells
5 – 8	IO0 – IO3	5 – 8	R0 – R3	Buried feedback of single feedback macrocells
9 – 10	IOF2 – IOF3	9 – 10	RF2 – RF3	Buried feedback of dual feedback macrocells
11	I4 or OE	–	–	–
12	GND	–	–	–
13	I1	–	–	–
14	I2	–	–	–
15 – 16	IOF4 – IOF5	11 – 12	RF4 – RF5	Buried feedback of dual feedback macrocells
17 – 20	IO4 – IO7	13 – 16	R4 – R7	Buried feedback of single feedback macrocells
21 – 22	IOF6 – IOF7	17 – 18	RF6 – RF7	Buried feedback of dual feedback macrocells
23	I3	–	–	–
24	VCC	–	–	–
–	–	1	GLOBAL	Global preload
–	–	2	OBS	Observability node

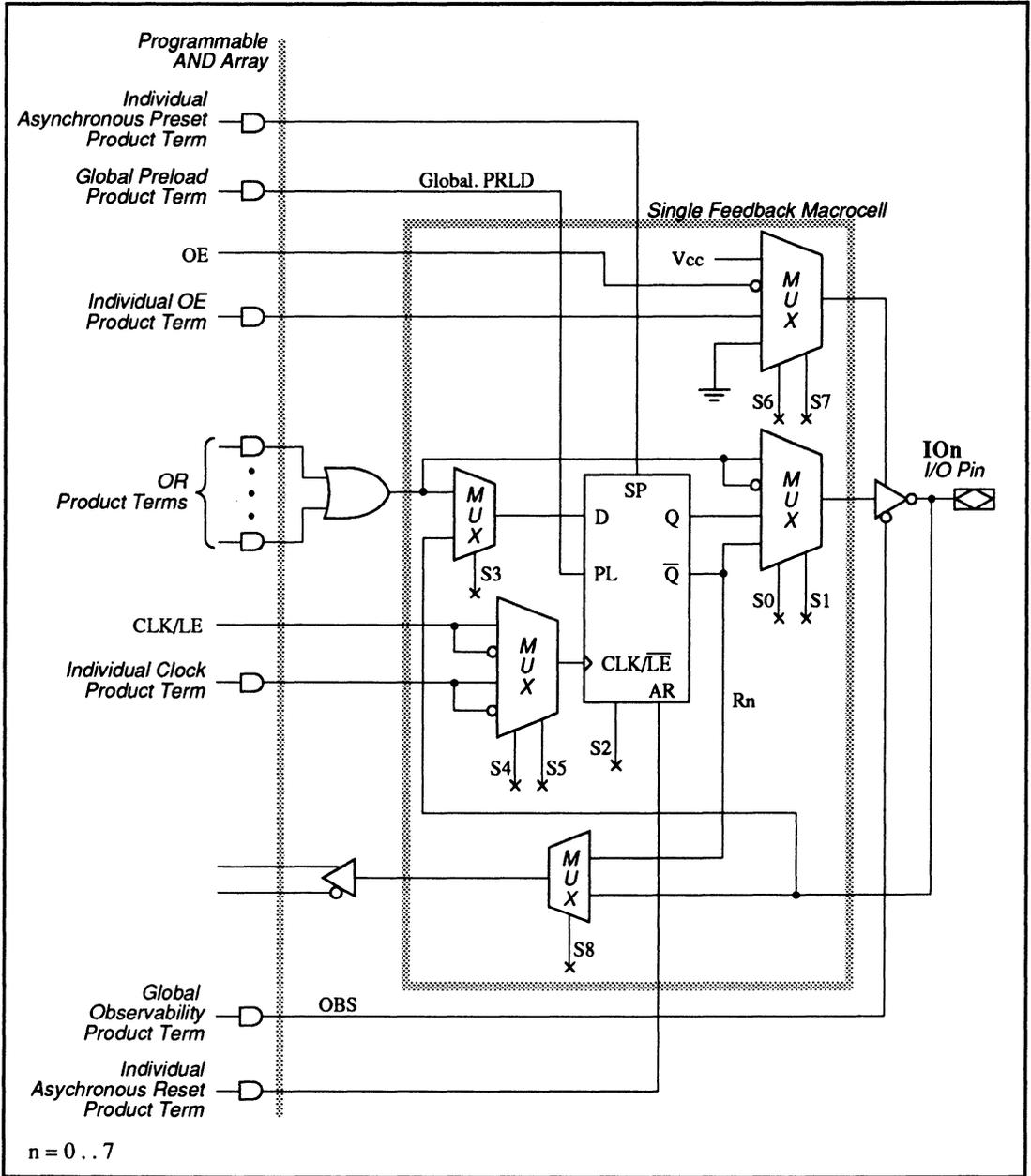
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



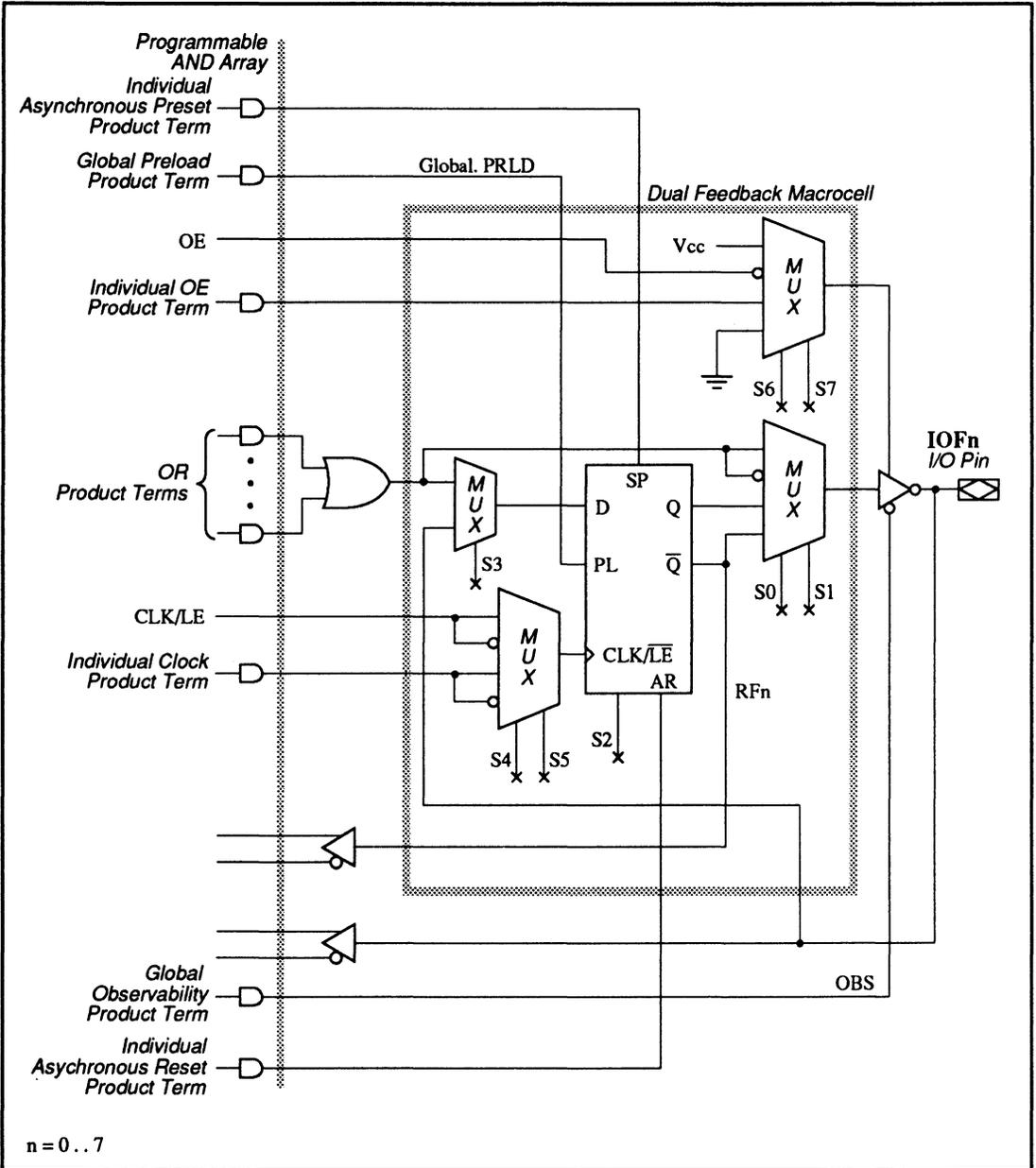
29MA16: Block Diagram Showing Pin and Node Locations

29MA16: PALCE29MA16



29MA16: Single Feedback Macrocell Diagram

29MA16: PALCE29MA16



29MA16: Dual Feedback Macrocell Diagram

29MA16: PALCE29MA16

SPECIAL PROGRAMMING FEATURES

- Clock control with individual clock product term or common external clock pin with programmable clock polarity
- Macrocells with different configurations

Clock Control

The 20MA16 allows you to select either a common external clock pin or an individual clock product term for each flip-flop using the .CLKF functional equation. An active-low equation produces either a falling edge triggered clock or an active-low latch enable. An active-high equation produces either a rising edge triggered clock or an active-high latch enable. To select a clock input, simply write a .CLKF functional equation for any output.

Syntax

```
Pin Statement(s) :  
    PIN Input_pin_location*   Input_pin_name**  
    PIN Output_pin_location I/O_pin_name REG or LAT  
    :
```

```
Equation(s)      Output_pin_name .CLKF  
                  for common external clock input  
                  = Input_pin_name  
    or            for individual clock product term  
                  = Boolean expression with one product term
```

* Clock or latch input

** For rising-edge-triggered clock or active-high latch enable, use an active-high output_pin_name; for falling-edge-triggered clock or active-low latch enable, use an active-low output_pin_name.

Example

```

:
PIN    1    CLK                ;clock input
PIN    2    I0                 ;input
PIN    14   I2                 ;input
PIN    3    I00    REG        ;output, registered
PIN    4    I01    REG        ;output, registered
PIN    5    I0     REG        ;output, registered
PIN    6    I1     REG        ;output, registered
:
/I00.CLKF = CLK                ;clocks I00 register on the falling edge of CLK
                                ; clock signal
I01.CLKF = CLK                 ;clocks I01 register on the rising edge of CLK1
                                ; clock signal
I0.CLKF = I0 * I2              ;output register clock is asserted when (I0 * I2)
                                ; is true
/I1.CLKF = I0 * I2             ;output register clock is asserted when /(I0 * I2) is
                                ; true

```

Macrocells with Different Configurations

The 29MA16 has two types of macrocells, a single-feedback macrocell and a dual-feedback macrocell.

Each type of macrocell provides a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	SINGLE-FEEDBACK MACROCELL	DUAL-FEEDBACK MACROCELL
Programmable Feedback		
Output with I/O feedback	X	X
Output with /Q feedback	X	X
Output with /Q and I/O feedback	N/A	X
Buried register with /Q feedback	X	X
Register input with /Q	X	X

30S16: PLS30S16

PIN AND NODE DESCRIPTIONS

- 28 pins
- 14 buried nodes
- 2 complement array nodes
- 1 observability nodes

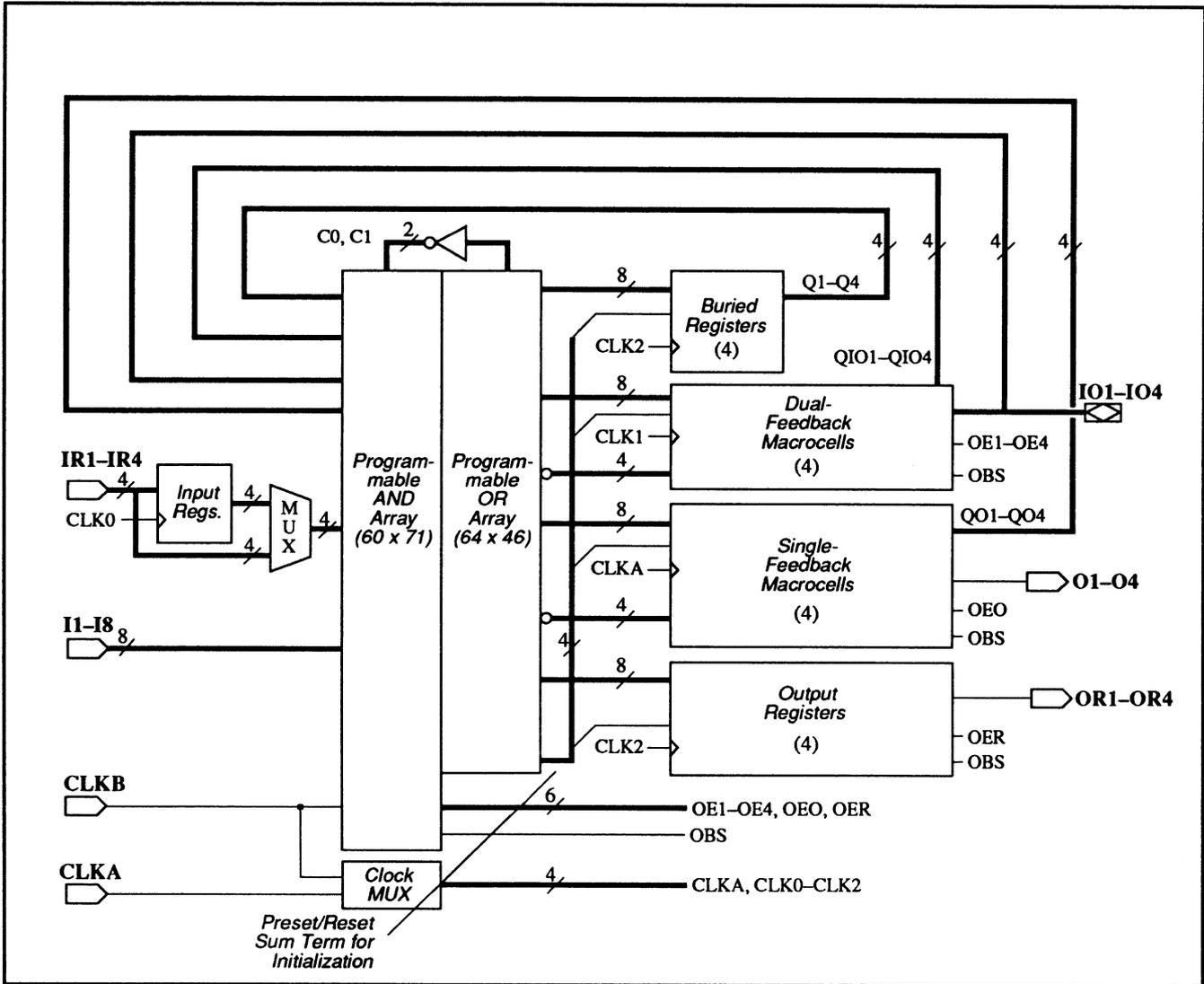
30S16

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLKA	-	-	-
2 - 3	I1 - I2	-	-	-
4	CLKB	-	-	-
5 - 7	I3 - I5	-	-	-
8 - 9	IO1 - IO2	5 - 6	QIO1 - QIO2	Q feedback from I/O
10 - 13	OR1 - OR4	-	-	-
14	GND	-	-	-
15 - 18	O1 - O4	1 - 4	Q01 - Q04	Q feedback from output
19 - 20	IO3 - IO4	7 - 8	QIO3 - QIO4	Q feedback from I/O
21 - 24	IR1 - IR4	-	-	-
25 - 27	I6 - I8	-	-	-
28	VCC	-	-	-
-	-	9 - 12	Q1 - Q4	Buried Q feedback
-	-	13 - 14	C0 - C1	Complement arrays
-	-	15	OBS	Observability

BLOCK AND MACROCELL DIAGRAMS

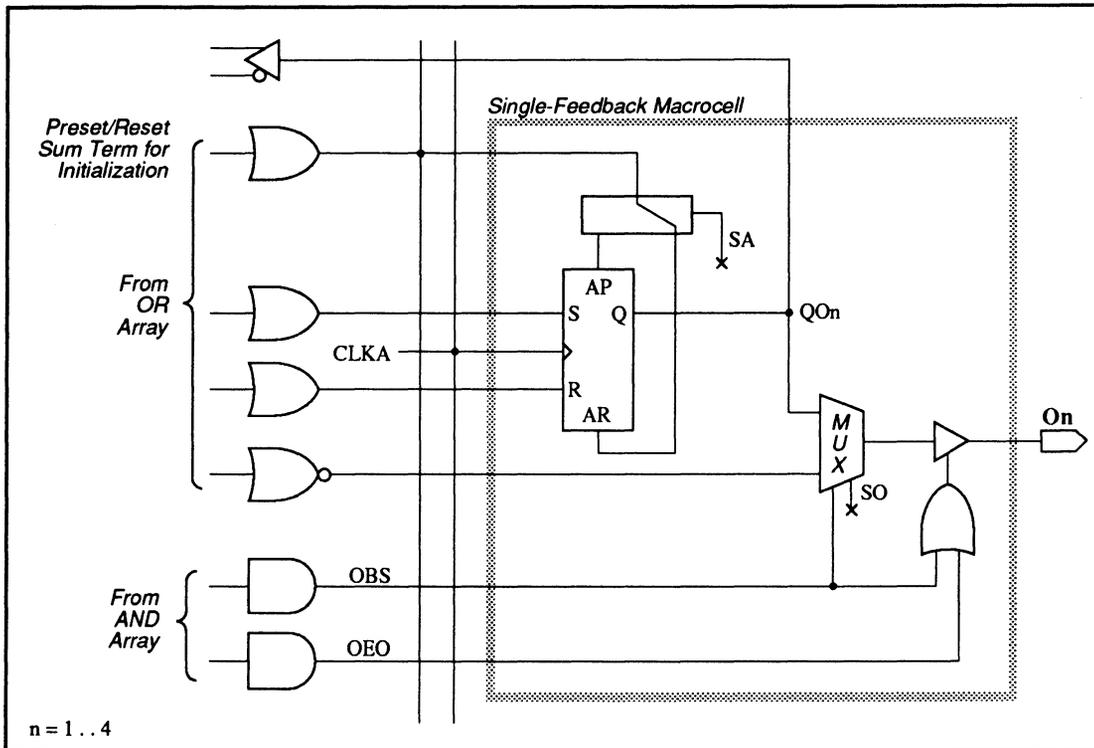
Block and macrocell diagrams follow.

30S16: Block Diagram Showing Pin and Node Locations

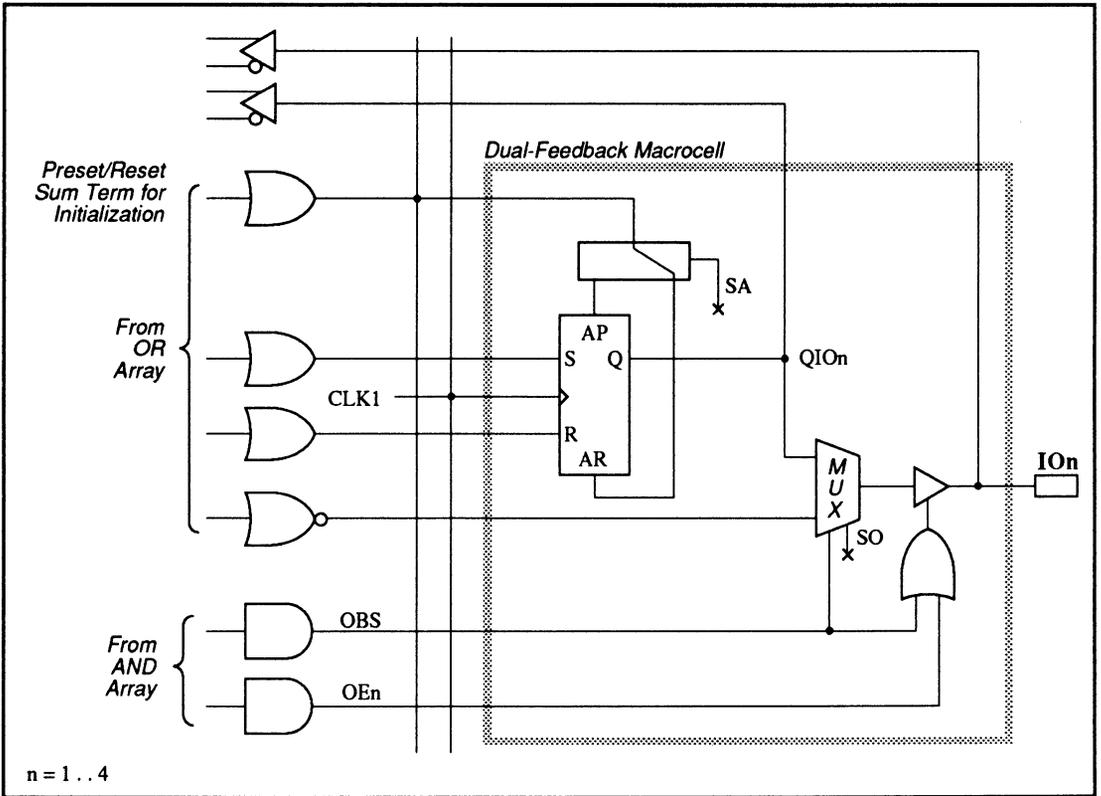


30S16: PLS30S16

30S16: PLS30S16

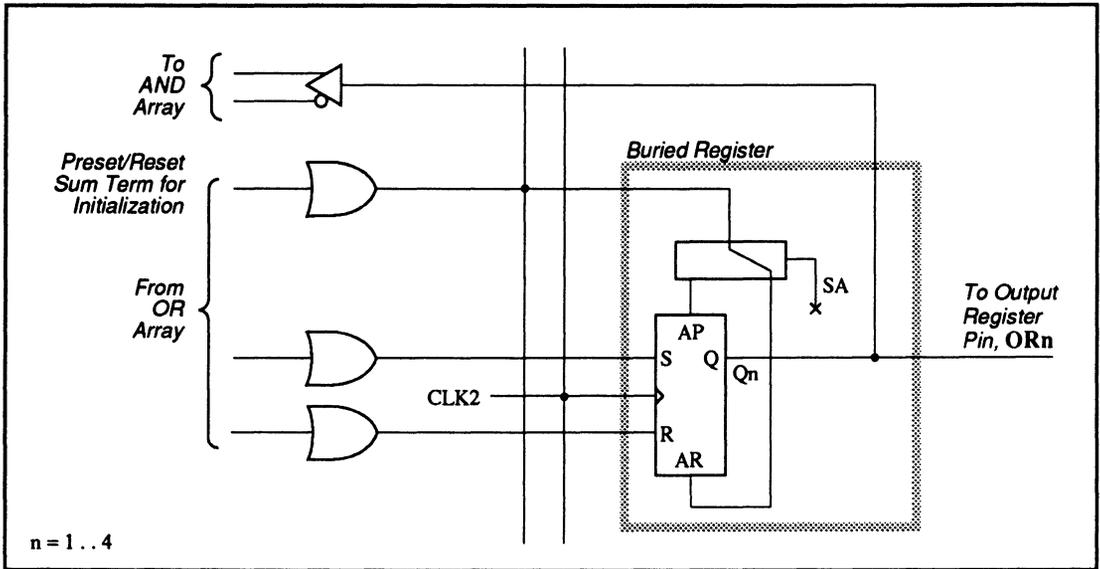


30S16: Single Feedback Macrocell Diagram

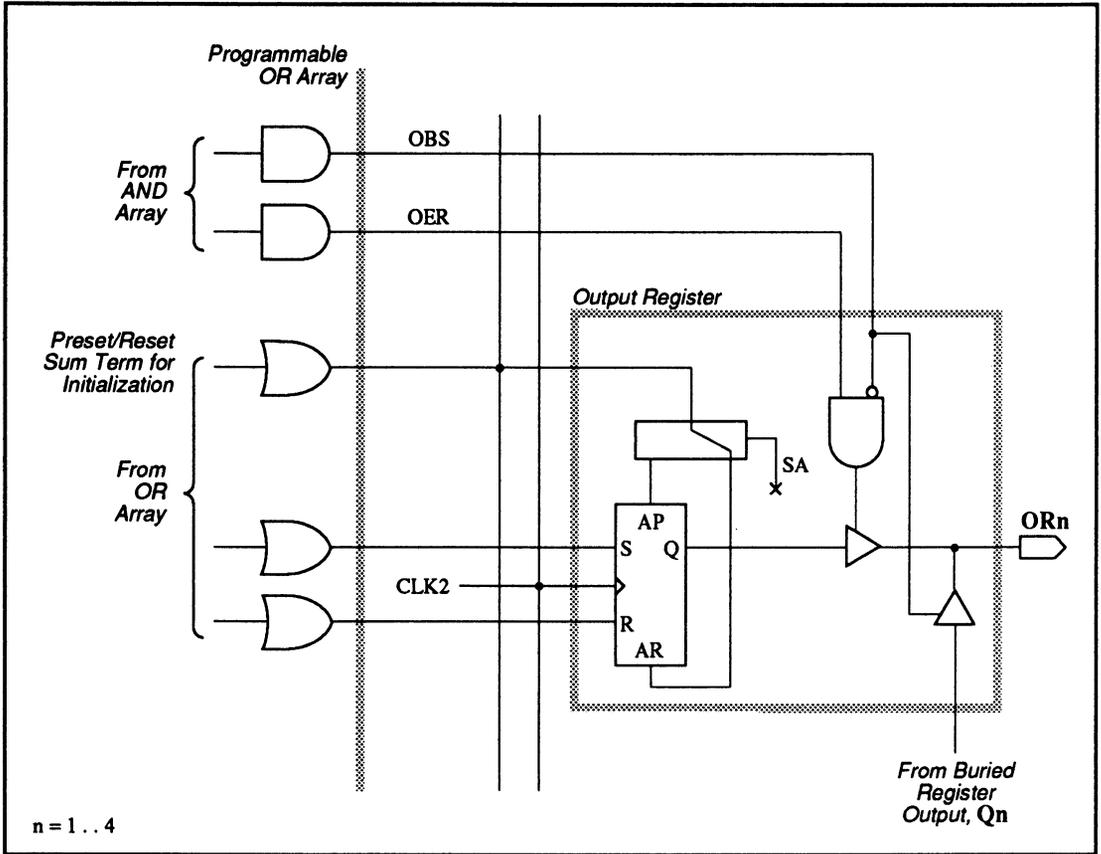


30S16: Dual Feedback Macrocell Diagram

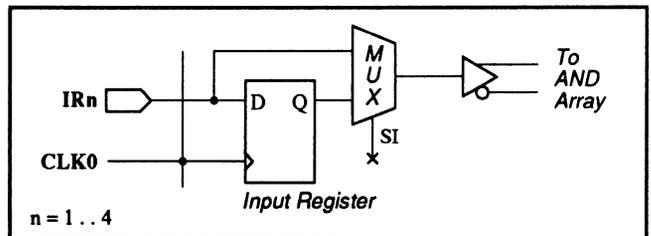
30S16: PLS30S16



30S16: Buried Register Macrocell Diagram

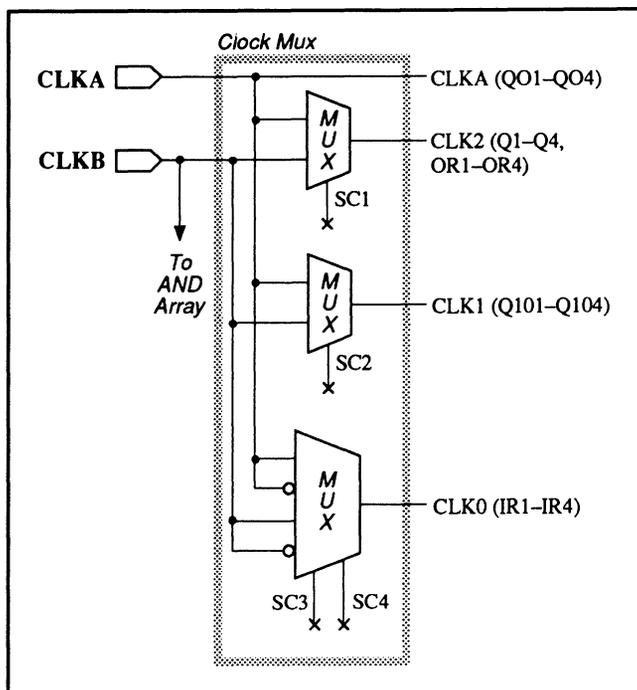


30S16: Output Register Macrocell Diagram



30S16: Input Register Macrocell Diagram

30S16: PLS30S16



30S16: Clock MUX Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Individual/group output-enable product terms
- Multiple clock controls
- Preset/reset sum terms for initialization
- Macrocells with different configurations
- Outputs with Q feedback
- Outputs with I/O and Q feedbacks
- Registered Inputs
- State bits

Individual and Group Output-Enable Product Terms

There are two types of output-enable control.

- Individual output-enable product term
- Group product term

OUTPUTS	PRODUCT TERMS	TYPE OF PRODUCT
IO1 - IO4	OE1 - OE4	Individual
O1 - O4	OEO	Group
OR1 - OR4	OER	Group

Individual Output-Enable Product Term

Outputs IO1 to IO4 each have an individual product term to control the output buffer. You use the syntax below to program each product term.

Syntax

Pin Statement(s) PIN Output_pin_location I/O_pin_name Storage_type
:

Equation(s) Output_pin_name.TRST = Boolean expression with one product term

Example

```

:
PIN    2    I1    COMB    ;input, combinatorial
PIN    3    I2    COMB    ;input, combinatorial
:
PIN    8    I01   REG     ;I/O, registered
:
I01.TRST = I1 * /I2      ;I01 is individually controlled by (I1 * /I2)

```

30S16: PLS30S16

Group Product Term

The output buffers for outputs O1 to O4 are controlled by one product term, OEO. The output buffers for outputs OR1 to OR4 are controlled by another product term, OER. To program these groups of product terms, write a .TRST equation for any output within a group using the syntax shown above. You can also group the outputs together and write a .TRST equation for the assigned group name using the GROUP statement, or write a .TRST equation for an output vector.¹³

Example

```
:
PIN    2   I1   COMB      ;input, combinatorial
PIN    3   I2   COMB      ;input, combinatorial
:
PIN    15  01   REG       ;output, registered
PIN    16  02   REG       ;output, registered
PIN    17  03   REG       ;output, registered
PIN    18  04   REG       ;output, registered
:
PIN    10  OR1  REG       ;output, registered
:
GROUP  Group0 01 02 03 04
:
Group0.TRST = I1 * /I2      ;01 to 04 is controlled by ( I1 * /I2 )
OR1.TRST = I2              ;OR1 to OR4 is controlled by ( I2 )
```

¹³ Refer to VECTOR in Chapter 10, in this section, for details.

Multiple Clock Controls

The 30S16 has four clock controls. Each controls a group of registers and can be programmed into a selection of clock signals, as shown in the clock MUX macrocell diagram. The clock arrangement is summarized in the table below.

OUTPUTS	CLOCK CONTROLS	CLOCK SIGNALS
IR1 - IR4	CLK0	CLKA (default) /CLKA CLKB /CLKB
IO1 - IO4	CLK1	CLKA (default) CLKB
OR1 - OR4, Q1 - Q4	CLK2	CLKA (default) CLKB
O1 - O4	CLKA	CLKA

You can assign a clock to a clock control by writing a .CLKF equation for any output within the clock bank. You can also use the VECTOR or GROUP statements to assign a group of registers.¹⁴ If you do not assign any clock signal to a bank of registers, the default clock signal is used.

Note: You may only assign the identical clock signal to the output registers within a clock bank.

Syntax

```
Pin Statement(s) :
    PIN I/O_pin_location I/O_pin_name Storage_type
    :
```

```
Equation(s)      Output_pin_name.CLKF = Clock_signal
```

¹⁴ Refer to Chapter 10, in this section, for details.

30S16: PLS30S16

Example

```
:
PIN 1 CLKA ;clock input
PIN 4 CLKB ;clock input
:
PIN 8..9, 19..20 IO[1..4] ;output vector
PIN 21..24 IR[1..4] ;output vector
PIN 10 OR1 REG ;output, registered
:
IO[1..4].CLKF = CLKB ;IO1 to IO4 registers are controlled by CLKB
OR1.CLKF = CLKA ;OR1 to OR4 registers are controlled by CLKA
:
; if no clock is assigned to outputs IR1 to
; IR4, by default they are controlled by
; CLKA
```

Preset/Reset Sum Term for Initialization

All registers in the 30S16 can be initialized to a pre-defined state under certain conditions. Each flip-flop contains a preset and a reset line, only one of which can be selected. There are four sum terms. Each sum term defines the conditions under which the selected set or reset line is activated within a bank of registers with the same clock control.¹⁵ While the initialization condition must be the same for each flip-flop in a register bank, you can individually program the preset/reset sum term to set or reset each flip-flop.

You can initialize registers using either state-machine language or Boolean equations. The PALASM language syntax for both follows.

¹⁵ See the multiple clock-control feature for the register-bank arrangements.

```

Pin Statement(s) :
    I/O_pin_location      I/O_pin_name      Storage_type
    :
    Buried_node_location  Buried_node_name  Storage_type

```

```

Equation(s) for initialization to one
    I/O_pin_name.SETF      = Boolean expression
    Buried_node_name.SETF = Boolean expression
    or
    for initialization to zero
    I/O_pin_name.RSTF      = Boolean expression
    Buried_node_name.RSTF = Boolean expression

```

Note: The Boolean expression used on the right side of the functional equations must use the same sequence of literals and operators for all outputs in the same register bank, since all use the same sum term.

State-Machine Language

For a state machine design, follow the steps below to initialize a state machine to a known starting state using the programmable, asynchronous preset/reset sum term for each bank of registers.

1. Initialize the state machine in the conditions segment, by defining the initialization condition as a set of input and/or feedback value(s).

```

CONDITIONS
INIT = Boolean expression

```

2. Define the starting state and outputs as a function of the INIT condition in the Setup and Defaults portion of the state segment using the syntax below.

```
START_UP := Condition -> Desired_state
```

or for a Mealy state machine design

```
START_UP.OUTF := Condition -> Desired_outputs
```

The next example shows initialization to a known state using a state-machine design for a 30S16 device.

30S16: PLS30S16

Example

```
:
PIN      2      I1          ;input
PIN      3      I2          ;input
PIN      5      I3          ;input
PIN      6      I4          ;input
PIN      15     01      REG  ;output,registered
PIN      16     02      REG  ;output,registered
:
NODE     9      Q1      REG  ;buried node
:
STATE
MEALY_MACHINE
DEFAULT_BRANCH  HOLD_STATE
START_UP := INIT -> NEXT_STATE
START_UP.OUTF :=  INIT -> 01 * Q2          ;when INIT is true, outputs 01 and 02 will be
; set to high, while other outputs will be
; reset to low

CONDITIONS
INIT = I1 * I2 * I3 * I4          ;INIT is defined as condition
; (I1 * I2 * I3 * I4)
```

Boolean Equations

You can initialize the registers using Boolean equations. Defining the programmable initialization function consists of writing a .SETF or .RSTF functional equation to set or reset for each flip-flop. The Boolean expression using sum term defines the conditions under which initialization occurs. The .SETF or .RSTF equation defines the value (set or reset) of each flip-flop. If all flip-flops within a bank of registers are set or reset to the same value, you can write a single equation for an output vector.¹⁶

¹⁶ Refer to Chapter 10, in this section, for details.

Syntax

```
Pin Statement(s)  I/O_pin_location  I/O_pin_name  Storage_type
                  :
                  Buried_node_location  Buried_node_name  Storage_type
```

Equation(s)

for initialization to one

I/O_pin_name.SETF = Boolean expression

Buried_node_name.SETF = Boolean expression

or for initialization to zero

I/O_pin_name.RSTF = Boolean expression

Buried_node_name.RSTF = Boolean expression

Note: The Boolean expression used on the right side of the functional equations must use the same sequence of literals and operators for all outputs in the same register bank, since all use the same sum term.

Example

```
:
PIN    2      I1                ;input
PIN    3      I2                ;input
PIN    5      I3                ;input
PIN    6      I4                ;input
PIN    8..9,  19..20  I0[1..4]  REG  ;output, registered
PIN    15..16, 17..18  0[1..4]  REG  ;output, registered
:
NODE   9  Q1                    REG  ;buried node
I0[1..4].SETF = I1 * I2 * /I4      ;when (I1*I2*/I4) is true, registers I01 to
                    +I3 * /I2      ; I04 are set to high
O[1..2].SETF = I1 * I2 * /I4      ;when ((I1*I2*/I4)+(I3*/I2)) is true,
                    ; registers 01 & 02 are set to high
O[3..4].RSTF = I1 * I2 * /I4      ;registers 03 & 04 are set to low under the
                    +I3 * /I2      ; same condition as 01 and 02
Q1.RSTF = I1 * I2                ;buried register Q1 is set to low when
                    ; (I1 * I2) is true
```

30S16: PLS30S16

Macrocells with Different Configurations

The 30S16 has four types of macrocells: output register, single-feedback macrocell, dual-feedback macrocell, and buried register. Each type of macrocell provides a different set of feedback configurations, as tabulated below. Allowable configurations are marked with an X in the table.

FEEDBACK CONFIGURATIONS	OUTPUT REGISTER	SINGLE-FEEDBACK MACROCELL	DUAL-FEEDBACK MACROCELL	BURIED REGISTER MACROCELL
Non-Programmable Feedback Registered Output with I/O feedback	X	N/A	N/A	N/A
Programmable Feedback Output with Q feedback	N/A	X	X	N/A
Output with Q and I/O feedback	N/A	N/A	X	N/A
Buried register with Q feedback	N/A	X	X	X

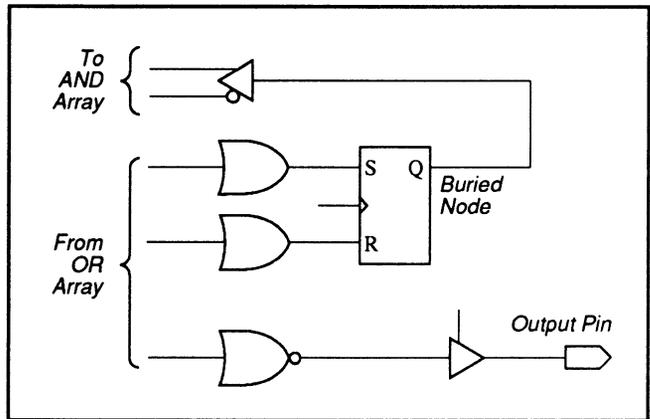
Output with Q Feedback

The single-feedback macrocell in a 30S16 can be configured as either a combinatorial or registered output with the Q register output feeding back to the array. For a combinatorial output, the output and the register each have separate logic using separate product terms. The output uses active-low logic; the buried register uses active-high logic.

Example

Combinatorial output with a feedback

```
:  
PIN    2    I1    COMB    ;input  
PIN    3    I2    COMB    ;input  
PIN    8    I01   COMB    ;I/O, combinatorial  
PIN    15   01    REG     ;I/O, registered  
:  
/I101 = I * /I2          ;defines I01's comb output  
I01.S = I2
```



30S16: Combinatorial Output with Q Feedback in Single-Feedback Macrocell

Syntax

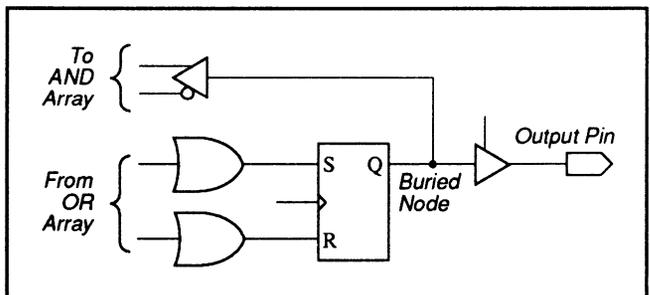
Combinatorial output with Q feedback

Pin Statement(s) :

PIN	I/O_pin_location	I/O_pin_name	COMB
:			
NODE	Buried_node_number	Buried_node_name	REG
:			

Equation(s)

/I/O_pin_name	=	Boolean expression
Buried_node_name.S	=	Boolean expression
Buried_node_name.R	=	Boolean expression
:		
< Equations using buried_node_name feedback >		



30S16: Registered Output with Q Feedback in Single-Feedback Macrocell

30S16: PLS30S16

Syntax

Registered output with Q feedback

```
Pin Statement(s)  PIN  I/O_pin_location I/O_pin_name REG
                  :
                  NODE Buried_node_number Buried_node_name REG
                  :
Equation(s)       I/O_pin_name .S = Boolean expression
                  I/O_pin_name .R = Boolean expression
                  :
                  < Equations using buried_node_name as feedback >
```

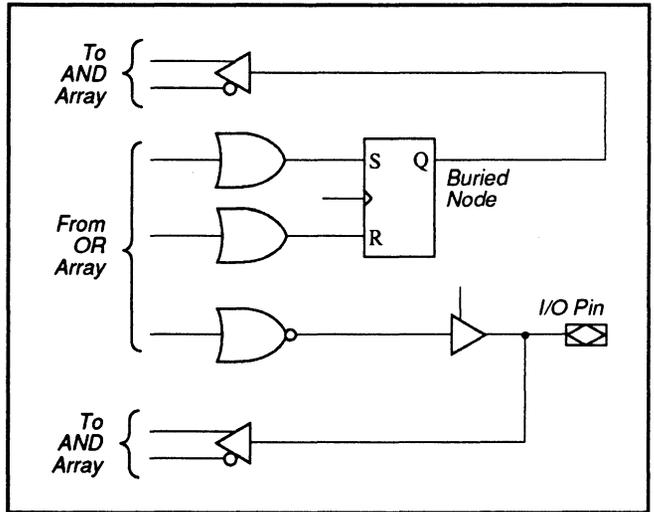
Example

Registered output with Q feedback

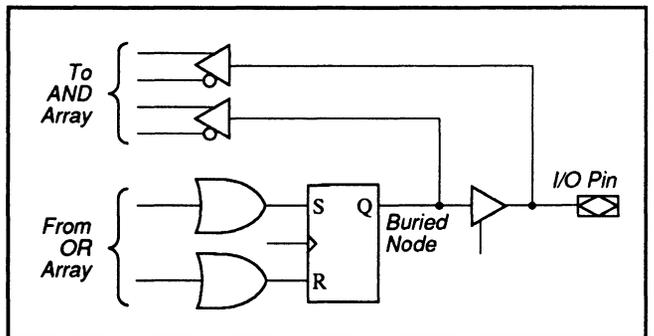
```
:
PIN    2    I1    COMB    ;input, combinatorial
PIN    3    I2    COMB    ;input, combinatorial
:
PIN    15   01    REG     ;output, registered
PIN    16   02    COMB    ;output, combinatorial
:
NODE   1    Q01   REG     ;buried node for register of 01
:
01.S = I1 * I2           ;equation for registered output I01, input S
01.R = /I2               ;equation for registered output I01, input R
Q01.S = { 01.S }        ;S equation for 01's buried node
Q01.R = { 01.R }        ;R equation for 01's buried node
/Q2 = Q02               ;equation using buried Q output, Q01, as feedback
```

Output with Q and I/O Feedbacks

The dual macrocell in a 30S16 can be configured to have both the I/O and Q register outputs feeding back to the logic array for either combinatorial or registered outputs. The language syntax for such arrangements is similar to those described above with just the Q feedback except that you can now include both the Q and I/O feedbacks in an output equations, as illustrated in the example below.



30S16: Combinatorial Output with Q and I/O Feedbacks



30S16: Registered Output with Q and I/O Feedbacks in Dual-Feedback Macrocell

30S16: PLS30S16

Example

Combinatorial and registered output with Q and I/O Feedback

```
      :
PIN   2   I1   COMB   ;input
PIN   3   I2   OMB    ;input
PIN   8   I01  REG    ;I/O, registered
PIN   9   I02  COMB   ;I/O, combinatorial
PIN  19   I03  COMB   ;I/O, combinatorial
:
NODE  5   QI01 REG    ;buried node for register of I01
NODE  6   I02  REG    ;buried node for register of I02
NODE  7   QI03 REG    ;buried node for register of I03
:
I01.S = I2 * I1      ;equation for registered output I01, input S
I01.R = /I2          ;equation for registered output I01, input R
:
QI02.S = I2 * / I1   ;equation for buried register of output I02, input S
QI02.R = /I2 * /I1   ;equation for buried register of output I02, input R
/I02 = I2            ;equation for combinatorial output I02
:
QI03.S = /I2 * QI02  ;equation using buried Q output, QI02, as feedback
QI03.R = /I2 * / QI01 ;equation using registered Q output, QI01, as feedback
I03 = I01 * /I02     ;equation using registered and combinatorial output,
;   I01 and I02 as feedback
```

Registered Inputs

Inputs IR1 to IR4 can be used as registered D-type or combinatorial inputs. The language syntax is shown below.

Syntax

```
Pin Statement(s) :
    for combinatorial inputs
        PIN Input_pin_location      Input_pin_name      COMB
    :
or   for registered Inputs
        PIN Input_pin_location      Input_pin_name      REG
```

Note: Combinatorial is the default, which is used when you do not assign a specific storage type.

Example

```
:  
PIN    21  IR1  COMB    ;input, combinatorial  
PIN    22  IR2  REG     ;input, registered  
PIN    23  IR3                ;input, combinatorial
```

State Bits

The buried registers and all output registers with feedback paths to the AND array can be used for state bit storage. The register associated with the outputs below can be used for state bit storage.

Q1 to Q4
O1 to O4
IO1 to IO4

<p>Note: Do not assign names to the outputs and/or buried state registers when you want to assign state bits automatically.</p>
--

32VX10: PAL32VX10

PIN AND NODE DESCRIPTIONS

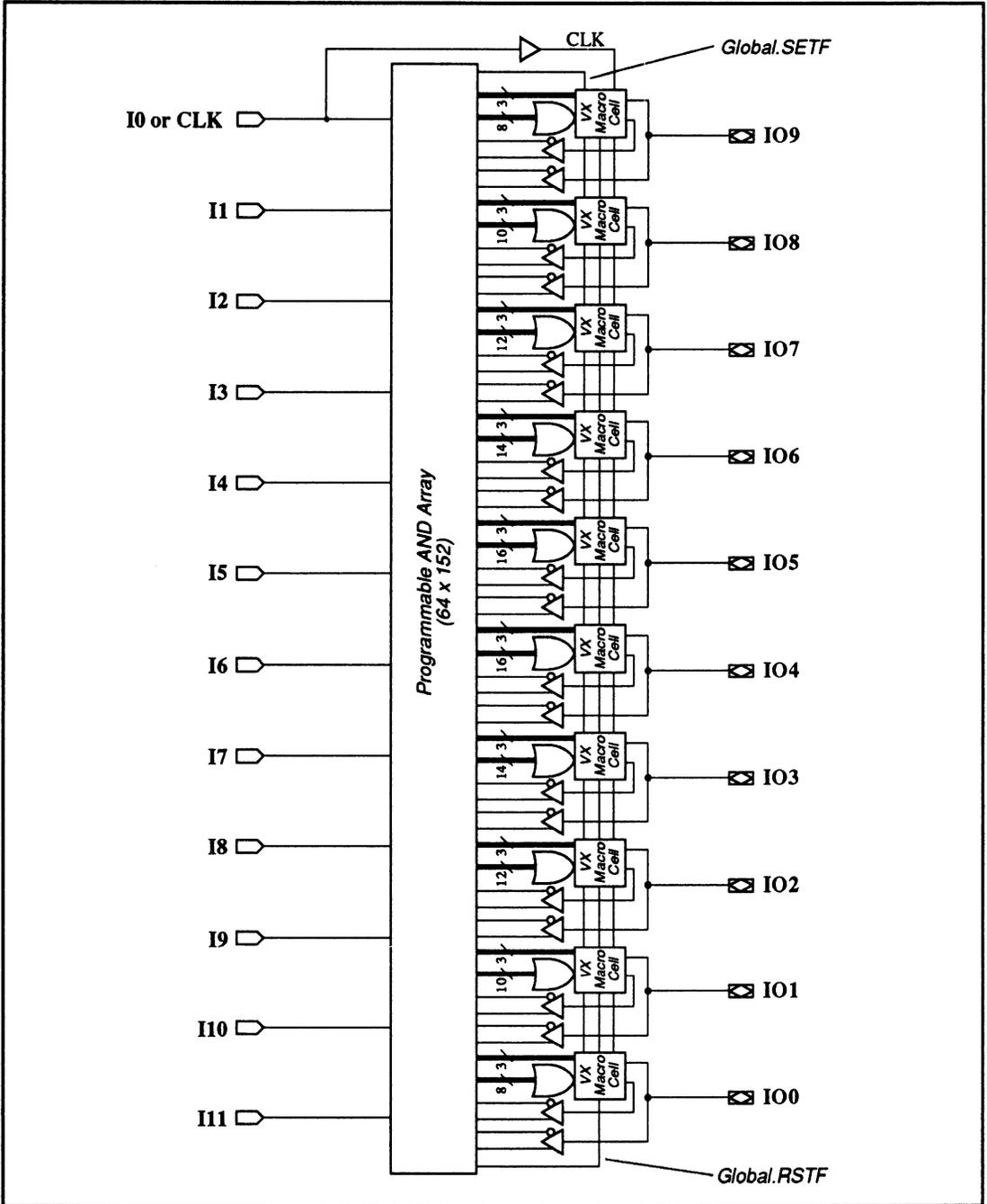
- 24 pins
- 10 buried nodes
- 1 global preset/reset node

32VX10

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	IO/CLK	-	-	-
2 - 11	I1 - I10	-	-	-
12	GND	-	-	-
13	I11	-	-	-
14	IO0 - IO9	2 - 11	R0 - R10	Buried nodes
24	VCC	-	-	-
-	-	1	GLOBAL	Global Preset and reset

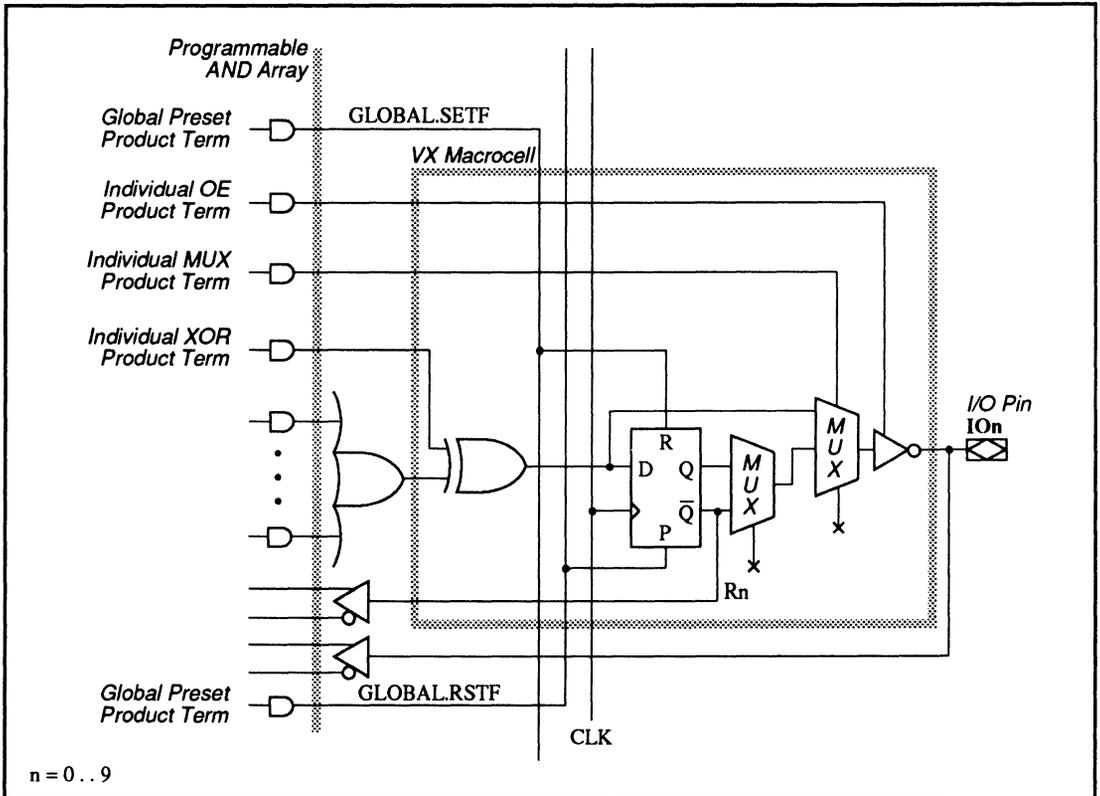
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.



32VX10: Block Diagram Showing Pin and Macrocell Locations

32VX10: PAL32VX10



32VX10: Macrocell Diagram

SPECIAL PROGRAMMING FEATURES

- Bypassable register
- Output with I/O feedback
- Output with /Q feedback
- Output with I/O and /Q feedback
- Buried /Q feedback

Bypassable Register

In each macrocell, there is a MUX product term available to control the dynamic multiplexing between the combinatorial and registered output. When the product term is asserted, the output register is bypassed. To program the MUX product term, use the PALASM syntax shown next.

Syntax

```
Pin Statement(s) :
                  PIN I/O_pin_location  I/O_pin_name
```

```
Equation(s)      I/O_pin_name.CMBF
                  :
                  for dynamic bypassable register control
                    = Boolean expression with one product term
                  :
or               for combinatorial output
                  = VCC
                  :
or               for registered output
                  = GND
```

Note: The .CMBF functional equation overrides the storage type declaration in the pin statement.

Example

Dynamic bypassable register control

```
:
PIN  2      I1          ;input
PIN  3      I2          ;input
:
PIN  21     I07         ;output
PIN  22     I08         ;output
PIN  23     I09         ;output
:
I07.CMBF = VCC          ;combinatorial output
I08.CMBF = GND          ;registered output
I09.CMBF = I1 * /I2     ;dynamic MUX output
                        ;when (I1*/I2) is true, the output register is bypassed
```

Output with I/O Feedback

Each macrocell can be configured to have either a combinatorial or registered output with the output feeding back to the logic array. The language syntax for such configuration is shown next.

32VX10: PAL32VX10

Syntax

```
Pin Statement(s)  PIN  I/O_pin_location  I/O_pin_location  Storage_type*
```

```
Equation(s)      I/O_pin_name** = Boolean expression
                  :
                  <Equation(s) using I/O_pin_name as feedback >
```

* The .CMBF equation will override any storage_type declaration in the pin statement.

** When the output is combinatorial and active high, the XOR function is not allowed in the output equation.

	<u>Output Polarity</u>	
<u>Output Configuration</u>	<u>Active-high</u>	<u>Active-low</u>
Combinatorial	XOR function not used	
Registered		

Example

Combinatorial and registered output with I/O feedback

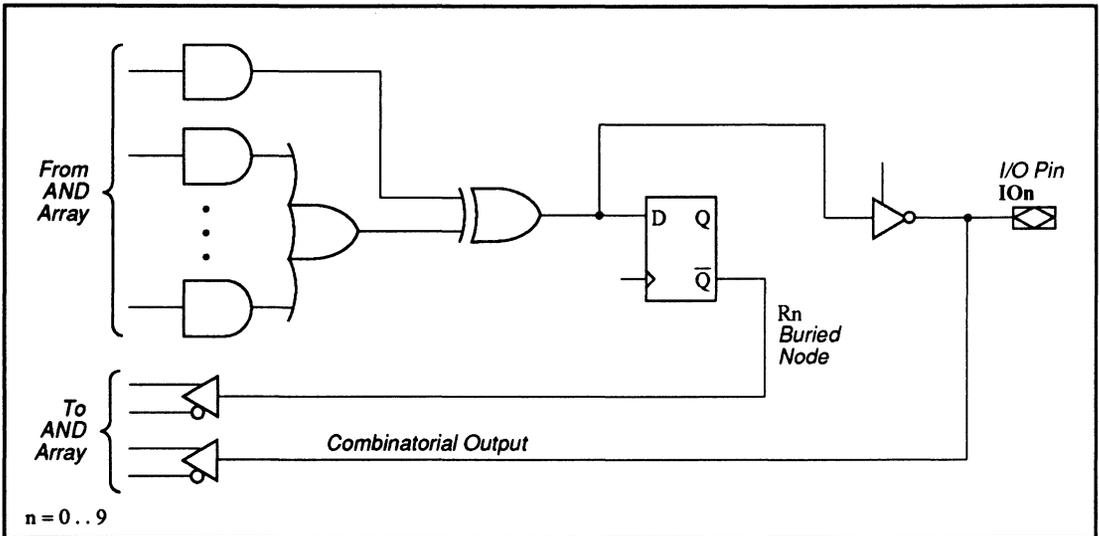
```
:
PIN    2    I1
PIN    3    I2
PIN   14   I00
PIN   15   I01
PIN   16   I02
PIN   17   I03
:
I00.CMBF = VCC           ;combinatorial output
I01.CMBF = VCC           ;combinatorial output
I00 = I1 * I2            ;XOR not allowed in active-high combinatorial output
/I01 = I1 :+: I2        ;XOR allowed in active-low combinatorial output
I02.CMBF = GND          ;registered output
I03.CMBF = GND          ;registered output
I02 = / I2 :+: I1       ;XOR allowed in active-high registered output
/I03 = I2 :+: (/I1*I00*/I02) ;XOR allowed in active-low registered output using
                           ; combinatorial and registered outputs I00 and
                           ; I02 as feedbacks
```

Output with /Q Feedback

Each macrocell can be configured to have either a combinatorial or registered output with the /Q output of the register feeding back to the logic array. The language syntax for this configuration is similar to the next configuration, output with /Q and I/O feedbacks. In this case only the /Q output and not the I/O is used in the output equations.

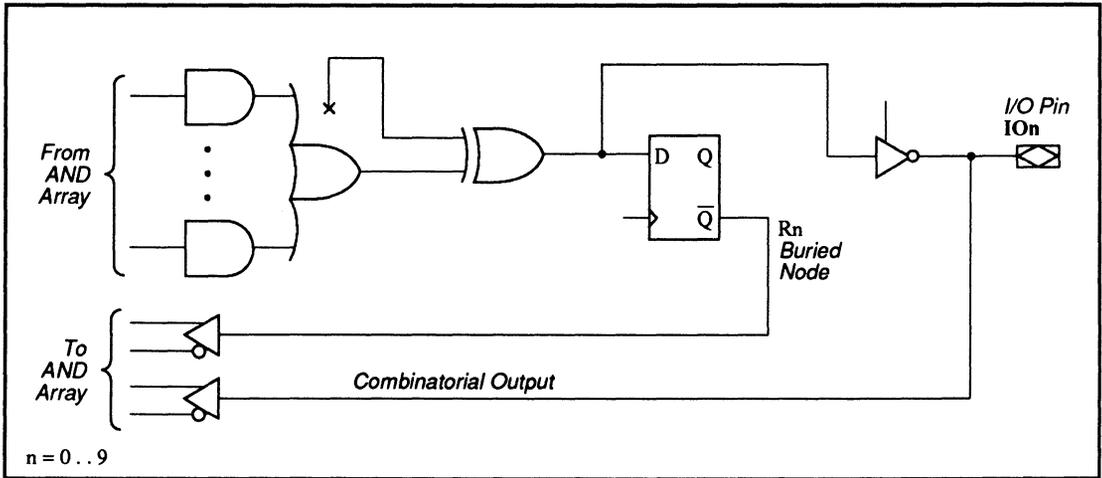
Output with I/O and /Q Feedbacks

Each macrocell can be configured to have either a combinatorial or registered output with both the /Q output of the register and the output itself feeding back to the logic array. The language syntax for both combinatorial and registered outputs is shown separately below.



32VX10: Combinatorial Output with /Q and I/O Feedback Where XOR Function is Available

32VX10: PAL32VX10



32VX10: Combinatorial Output with /Q and I/O Feedback Where XOR Function is Not Allowed

Syntax

Combinatorial output with /Q and I/O feedbacks

Pin Statement(s)	PIN	I/O_pin_location	I/O_pin_location	COMB*
	NODE	Buried_node_location	Buried_node_name	REG
Equation(s)	I/O_pin_name**	=	Boolean expression	
	Buried_node_name***	=	{ I/O_pin_name }	
	:		<Equation(s) using Buried_node_name, or I/O_pin_name, or both as feedback(s)>	

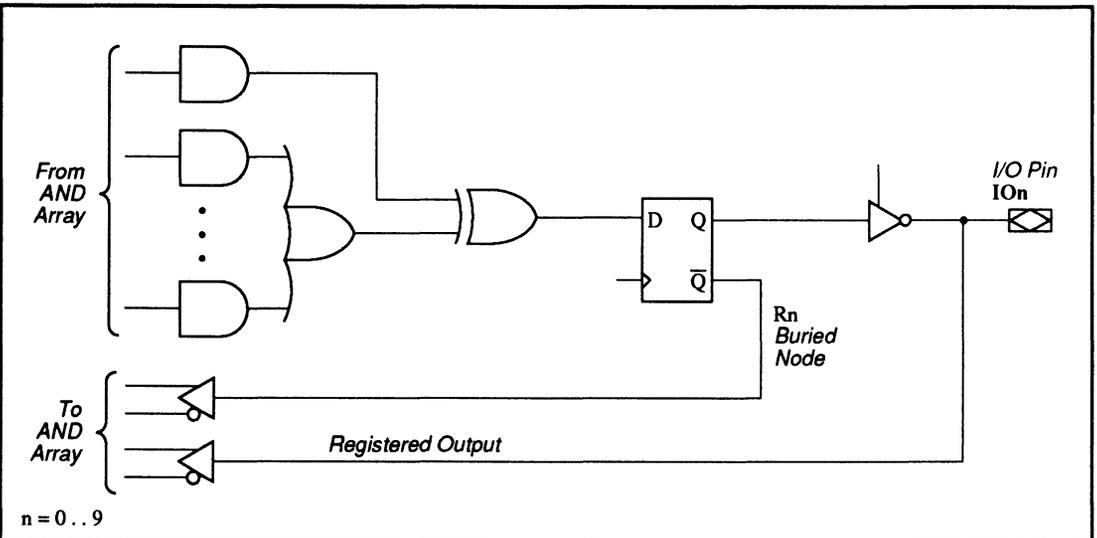
* The .CMBF equation will override any storage_type declaration in the pin statement.

** For combinatorial outputs, only active-high equations are allowed.

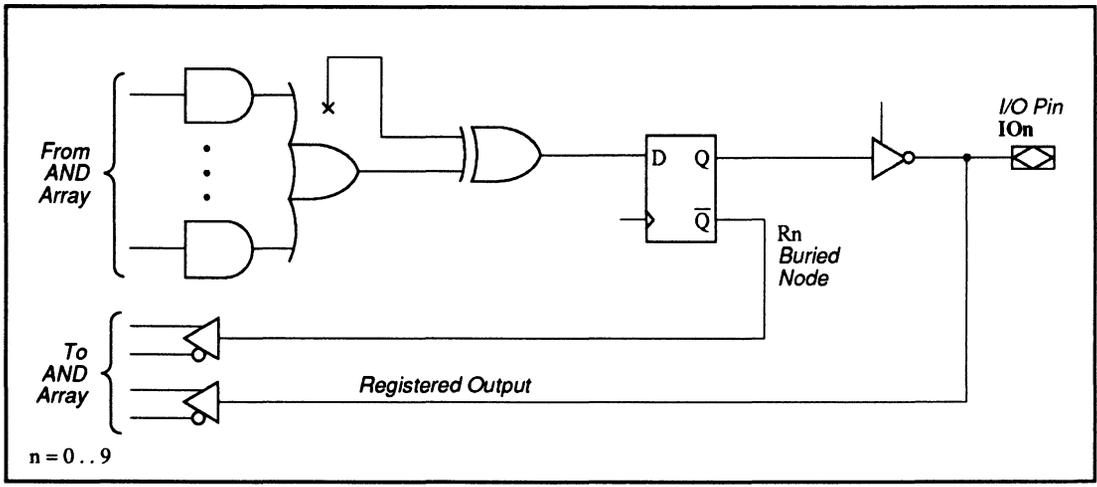
*** For combinatorial outputs, XOR function is only allowed when the output equation is active high and the Buried_node_name is active low.

	Combinatorial Output Polarity	
Buried_node_name polarity	Active-high	Active-low
Active-high		Illegal
Active-low	XOR allowed	Illegal

32VX10: PAL32VX10



32VX10: Registered Output with /Q and I/O Feedback Where XOR Function is Available



32VX10: Registered Output with /Q and I/O Feedback Where XOR Function is Not Allowed

32VX10: PAL32VX10

Syntax

Registered output with /Q and I/O feedback

<i>Pin Statement(s)</i>	PIN	I/O_pin_location	I/O_pin_location	REG*
	NODE	Buried_node_location	Buried_node_name	REG

<i>Equation(s)</i>	I/O_pin_name	=	Boolean expression
	Buried_node_name**	=	{ I/O_pin_name }
	:		
	<Equation(s) using Buried_node_name, or I/O_pin_name, or both as feedback(s)>		

* The .CMBF equation will override any storage_type declaration in the pin statement.

** For registered outputs, XOR function is only allowed when the Buried_node_name is

active low.

	<u>Registered Output Polarity</u>	
<i>Buried_node_name polarity</i>	<i>Active high</i>	<i>Active low</i>
<i>Active high</i>		
<i>Active low</i>	<i>XOR allowed</i>	<i>XOR allowed</i>

ExampleCombinatorial and registered output with /Q and I/O
feedbacks

```

:
PIN      2      I1          ;input
PIN      3      I2          ;input
PIN     14     I00
PIN     15     I01
PIN     16     I02
PIN     17     I03
PIN     18     I04
NODE      2      R0      REG
NODE      3      R1      REG
NODE      4      R2      REG
:
I00.CMBF = VCC          ;combinatorial output
I01.CMBF = VCC          ;combinatorial output
I00 = I1 * I2           ;XOR not allowed in active-high combinatorial output and
; active-high buried node, R0
R0 = { I00 }
/I01 = I1 :+: I2       ;XOR allowed in active-high combinatorial output and
; active-low buried node, R1
/R1 = { I01 }
I02.CMBF = GND         ;registered output
I03.CMBF = GND         ;registered output
I02 = / I2 :+: I1      ;XOR allowed in active-high registered output and
; active-low buried node, R2
/R2 = { I02 }
/I03 = I2 :+: (/I1*I00*/R2) ;XOR allowed in active-low registered output using
; combinatorial output I01 and buried /Q output R2 as
; feedbacks
/R3 = { /I03 }
I04.CMBF = VCC         ;combinatorial output
I04 = I02 * / R1 * R3  ;using combinatorial output I02 and buried /Q outputs R1
; and R3 as feedbacks

```

32VX10: PAL32VX10

Buried /Q Feedback

The output macrocell can be configured as a buried register with the /Q output feeding back to the logic array. The language syntax for this configuration is shown below.

Syntax

<i>Pin Statement(s)</i>	PIN Buried_node_location	Buried_node_name	REG
-------------------------	--------------------------	------------------	-----

<i>Equation(s)</i>	Buried_node_name*	= Boolean expression
	:	
	<	Equation(s) using Buried_node_name as feedback >

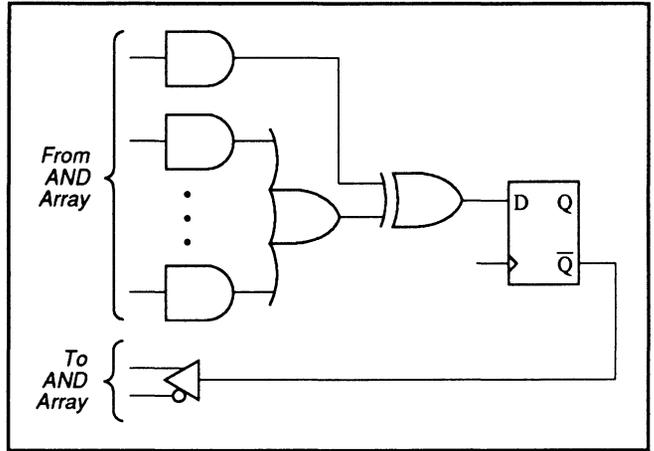
* XOR function is only allowed in active-low buried node equations.

Example

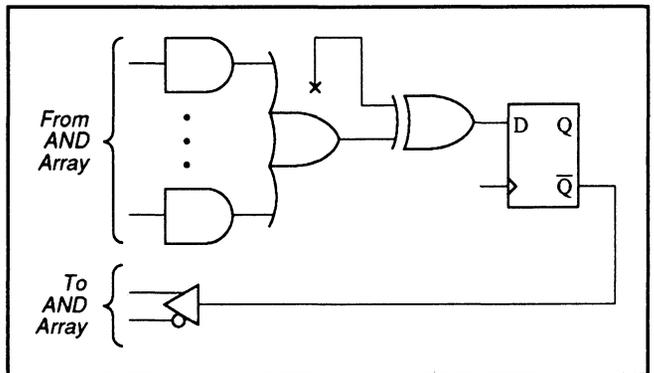
Buried /Q output

```
:
PIN    2    I1           ;input
PIN    3    I2           ;input
PIN    16   IO2
NODE   2    R0    REG
NODE   3    R1    REG
:
R0 = I1 * I2           ;XOR not allowed in active-high node equation
/R1 = I1 :+: I2       ;XOR allowed in active-low node equation
IO2.CMBF = GND        ;registered output
IO2 = I1 * /R0 * /R1  ;using buried /Q feedbacks R0 and R1
```

32VX10: PAL32VX10



32VX10: Buried /Q Feedback Where XOR Function is Available



32VX10: Buried /Q Feedback Where XOR Function is Not Allowed

610: PALCE610

PIN AND NODE DESCRIPTIONS

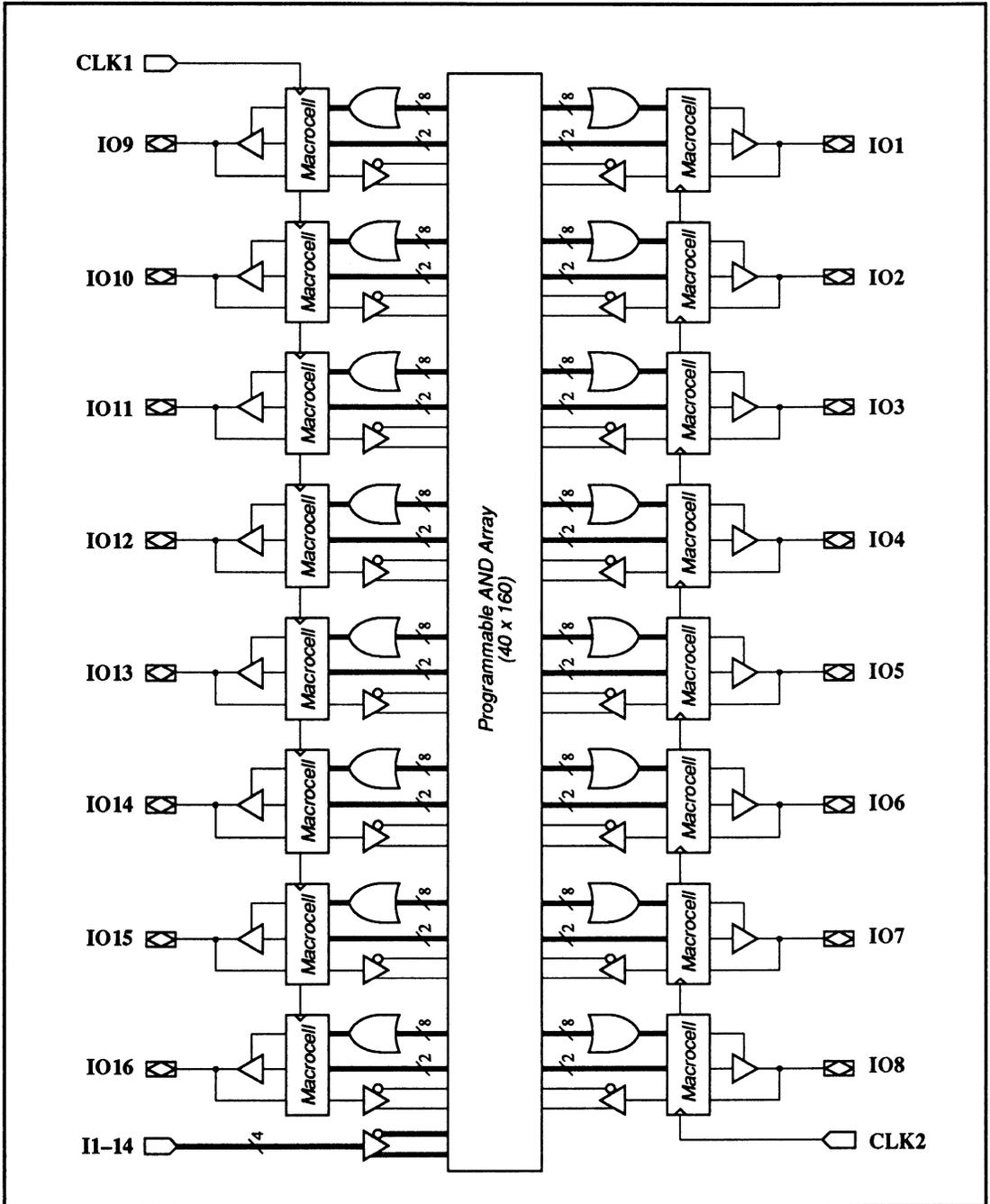
- 24 pins
- 16 buried nodes

610

PIN LOCATION	PIN NAME	NODE LOCATION	NODE NAME	NODE DESCRIPTIONS
1	CLK1	-	-	-
2	I3	-	-	-
3-10	IO9-IO16	1-8	R9-R16	Buried feedback
11	I4	-	-	-
12	GND	-	-	-
13	CLK2	-	-	-
14	I2	-	-	-
15-22	IO8-IO1	9-16	R8-R1	Buried feedback
23	I1	-	-	-
24	VCC	-	-	-

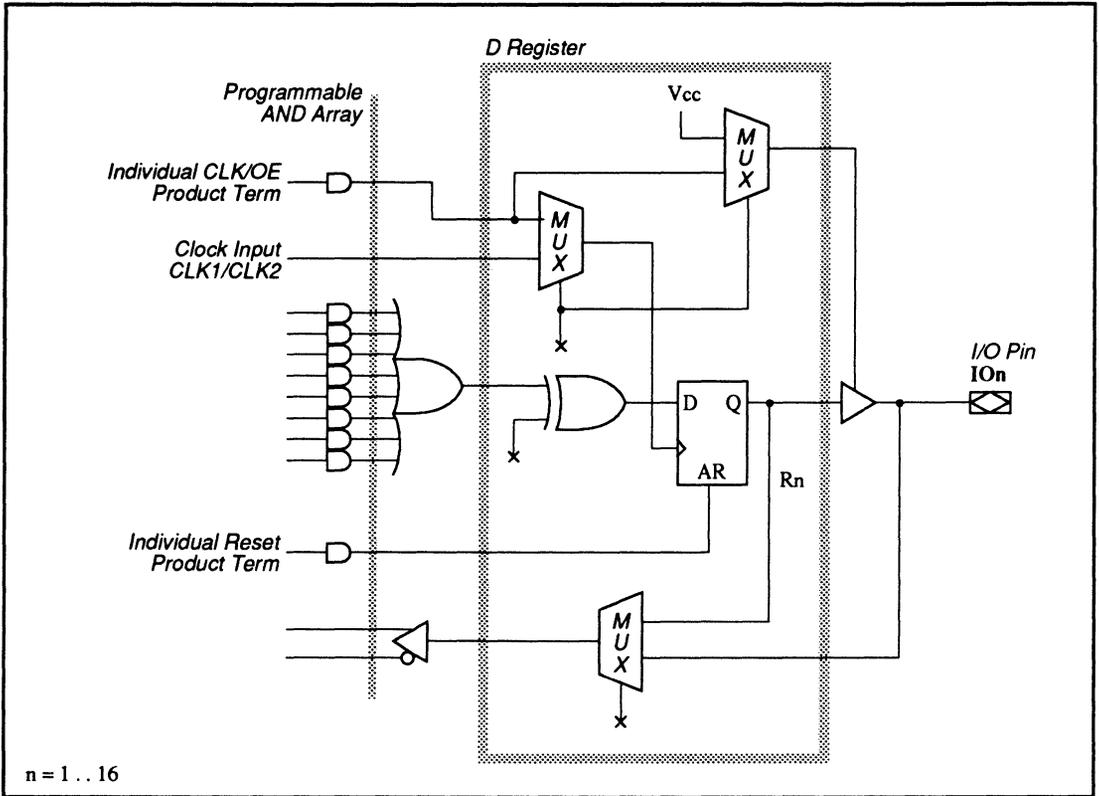
BLOCK AND MACROCELL DIAGRAMS

Block and macrocell diagrams follow.

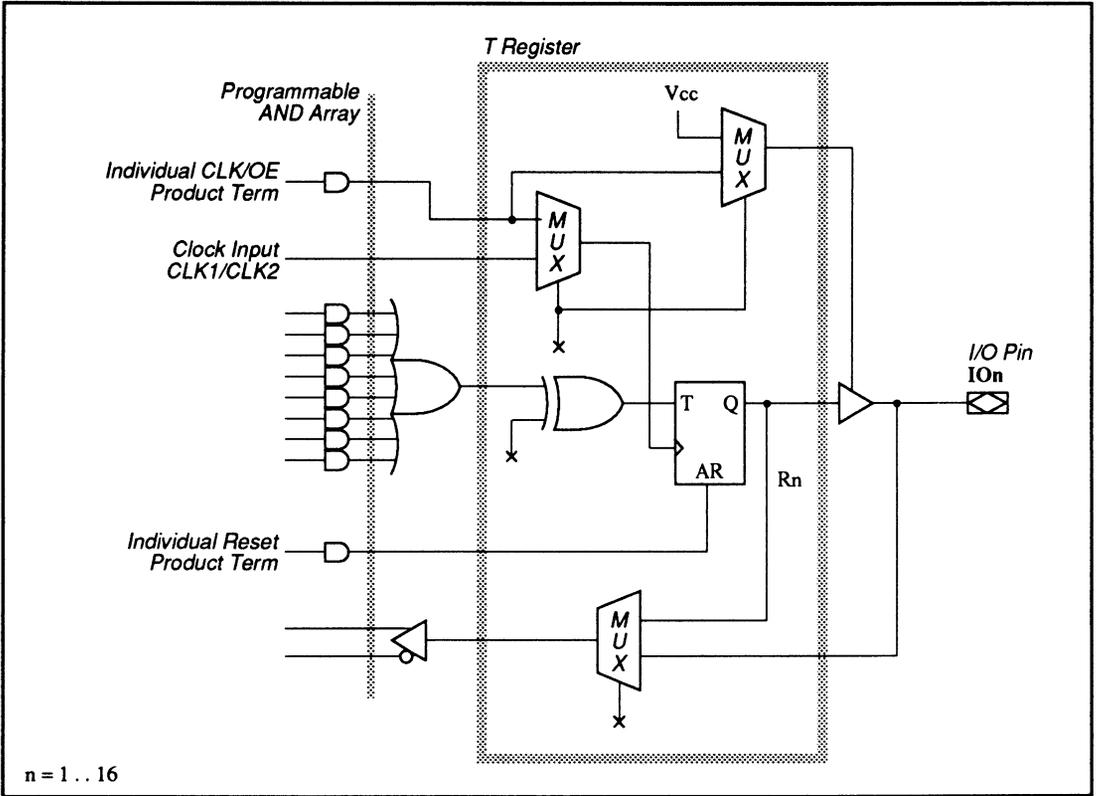


610: Block Diagram Showing Pin and Node Locations

610: PALCE610

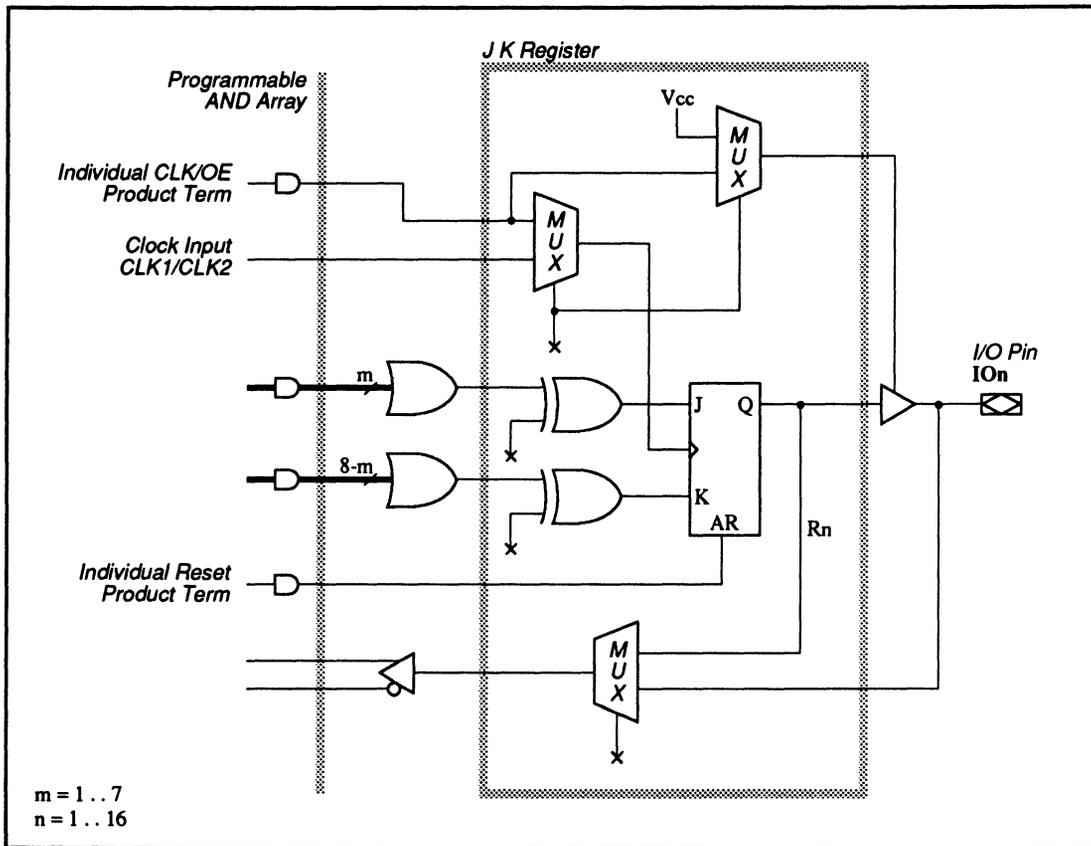


610: D Flip-Flop Macrocell Diagram

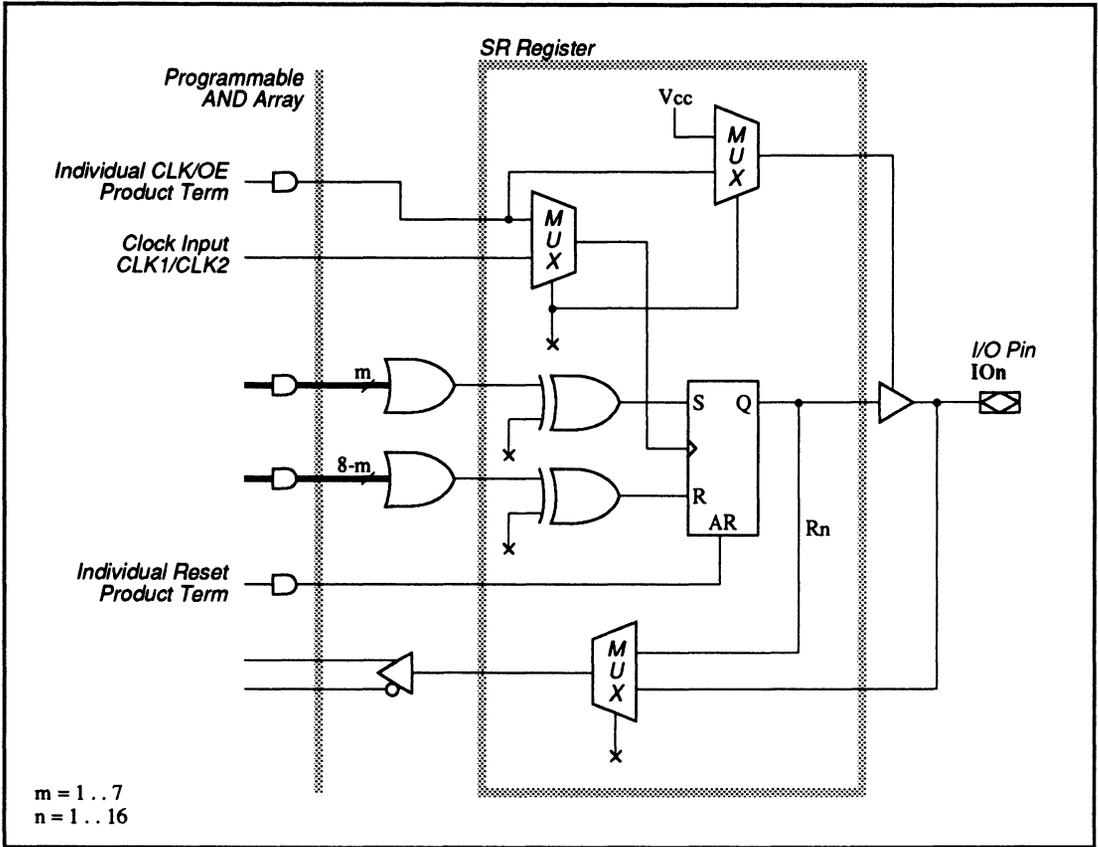


610: T Flip-Flop Macrocell Diagram

610: PALCE610



610: JK Flip-Flop Macrocell Diagram



610: SR Flip-Flop Macrocell Diagram

610: PALCE610

SPECIAL PROGRAMMING FEATURES

- * Clock and output-enable product term control
- * Register configurations
- Macrocells with different configurations
- Output with I/O feedback
- Output with Q feedback
- Buried register with Q feedback

Clock and Output-Enable Product Term Control

Each macrocell can select as its clock either the corresponding clock pin or the CLK/OE product term. If the clock pin is selected, the output enable is controlled by the CLK/OE product term. If the CLK/OE product term is selected to control the clock, the output is always enabled.

To select the CLK/OE product for the clock control, simply write the .CLKF equation for the corresponding output. To select the CLK/OE product term for output-enable control, write the .TRST equation instead. The syntax for both cases is shown next.

Syntax

Pin Statement(s) PIN Output_pin_location Output_pin_name Storage_type

:

Equation(s) for clock control

Output_pin_name.CLKF = Boolean expression with one product term

or for output-enable control

Output_pin_name.TRST = Boolean expression with one product term

610: PALCE610

The table below summarizes how the PALASM 4 software interprets the .CLKF and .TRST equations when there is a conflict. For example, in case 3 for the same output, if the .CLKF equation exists and the .TRST equation also exists and is equal to VCC, then the clock is controlled by the CLK/OE product term, and the output enable is connected directly to VCC.

	.CLKF Equation	.TRST Equation	PALASM software's interpretation
Case 1 output	Exists	-	clock controlled by CLK/OE product term enable is enabled by VCC
Case 2	-	Exists	clock controlled by external clock pin output enable is controlled by CLK/OE product term
Case 3	Exists	Exists and = VCC	clock controlled by CLK/OE product term; output enable is enabled by VCC
Case 4	Exists	Exists and not = VCC	ERROR

Example 1

Using CLK/OE product term to control clock

```
:  
PIN    2    I3        ;input  
PIN    11   I4        ;input  
:  
PIN    3    I09   REG ;output, registered  
:  
I09.CLKF = I3 * /I4    ;clock of I09's register is controlled by the product term  
;    (I3*/I4)
```

610: PALCE610

Example 2

Using CLK/OE product term to control output enable

```
:  
PIN      2    I3      ;input  
PIN     11    I4      ;input  
:  
PIN 3    I09  REG      ;output, registered  
:  
I09.TRST = I3 * /I4    ;output enable of I09's output buffer is controlled by  
;      product term (I3*/I4)
```

Register Configurations

Each register in a 610 can be configured as one of four types of registers with programmable polarity.

- D flip-flop
- T flip-flop
- JK flip-flop
- RS flip-flop

Syntax

```
Pin Statement(s) :  
                PIN I/O_pin_number    I/O_pin_name    REG  
                :
```

```
Equation(s) for D flip-flop  
                I/O_pin_name* = Boolean expression  
or for T flip-flop  
                I/O_pin_name.T* = Boolean expression  
or for JK flip-flop  
                I/O_pin_name.J* = Boolean expression  
                I/O_pin_name.K* = Boolean expression  
or for SR flip-flop  
                I/O_pin_name.R* = Boolean expression  
                I/O_pin_name.S* = Boolean expression
```

* I/O_pin_name can be active high or active low.

Example**Register configurations**

```

:
PIN    23   I1           ;input
PIN    14   I2           ;input
PIN     3   IO9   REG    ;output, registered
PIN     4   IO10  REG    ;output, registered
PIN     5   IO11  REG    ;output, registered
PIN     6   IO12  REG    ;output, registered
:
/IO9 = I1                ;output equation, D FF, active low
IO10.T = I2              ;output equation, T FF, active high
/IO11.S = I1 * /I2       ;output equation, S input of SRFF, active low
IO11.R = /I1             ;output equation, R input of SRFF, active high
/IO12.J = I1 * /I2       ;output equation, J input of JKFF, active low
/IO12.K = /I1           ;output equation, K input of JKFF, active high

```

Macrocells with Different Configurations

The 610 output macrocell can be configured as one of the following types of output.

- Combinatorial
- D registered
- T registered
- JK registered
- RS registered

Each type of output provides a different set of feedback configurations, as tabulated next. Allowable configurations are marked with an X in the table.

610: PALCE610

FEEDBACK CONFIGURATIONS	COMBINATORIAL	D	T	JK	RS
Programmable Feedback					
Output with I/O feedback					
Combinatorial	X	N/A	N/A	N/A	N/A
Registered	N/A	X	X	N/A	N/A
Output with /Q feedback					
Registered	N/A	X	X	X	X
Buried register with /Q feedback	N/A	X	X	X	X

Output with I/O Feedback

If an output is programmed as combinatorial, it can be used as a feedback. If an output is programmed as registered, it can be used as a feedback only if the register is configured as either a D or T flip-flop. It is illegal to use the output as a feedback if the output is configured to be either a JK or SR flip-flop.

To use the output as an I/O with a feedback, just use the I/O pin name for the required output equation, no special language syntax is required.

Output with Q Feedback

The output macrocell can be configured as a registered output with the Q output of the register available as a feedback. If the output is configured as combinatorial, the Q feedback is not allowed.

To use the Q output of the register as a feedback while also using the register as an output, you must first write a transfer equation for the buried node then use the buried node name in the required equation, as shown next.

Syntax

```

Pin Statement(s)  PIN  I/O_pin_location    I/O_pin_name    REG
                  :
                  NODE Buried_node_location    Buried_node_name    REG

```

```

Equation(s)  for D flip-flop
              I/O_pin_name*      = Boolean expression
              Buried_node_name** = { I/O_pin_name*** }
or  for T flip-flop
              I/O_pin_name.T*    = Boolean expression
              Buried_node_name.T** = { I/O_pin_name*** }
or  for JK flip-flop
              I/O_pin_name.J*    = Boolean expression
              I/O_pin_name.K*    = Boolean expression
              Buried_node_name.J** = { I/O_pin_name*** }
              Buried_node_name.K** = { I/O_pin_name*** }
or  for SR flip-flop
              I/O_pin_name.R*    = Boolean expression
              I/O_pin_name.S*    = Boolean expression
              Buried_node_name.R** = { I/O_pin_name*** }
              Buried_node_name.S** = { I/O_pin_name*** }
              :
              < Equation(s) using Buried_node_name as feedback >

```

- * *I/O_pin_name can be active high or active low.*
 - * *Buried_node_name must be the same polarity as the I/O_pin_name defined in the output equation.*
 - *** *I/O_pin_name inside curly bracket must be the same polarity as the I/O_pin_name defined in the output equation.*
-

610: PALCE610

Example

Registered output with Q feedback

```
:
PIN      23  I1    COMB ;input
PIN      14  I2    COMB ;input
PIN      22  I01   REG      ;I/O, registered
:
NODE     16  R1                ;buried node
:
/I01.R = /I1 * I2                ;equation for R input of register I01 with active-low
;    input
I01.S = /I1 * I2                ;equation for S input of register I01 with active-high
;    input
/R1.R = {/I01.R}                ;buried node has same polarity as I01.R
R1.S = { I01.S}                 ;buried node has same polarity as I01.S
I02.T = I2* / I1*R1             ;equation for T input I02 with R1 as feedback
```

Buried Register with Q Feedback

The register in the output macrocell can be configured as a buried register. To use the Q output of the buried register as a feedback, just write an equation for the buried node and use the buried node name in the required equation, shown next.

Syntax

```
Pin Statement(s)  NODE Buried_node_location  Buried_node_name  REG
Equation(s)  for D flip-flop
              Buried_node_name.D* = Boolean expression
              or
              for T flip-flop
              Buried_node_name.T* = Boolean expression
              or
              for JK flip-flop
              Buried_node_name.J* = Boolean expression
              Buried_node_name.K* = Boolean expression
              or
              for SR flip-flop
              Buried_node_name.R* = Boolean expression
              Buried_node_name.S* = Boolean expression
              :
              < Equation(s) using Buried_node_name as feedback >
```

* *Buried_node_name can be active-high or active-low*

Example**Buried register with Q feedback**

```
:
PIN      23  I1   COMB      ;input
PIN      14  I2   COMB      ;input
PIN      21  I02  COMB      ;output, combinatorial
:
NODE     16  R1   REG        ;buried node of buried register
:
/R1.J = /I2 * I1           ;equation for J input of register I01
/R1.K = I2 * I02          ;equation for K input of register I01
I02 = I2* / I1*R1         ;equation for I02 with R1 as feedback
```

11.5 MACH 1 AND MACH 2 SERIES DEVICES

This topic provides device and language information related to programming the MACH 1 and 2 series devices.

- The overview, 11.5.1, introduces the features.
- The discussion on sample equations, 11.5.2, provides examples of how to use the PALASM language syntax to configure macrocells.

Important: Unless otherwise stated, all discussions in this chapter pertain to all MACH 1 and 2 series devices.

Note: The term MACH 1 series refers to the MACH 110, 120, and 130 devices. The term MACH 2 series refers to the MACH 210, 220, and 230 devices.

11.5.1 OVERVIEW

This overview includes three discussions.

- 11.5.1.1, Device Features
- 11.5.1.2, Pin and Node Descriptions
- 11.5.1.3, PALASM Programming Features

11.5.1.1 Device Features

The MACH 1 and 2 series devices are similar; however, the MACH 2 devices are larger and contain both output and buried macrocells.¹⁷

- MACH 1 series devices are generally best suited for I/O-intensive designs.
- MACH 2 series devices are generally best suited for logic-intensive designs.

¹⁷ Refer to the *AMD High Density EE CMOS Programmable Logic MACH™ Devices Data Book* for details about each device.

11.5 MACH 1 AND MACH 2 SERIES DEVICES

This topic provides device and language information related to programming the MACH 1 and 2 series devices.

- The overview, 11.5.1, introduces the features.
- The discussion on sample equations, 11.5.2, provides examples of how to use the PALASM language syntax to configure macrocells.

Important: Unless otherwise stated, all discussions in this chapter pertain to all MACH 1 and 2 series devices.

Note: The term MACH 1 series refers to the MACH 110, 120, and 130 devices. The term MACH 2 series refers to the MACH 210, 215, 220, and 230 devices.

11.5.1 OVERVIEW

This overview includes three discussions.

- 11.5.1.1, Device Features
- 11.5.1.2, Pin and Node Descriptions
- 11.5.1.3, PALASM Programming Features

11.5.1.1 Device Features

The MACH 1 and 2 series devices are similar; however, the MACH 2 devices are larger and contain both output and buried macrocells.¹⁷

- MACH 1 series devices are generally best suited for I/O-intensive designs.
- MACH 2 series devices are generally best suited for logic-intensive designs.
- The MACH215 is an asynchronous device.

¹⁷ Refer to the *AMD High Density EE CMOS Programmable Logic MACH™ Devices Data Book* for details about each device.

This difference in architecture also affects product-term steering.¹⁸

The following table summarizes the I/O, block, and macrocell features of each device.

FEATURE	MACH DEVICE						
	110	120	130	210	220	230	215
Input Pins	4	8	6	4	8	6	4
Clocks/Input Pins	2	4	4	2	4	4	2*
Input/Output Pins	32	48	64	32	48	64	32
Blocks	2	4	4	4	8	8	4
Output Macrocells	32	48	64	32	48	64	32
Buried Macrocells	0	0	0	32	48	64	32**

* The MACH215 has dedicated clock pins. All other input and I/O pins can drive PT clocks.

** The MACH215 buried macrocells are available for input register use only.

11.5.1.2 Pin and Node Descriptions

You can specify the pin or node location of a signal in the design file.¹⁹ Following discussions provide illustrations of available pins and nodes for each device.

11.5.1.3 PALASM Programming Features

Discussions below illustrate the PALASM programming features for the MACH 1 and 2 series devices, and are divided as follows.

- Fixing Pin and Node Locations
- Pairing Pins and Nodes
- Steering Product Terms

¹⁸ Refer to the discussion on steering product terms, in this chapter, for more information.

¹⁹ Refer to Section II, Chapter 4, for details about the various design-entry methods supported for MACH-device designs. Also, refer to Chapter 10, in this section, for details about MACH-specific syntax.

- Steering Product Terms
- Splitting Functions
- Programming Flip-Flop Types
- Optimizing with D- or T-Type Flip-Flops
- Defining Preset and Reset
- Defining Clock Pins
- Defining Output Enable

Fixing Pin and Node Locations

You can assign **fixed** pin and node locations or leave them **floating**.

- You **float** a pin or node by placing a question mark, **?**, in the location field of a pin or node statement, as shown below.

```
PIN      ?      <pin name>
NODE    ?      <node name>
```

- Or, you can force all locations to float using the corresponding command on the MACH Fitting Options form.

Each floating pin and node is assigned a location during compilation and fitting. The location is based on optimal use of the device.

Recommendation: It is best to float all pin and node locations.

Also: If you float some locations and fix others, a no-fit situation may occur. Consider whether your design is close to the device-resource limit. If it is, you may have to sacrifice fixed assignments to fit the design in the device.

- You assign a **fixed** pin or node location by specifying a number in the location field of a pin or node statement using the following syntax.

```
PIN      3      <pin name>
NODE    2      <node name>
```

Important: Fixed locations may limit available resources, which affects both product-term steering and gate splitting.²⁰

The group statement allows you to assign pins and nodes to a specific block. However, the only way to preserve the order is to assign fixed locations. The following example assigns three pins and three nodes to block A of a MACH device, using the reserved word MACH_SEG_A as a group name.²¹

```
GROUP MACH_SEG_A IO0 IO1 IO2 A0 A1 A2
```

Pairing Pins and Nodes

You can use the optional Pair attribute in a pin or node statement to direct input or output pairing manually.²² Pairing can save resources when pins and nodes are left floating. If a node and a pin have the same equation, pairing them eliminates the need to generate the same equation twice.

- **Input** pairing includes the Pair attribute in a pin statement to associate an input pin with a node logically.

Input pairing can only be implemented in MACH 2 series devices since they have buried macrocells.

- **Output** pairing includes the Pair attribute in a node statement to associate a node with an output pin logically.

Output pairing can be implemented in all MACH devices.

²⁰ Refer to discussions on steering product terms and splitting gates, in this chapter, and to Section II, Chapter 5, for more information.

²¹ Refer to Chapter 10, in this section, for details about the group name MACH_SEG_block.

²² Refer to Chapter 10, in this section, for information on input and output pairing.

The keywords OPAIR and IPAIR are also valid and denote output and input pairing, respectively. To enable manual pairing, enter the letter N beside the Use automatic pin/node pairing field of the Logic Synthesis Options form.

Recommendation: Use the default setting that allows the software to establish pairs automatically.

Steering Product Terms

Four product terms are available to each macrocell. Product-term steering is an automatic software process that borrows additional terms from **unused** adjacent macrocells.

Note: Product terms can be borrowed only in groups of four, so an equation that requires five product terms actually consumes two groups of four.

In MACH 1 series designs, up to eight product terms can be borrowed from adjacent macrocells for a total of 12. Four product terms can be borrowed from the adjacent macrocells above and below. However, the end macrocells in a block are limited to four borrowed product terms. For a MACH 110 device, an end macrocell occurs at the boundary of each bank of eight cells. For example, for block A, the nodes 2, 9, 10, and 17 originate in end macrocells. For a MACH 130 device, an end macro cell occurs at the end of each block of 16 cells. For example, nodes 2 and 17 originate in end macrocells.

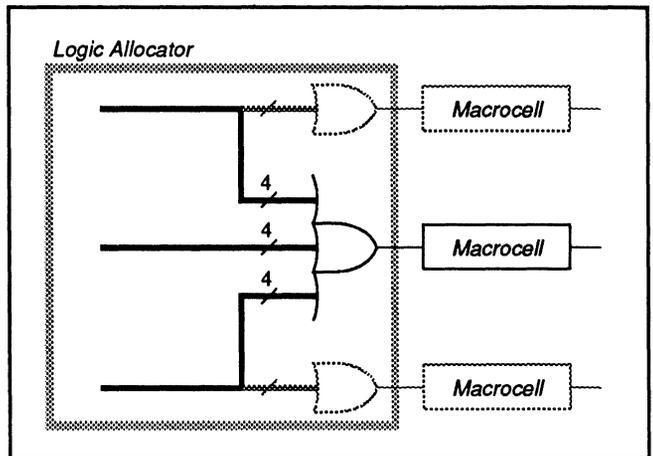
In MACH 2 series designs, up to 12 product terms can be borrowed from adjacent macrocells for a total of 16. Four product terms can be borrowed from the one adjacent macrocell above and eight from the two adjacent macrocells below. However, the end macrocells in a block are limited to four or eight borrowed product terms, depending on whether it is at the bottom or top of the block, respectively. For a MACH 210 device, an end macrocell occurs at the boundary of each block of 16 macrocells. For example,

for block A, the nodes 2 and 17 originate in end macrocells.

Note: Product-term steering does not result in additional delays in the signal path.

Reminder: In product-term steering, the first and last macrocells of a block cannot use product terms from the first or last macrocell of an adjacent block.

The figure below illustrates full product-term steering for a MACH 1 series device.



MACH 1 Series Full PT Steering

The next figure shows an example of full product-term steering for a MACH 2 series device. In this case, four product terms are borrowed from the macrocell above and four each from the two macrocells below.

Each loop back through the array results in an additional level of delay in the signal path.

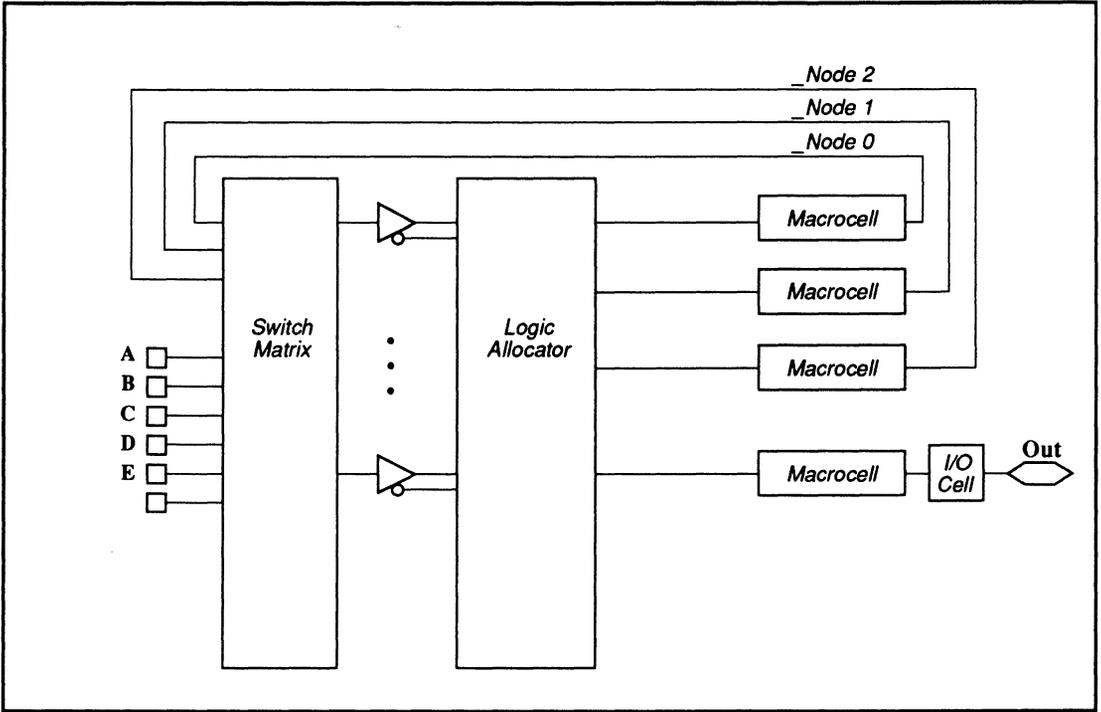
By default, the fitting process first attempts product-term steering. If this cannot be accomplished, gate splitting is used.

You enable automatic gate splitting by entering a Y in the Use automatic gate splitting field and defining the cluster size in the Max= field of the Logic Synthesis Options form.

Simulation shows no difference between product-term steering and gate splitting in waveform or trace files; it does not show propagation delays or gate widths.²³

The next figure illustrates a design with 16 product terms implemented with automatic gate splitting enabled and a maximum cluster size of four.

²³ Refer to Section II, Chapter 6, for details about simulation.



Gate Splitting For MACH 110

Pin Declaration Segment

```

PIN      ?   A      ;INPUT
PIN      ?   B      ;INPUT
PIN      ?   C      ;INPUT
PIN      ?   D      ;INPUT
PIN      ?   E      ;INPUT
PIN      ?   OUT    ;OUTPUT

```

Equation Segment

```

OUT = A :+: B :+: C :+: D :+: E

```

After automatic gate splitting, equations are generated for the exclusive-OR function, as shown next. You can view these equations by compiling the design, disassembling the intermediate file, and then viewing the <design>.pl2 file.

Equation Segment

```
OUT      =  /A * B * /C * D * E
          +  /A * B * C * /D * E
          +  /A * B * C * D * /E
          +  A * B * C * D * E
          +  _NODE0
          +  _NODE1
          +  _NODE2
_NODE0   =  A * B * C * /D * /E
          +  A * B * /C * D * /E
          +  A * B * /C * /D * E
          +  A * /B * C * D * /E
_NODE1   =  A * /B * C * /D * E
          +  A * /B * /C * D * E
          +  A * /B * /C * /D * /E
          +  /A * /B * C * D * E
_NODE2   =  /A * /B * C * /D * /E
          +  /A * /B * /C * /D * E
          +  /A * /B * /C * D * /E
          +  /A * B * /C * /D * /E
```

Programming Flip-Flop Types

Output macrocells are individually defined as either combinatorial, latched,²⁴ or registered by using the optional storage attribute in a pin or node statement.

If you specify registered output, the flip-flop can be programmed as **either** a D-type **or** T-type. D-type is the default. The syntax for both types is shown below.

```
IO1      = Boolean expression      ;D-type is default
IO1.T    = Boolean expression      ;T-type is specified
```

Optimizing with D- or T-Type Flip-flops

The software can determine whether the design could be implemented more efficiently using one type flip-flop over the other. To accomplish this, you specify the following on the Logic Synthesis Options form.

```
Optimize registers for D/T type      Best type for device
```

The software compares the minimized results of a D-type and T-type implementation. The result with the lowest resource usage is chosen for each equation.

²⁴ Macrocell latch has an active-low latch enable input.

Important: You can also specify the flip-flop type in an equation. However, the Optimize registers for D/T-type option overrides the design file specifications. This option allows you to use the type(s) specified in the design file, change all to D, change all to T, or use the best type for the device.

Defining Preset and Reset

Each block in a MACH device has a single asynchronous set and reset line. You can enable each preset and reset at the macrocell level using a .SETF or .RSTF equation. For flip-flop and latch²⁵ configurations, both set and reset definitions must be provided to clear the "No Set/Reset initialization function found!" warning in the report file.

Note: MACH devices provide programmable polarity between the macrocell output and the pin. If you select the Ensure polarity after minimization is Best For Device option on the Logic Synthesis Options form, the logic polarity at the pin may be inverted.

To assign a preset or reset signal to the entire device, use node 1 to define the global asynchronous preset and reset product terms. In this case, you can assign a descriptive name, such as GLOBAL, to node 1, then write either the desired preset or reset equation using the following general form.

Important: You must specify node 1 in the pin declarations segment if you use the global set or reset features. When back annotating, make sure the global node has not been converted to floating.

Recommendation: Keep set/reset functions simple. Each input for these functions uses an array input that would otherwise be available for other logic.

²⁵ This applies to MACH 2 series latches with active-low latch inputs only.

Preset

NODE 1 GLOBAL
GLOBAL.SETF = <preset equation>

Reset

NODE 1 GLOBAL
GLOBAL.RSTF = <reset equation>

The example below initializes all registers to 1, high, when inputs I1 = 1, I2 = 0, and I3 = 1, for positive polarity in the pin and equation declaration segments.

GLOBAL.SETF = I1 * /I2 * I3

The preset and reset functions are asynchronous; therefore, the registers are initialized independent of the clock.

Defining Clock Pins

The following table summarizes the clock pins available for each MACH device.

MACH 110 AND 210	MACH 120 AND 220	MACH 130 AND 230
CLK0, pin 13 CLK1, pin 35*	CLK0, pin 15 CLK1, pin 17 CLK2, pin 49 CLK3, pin 50*	CLK0, pin 20 CLK1, pin 23 CLK2, pin 62 CLK3, pin 65*

* Default clock if you do not specify a clock pin

Defining Output Enable

Each I/O cell has a three-state buffer that can be

- permanently enabled, or
- permanently disabled, or
- controlled by a product term.

The following output-enable equation examples show the three types of output enables.

Permanently Enabled

<pin name>.TRST = VCC ;

Preset

NODE 1 GLOBAL
GLOBAL.SETF = <preset equation>

Reset

NODE 1 GLOBAL
GLOBAL.RSTF = <reset equation>

The example below initializes all registers to 1, high, when inputs I1 = 1, I2 = 0, and I3 = 1, for positive polarity in the pin and equation declaration segments.

GLOBAL.SETF = I1 * /I2 * I3

The preset and reset functions are asynchronous; therefore, the registers are initialized independent of the clock.

Defining Clock Pins

The following table summarizes the clock pins available for each MACH device.

MACH 110 AND 210	MACH215	MACH 120 AND 220	MACH 130 AND 230
CLK0, pin 13 CLK1, pin 35*	CLK0, pin 13* CLK1, pin 35	CLK0, pin 15 CLK1, pin 17 CLK2, pin 49 CLK3, pin 50*	CLK0, pin 20 CLK1, pin 23 CLK2, pin 62 CLK3, pin 65*

* Default clock if you do not specify a clock pin

Defining Output Enable

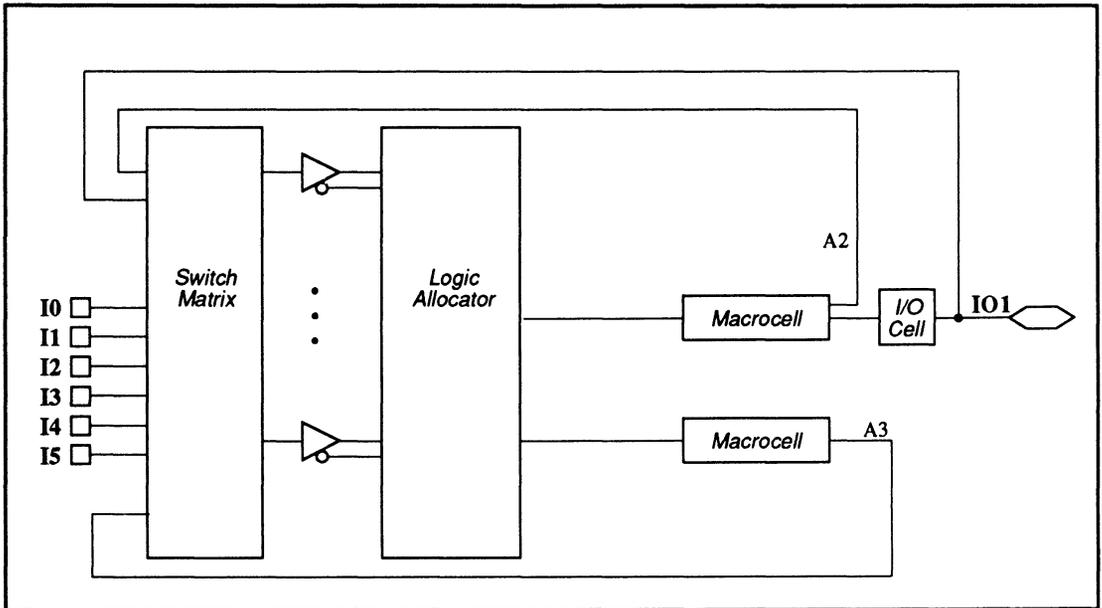
Each I/O cell has a three-state buffer that can be

- permanently enabled, or
- permanently disabled, or
- controlled by a product term.

The following output-enable equation examples show the three types of output enables.

Permanently Enabled

<pin name>.TRST = VCC ;



MACH Resource Organization

11.5.2.1 I/O Cell and Macrocell

You can choose one of the following types of configurations for a pin, output macrocell node, or buried macrocell node.

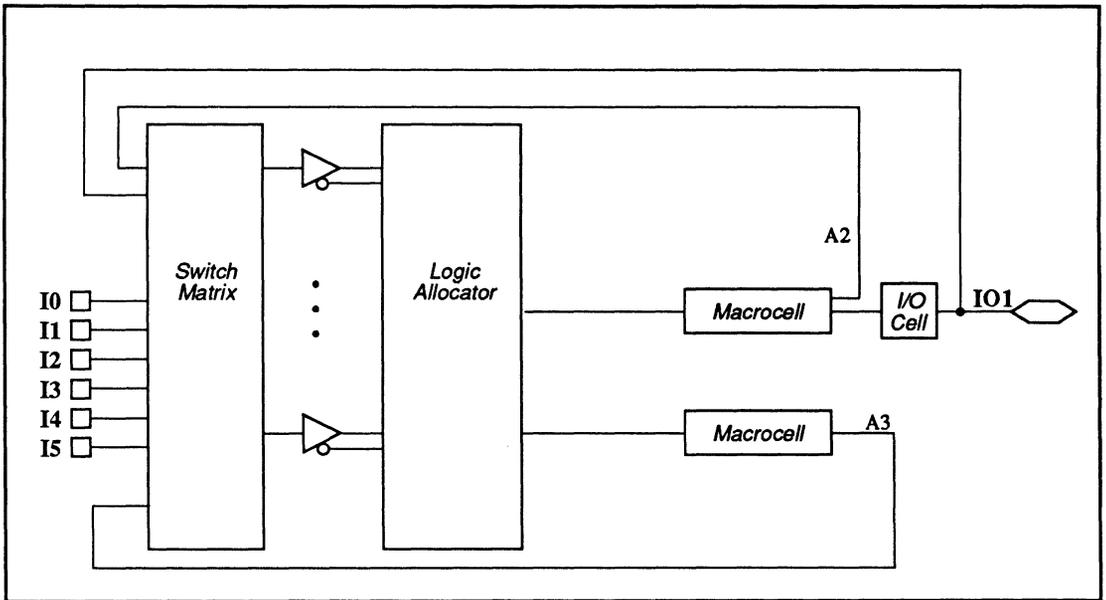
- Combinatorial
- Latched (active-low latch input)
- D-registered
- T-registered

You specify combinatorial, latched, or registered in the pin declaration segment of the design file, as follows.

PIN DECLARATION SEGMENT

Pin	4	IO2	COMBINATORIAL
Node	5	A3	REGISTERED
Pin	5	IO3	LATCHED

For **registered** nodes and pins, you must also specify the type of register in the equations segment of the design file, as follows.



MACH Resource Organization

11.5.2.1 I/O Cell and Macrocell

You can choose one of the following types of configurations for a pin, output macrocell node, or buried macrocell node.

- Combinatorial
- Latched (active-low latch input)
- D-registered
- T-registered

You specify combinatorial, latched, or registered in the pin declaration segment of the design file, as follows.

PIN DECLARATION SEGMENT

Pin	4	IO2	REGISTERED
Node	5	A3	REGISTERED
Pin	5	IO3	LATCHED

For **registered** nodes and pins, you must also specify the type of register in the equations segment of the design file, as follows.

EQUATIONS SEGMENT

IO2 = I1 * I2 * I3 ;D-type, active high
IO2.D = I1 * I2 * I3 ;D-type, active high
IO2.T = I1 * I2 * I3 ;T-type, active high

Note: For a T-registered signal, you must use the .T suffix in string definitions. For example:

A6.T: = {IO2.T}

11.5.2.2 Pin and Node Feedback

A MACH device allows you to specify feedback from a pin or a node. The following examples illustrate equations for I/O pin, output macrocell, and buried macrocell feedback.

Important: Since the MACH device emulates 22V10 behavior, feedback is normally taken from the register instead of the I/O pin, unless you specify otherwise. To realize pin feedback for a **registered** or **latched** output, you must specify an equation for the associated node, even if you do not use the node in the design. If you do not define the node, the configuration will default to feedback from the node. However, this is not true if the output pin is IPAIReD, in which case feedback defaults to I/O pin feedback, without the necessity of writing a dummy equation.

To realize pin feedback for a **combinatorial** output, you do not need to specify a node equation.

I/O Pin Feedback

The examples below are divided into two categories: combinatorial and registered, or latched outputs.

Combinatorial Output

IO1 = I1 * IO2 * I3 ;feedback from pin IO2

Registered or Latched Outputs

PIN DECLARATION SEGMENT

```
Pin      ?    IO1  REG
Node     ?    A4   PAIR  IO1   REG
```

EQUATIONS SEGMENT

```
IO1      =    I1 * IO2 * I3      ;feedback from pin IO2
A4       =    I1 * IO2 * I3      ;mandatory node eqn.
```

Node Feedback

You specify feedback from an output macrocell or buried macrocell node as shown below.

Output Macrocell Node

```
IO1 = I1 * A2 * I3      ;feedback from output macrocell
                        ; node A2
```

Buried Macrocell Node

```
IO1 = I1 * A3 * I3      ;feedback from buried macrocell
                        ;node A327
```

11.5.2.3 Registered and Latched Inputs

The following examples illustrate equations for latched and registered inputs.²⁸

Registered Inputs

PIN DECLARATION SEGMENT

```
Pin      33   I4   PAIR A3
Node     5    A3   REGISTERED
```

EQUATION SEGMENT

```
A3 = I4      ;D-type registered input
```

Note: T-type registered inputs are not supported.
--

Latched Inputs (active-low latch Input)

PIN DECLARATION SEGMENT

²⁷ This applies to MACH 2 series devices only.

²⁸ This applies to MACH 2 series devices only.

Pin	33	I4	PAIR A3
Node	5	A3	LATCHED

EQUATION SEGMENT

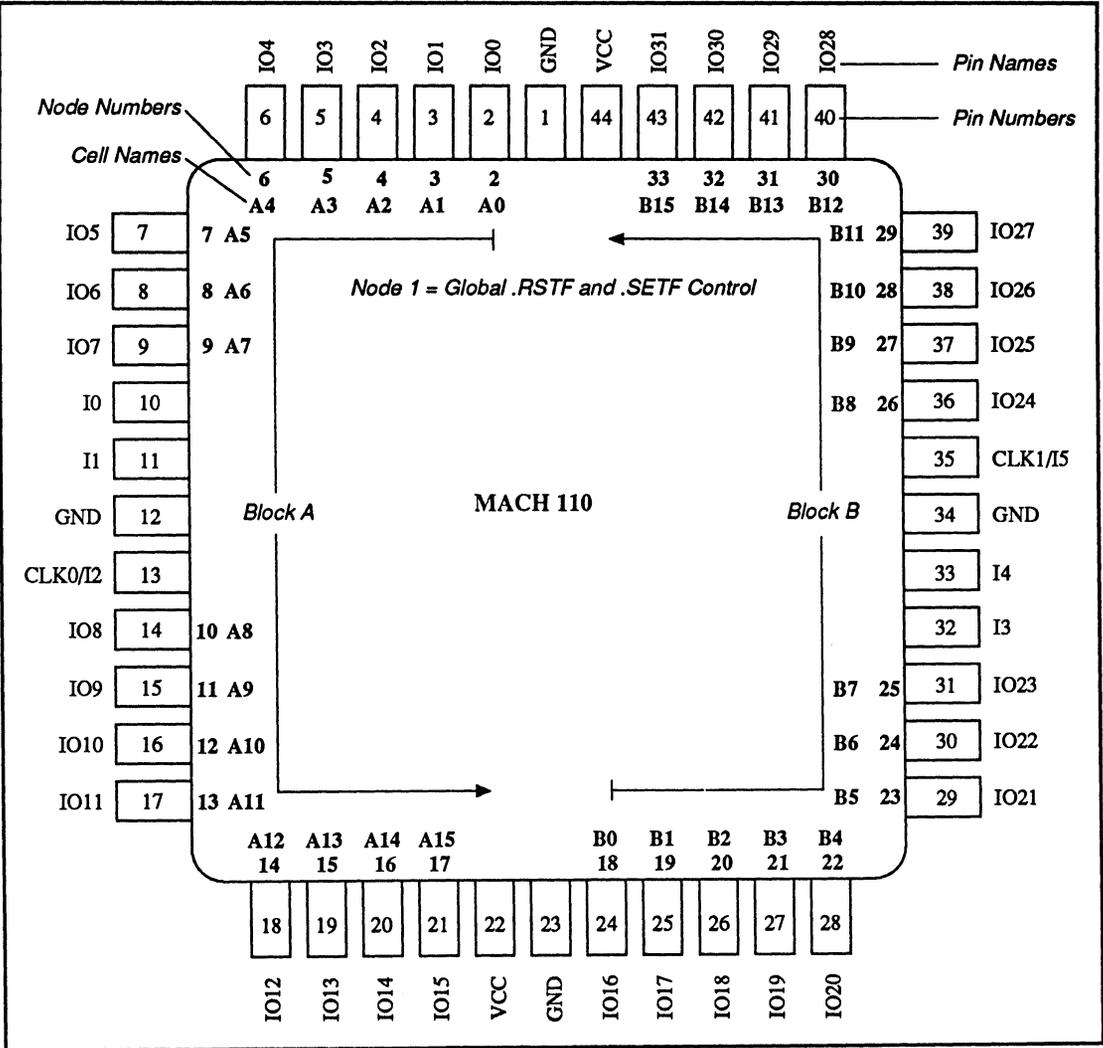
A3 = I4 ;latched input

MACH 110 DEVICE

An illustration of the MACH 110 node numbers and cell names is shown next. Each I/O pin in the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell names have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

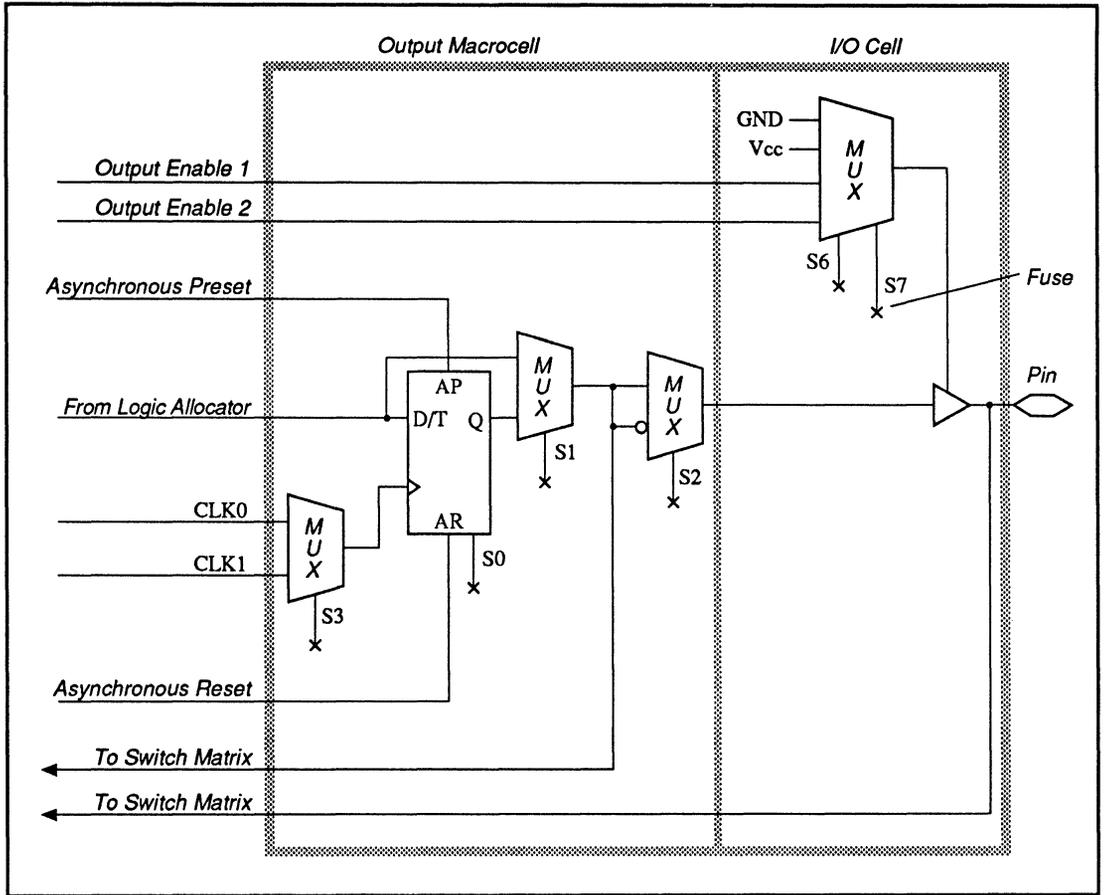
MACH 110 DEVICE



MACH 110 Node Numbers and Cell Names

MACH 110 DEVICE

The logic for a MACH 110 output macrocell and I/O cell is shown next.



MACH 110 Output Macrocell and I/O Cell

MACH 110 DEVICE

The output of the macrocell can be combinatorial, D flip-flop, or T flip-flop. You define the configuration in the pin declaration or equation segment of the design.

Note: For MACH 1 series devices, latches are implemented as combinatorial functions. MACH 2 series devices have resources that allow latches with active-low latch inputs to be configured directly.

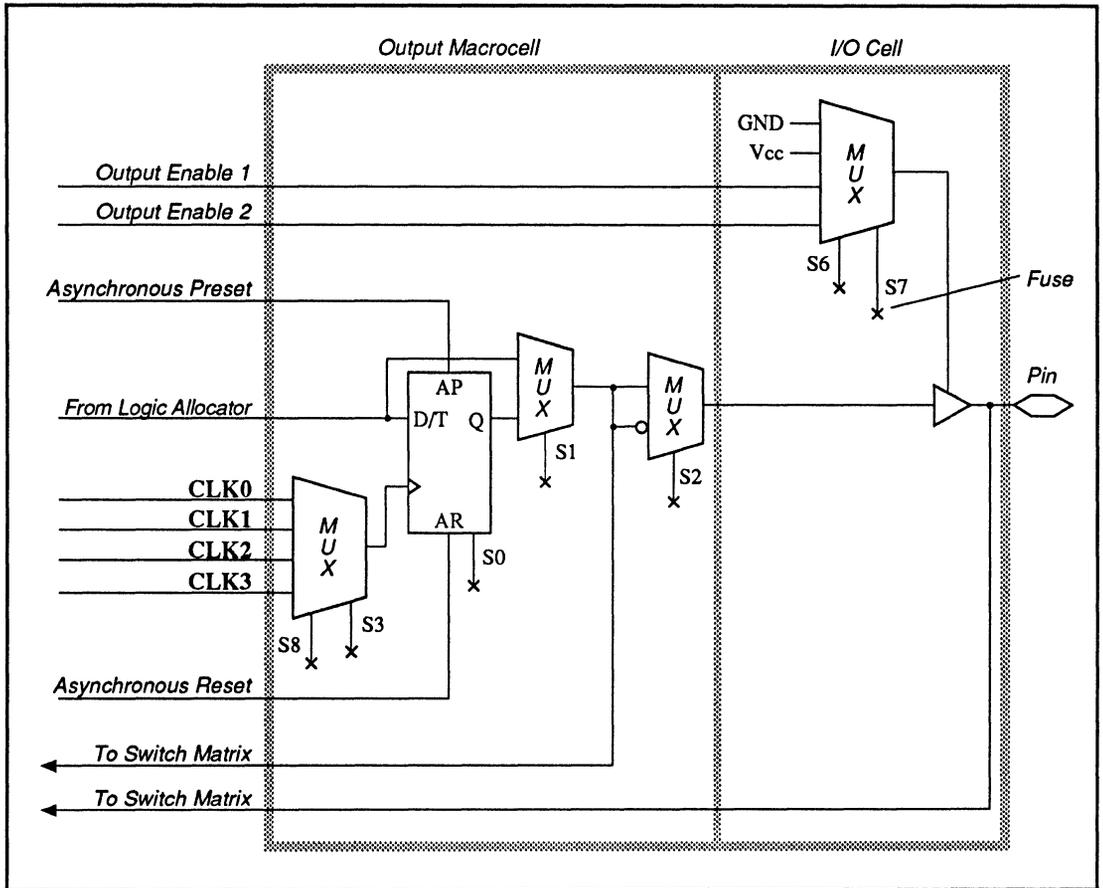
MACH 120 DEVICE

An illustration of the MACH 120 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell names have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

MACH 120 DEVICE

The logic for a MACH 120 output macrocell and I/O cell is shown next.



MACH 120 Output Macrocell and I/O Cell

MACH 120 DEVICE

The output of the macrocell can be combinatorial, D flip-flop or T flip-flop. You define the configuration in the pin declaration or equation segment of the design.

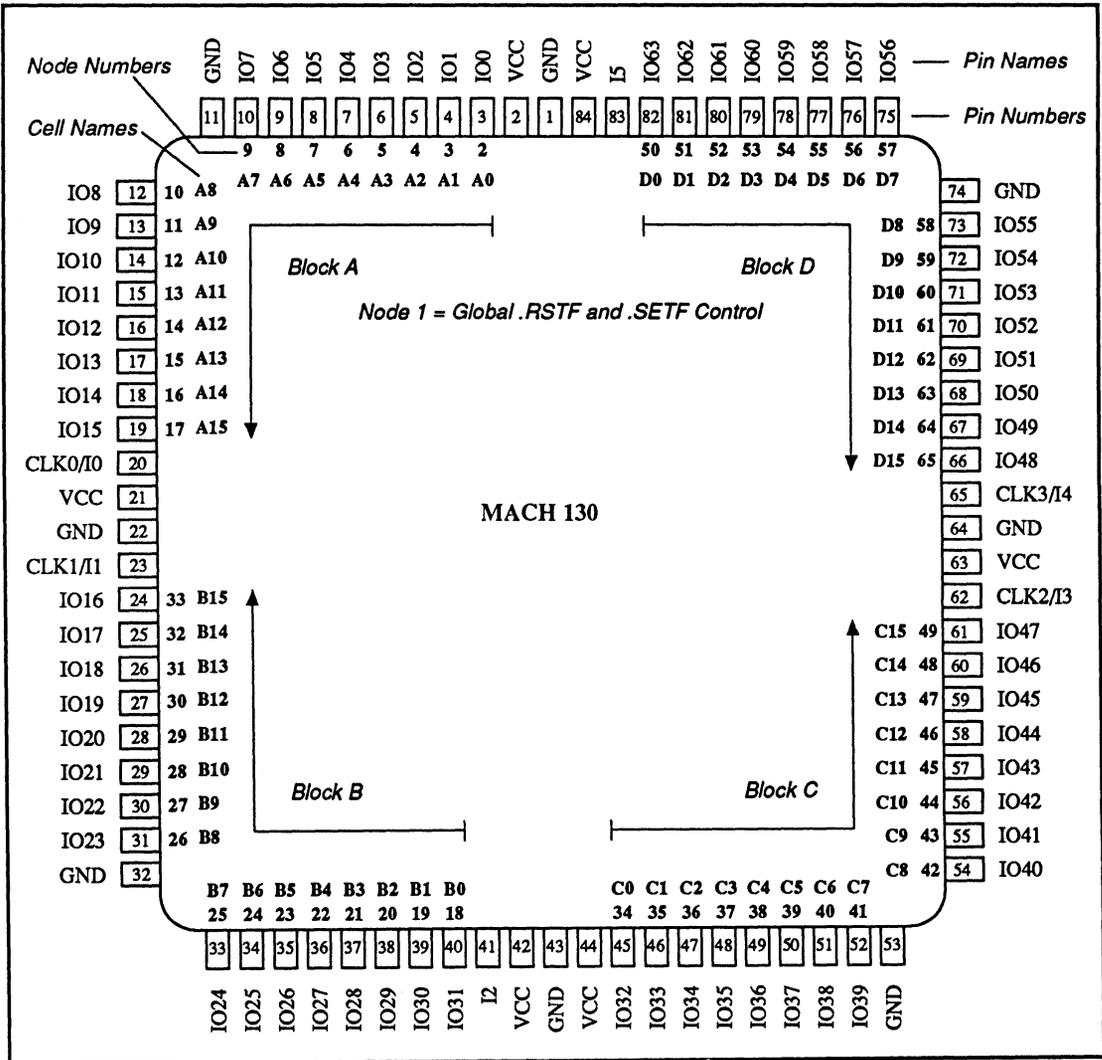
Note: For MACH 1 series devices, latches are implemented as combinatorial functions. MACH 2 series devices have resources that allow latches with active-low latch inputs to be configured directly.

MACH 130 DEVICE

An illustration of the MACH 130 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 3 corresponds to node 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell names have been assigned for reference purposes, and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

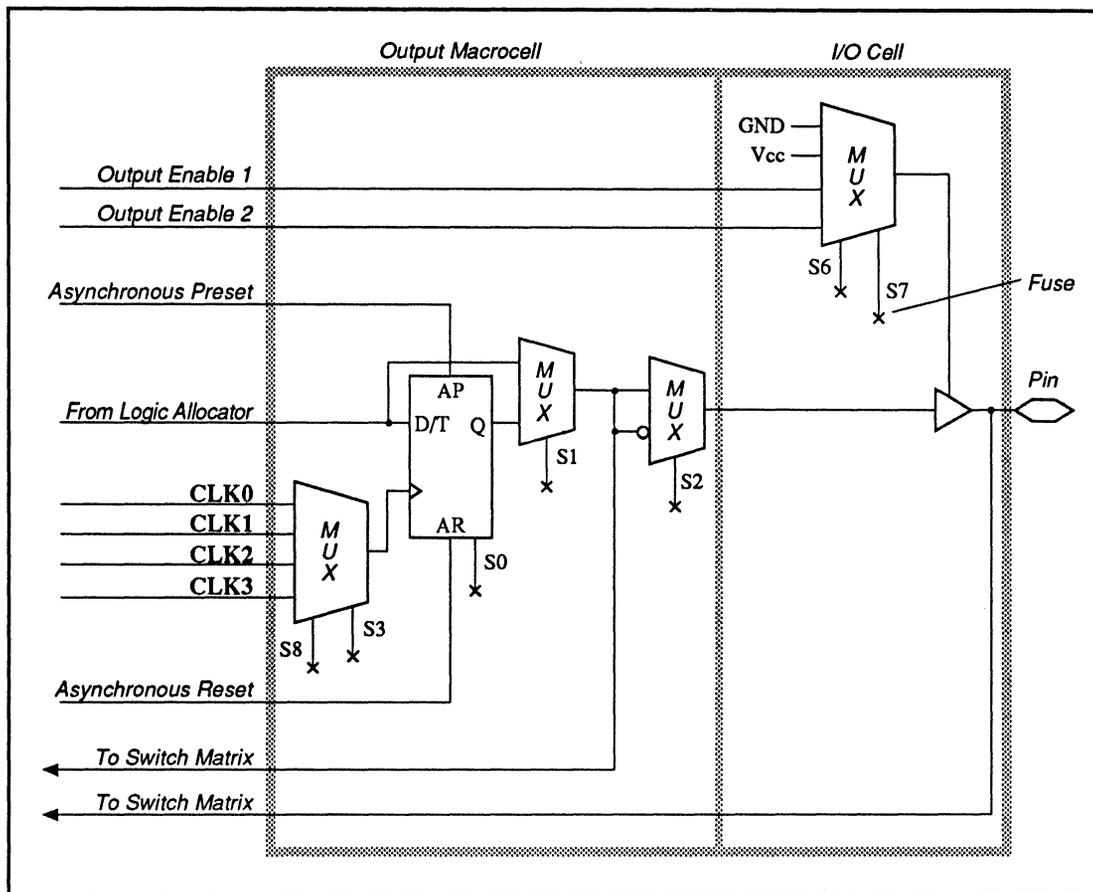
MACH 130 DEVICE



MACH 130 Node Numbers and Cell Names

MACH 130 DEVICE

The logic for a MACH 130 output macrocell and I/O cell is shown next.



MACH 130 Output Macrocell and I/O Cell

MACH 130 DEVICE

The output of the macrocell can be combinatorial, D flip-flop, or T flip-flop. You define the configuration in the pin declaration or equation segment of the design.

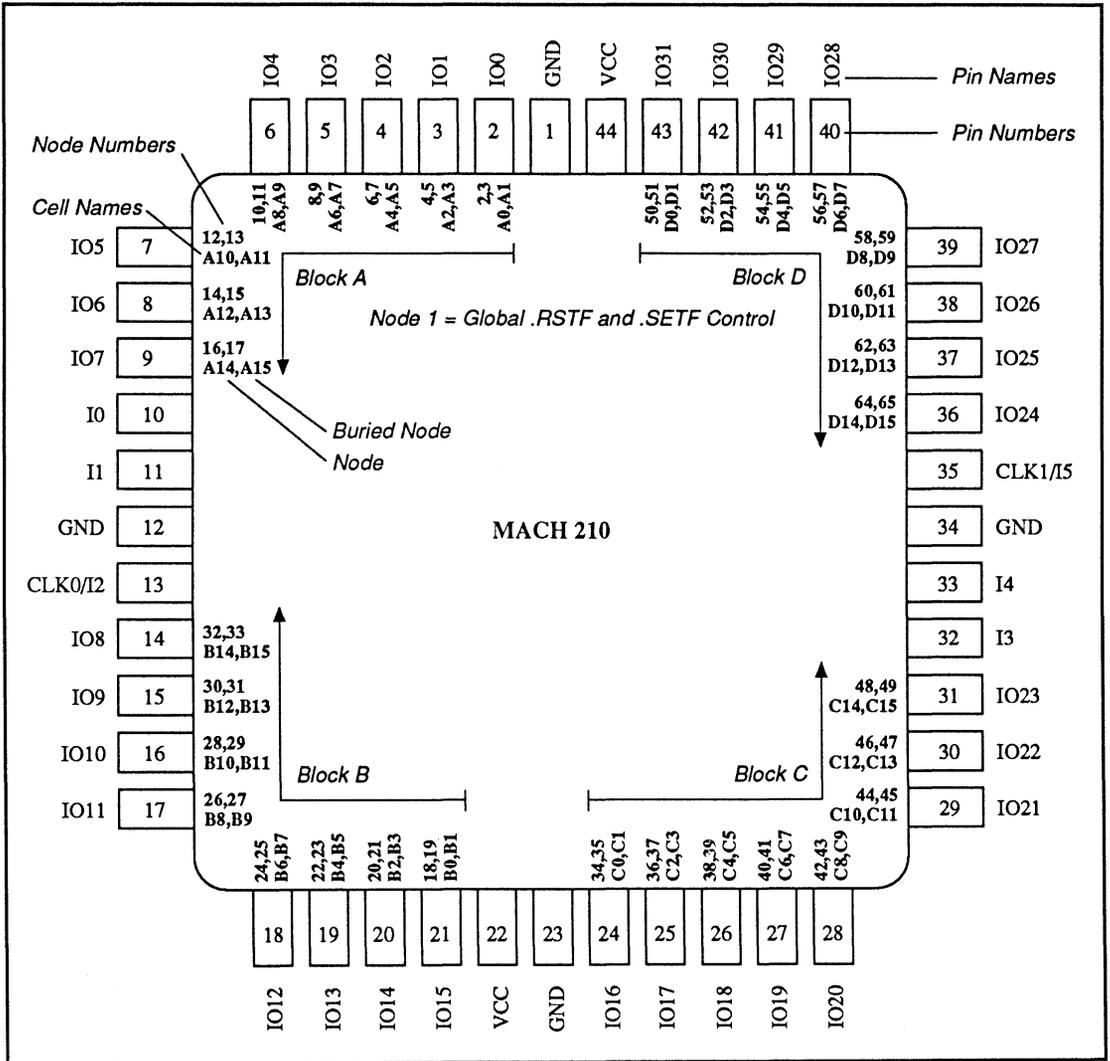
Note: For MACH 1 series devices, latches are implemented as combinatorial functions. MACH 2 series devices have resources that allow latches with active-low latch inputs to be configured directly.

MACH 210 DEVICE

An illustration of the MACH 210 node numbers and cell names is shown next. Each I/O pin in the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell names have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

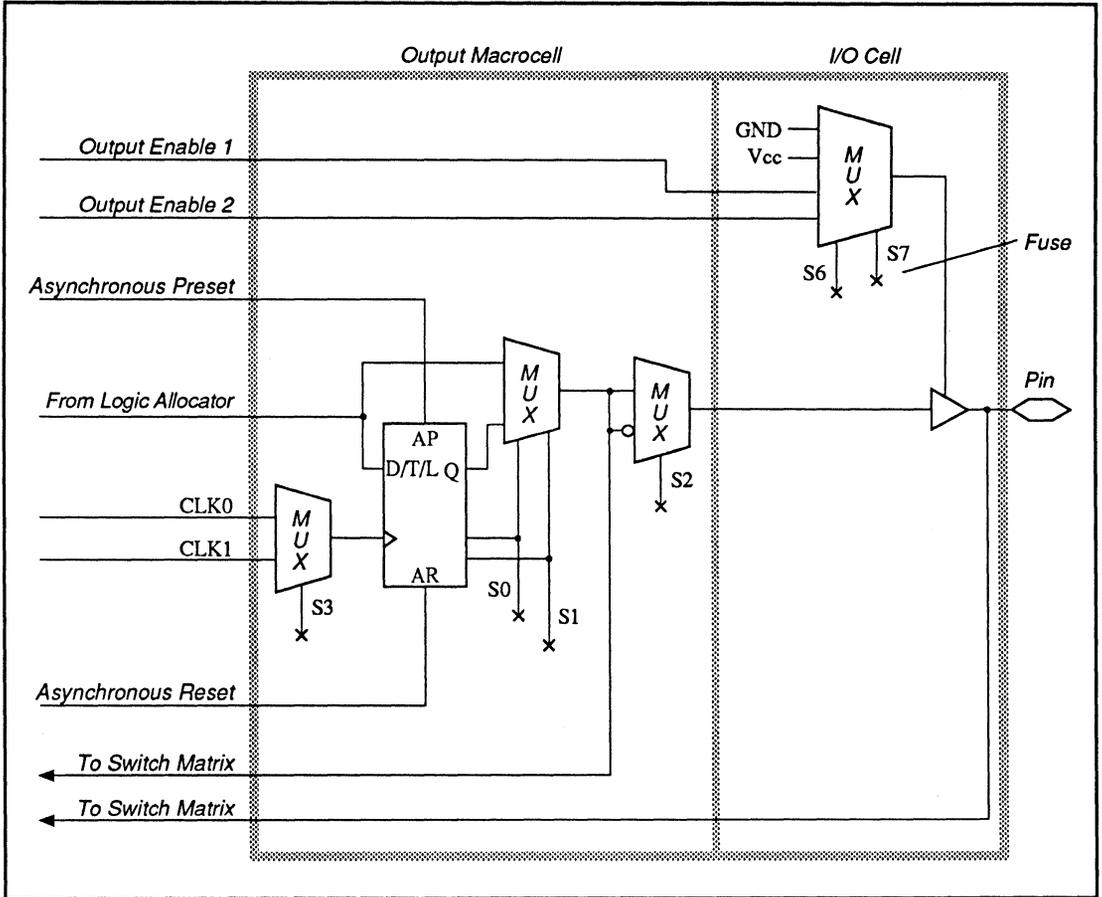
MACH 210 DEVICE



MACH 210 Node Numbers and Cell Names

MACH 210 DEVICE

The logic for a MACH 210 output macrocell and I/O cell is shown next.

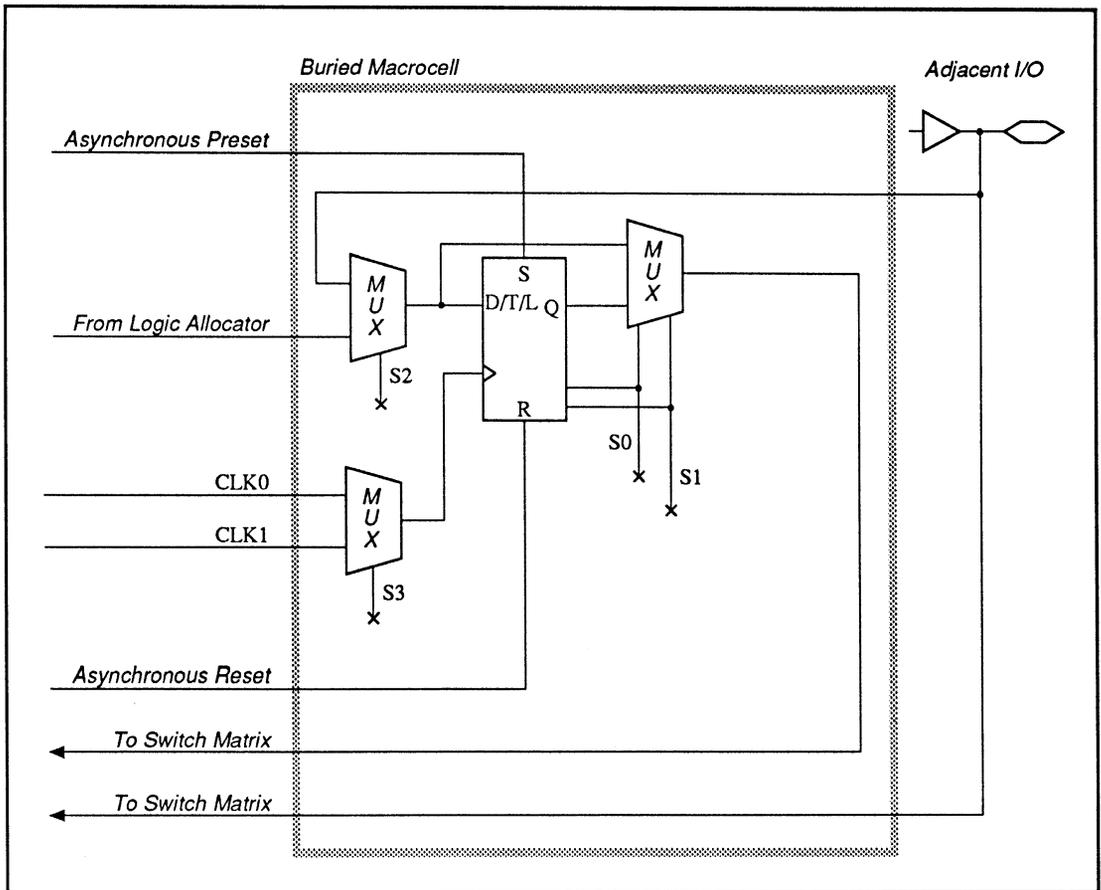


MACH 210 Output Macrocell and I/O Cell

MACH 210 DEVICE

The output of the macrocell can be combinational, D flip-flop, T flip-flop, or latch.²⁹ You define the configuration in the pin declaration or equation segment of the design.

The next figure illustrates the MACH 210 buried macrocell layout.



MACH 210 Buried Macrocell

²⁹ Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

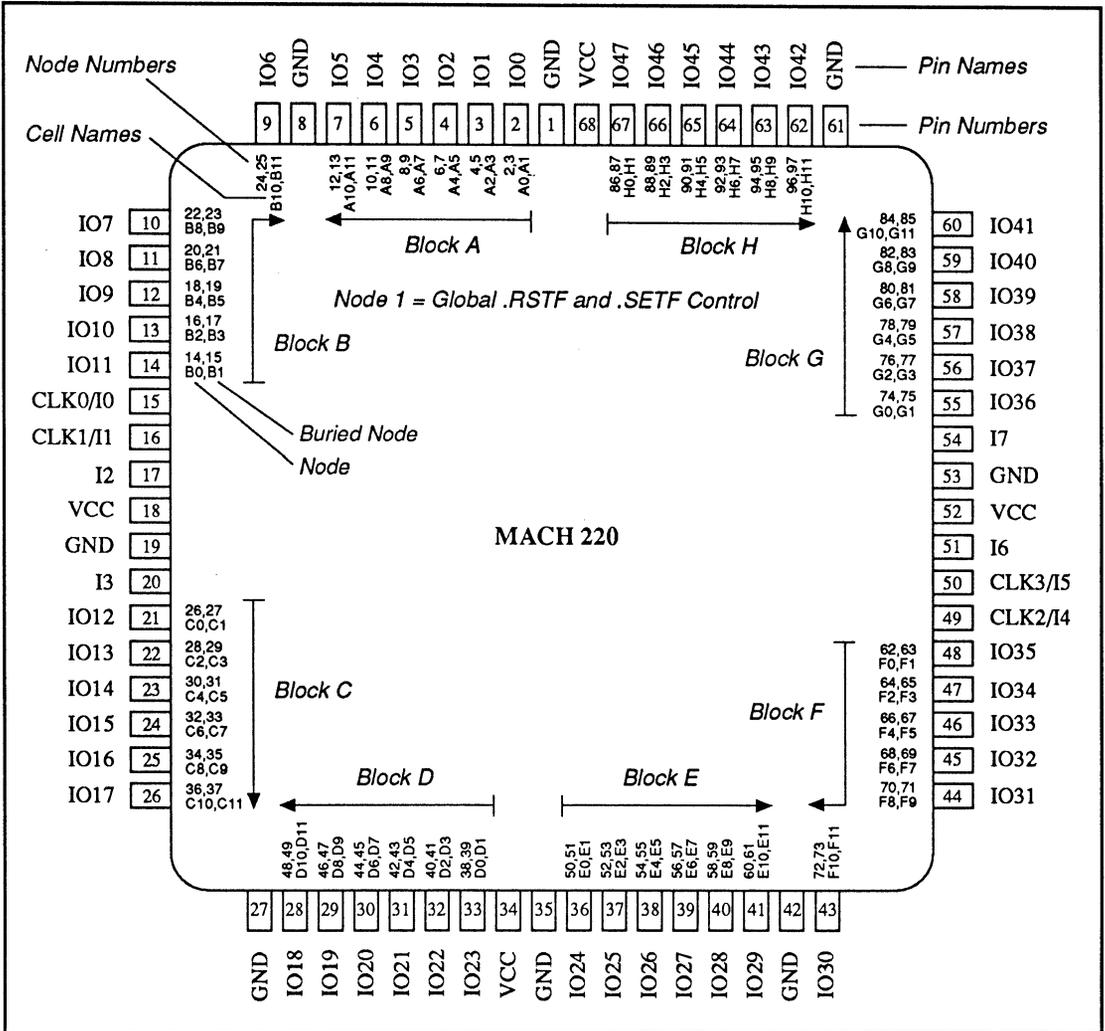
MACH 220 DEVICE

An illustration of the MACH 220 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 2 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 2. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: The illustrated node numbers for the MACH 220 device are PRELIMINARY, and may change without notice.

Also: Pin and cell **names** have been assigned for reference purposes and reflect functionality when appropriate. You can assign your own names in the pin declaration segment of the design file.

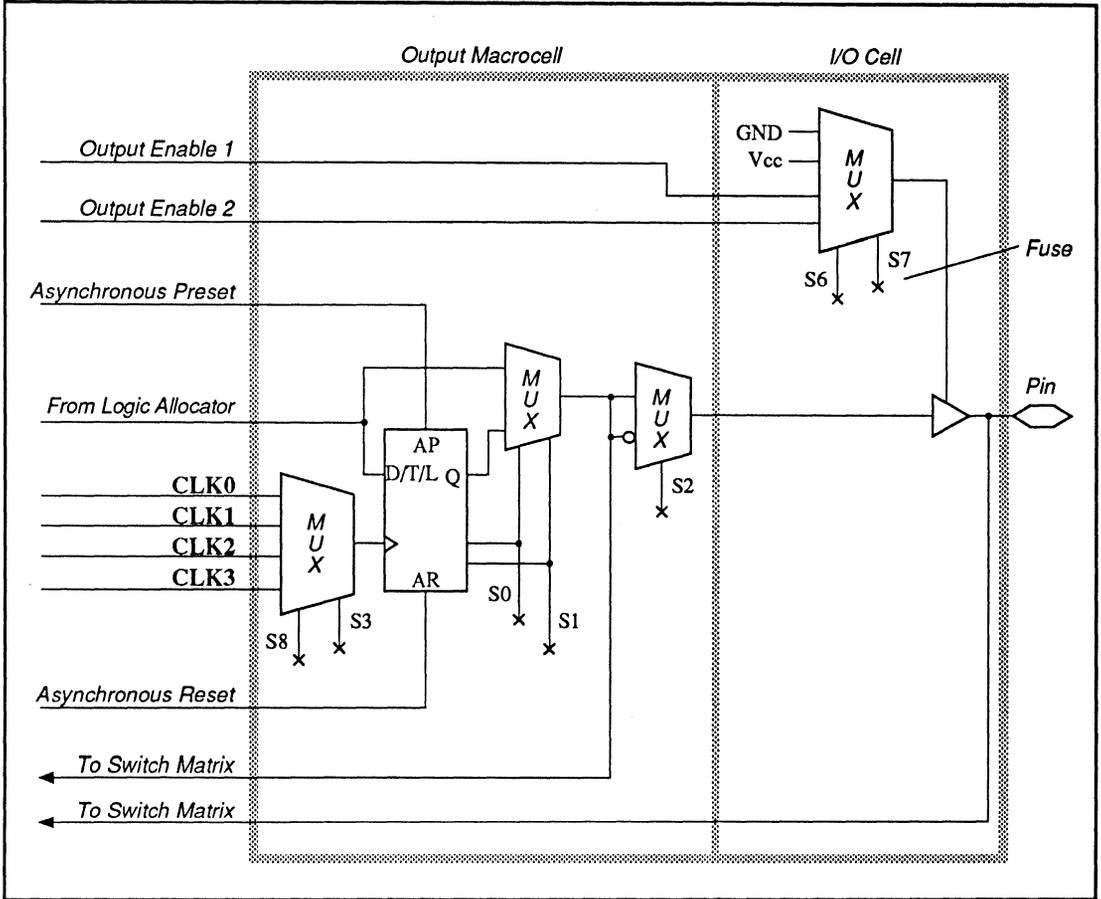
MACH 220 DEVICE



MACH 220 Node Numbers and Cell Names (Preliminary)

MACH 220 DEVICE

The logic for a MACH 220 output macrocell and I/O cell is shown next.

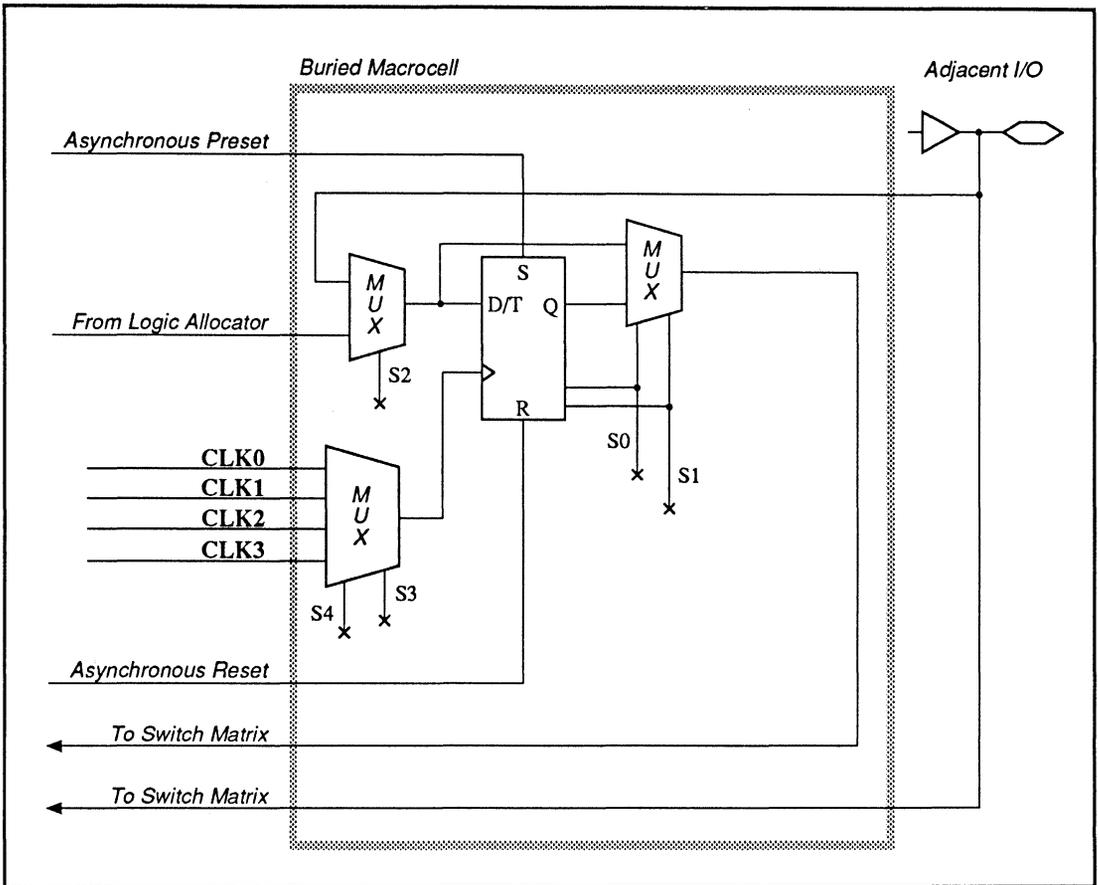


MACH 220 Output Macrocell and I/O Cell

MACH 220 DEVICE

The output of the macrocell can be combinational, D flip-flop, T flip-flop, or latch.³⁰ You define the configuration in the pin declaration or equation segment of the design.

The next figure illustrates the MACH 220 buried macrocell layout.



MACH 220 Buried Macrocell

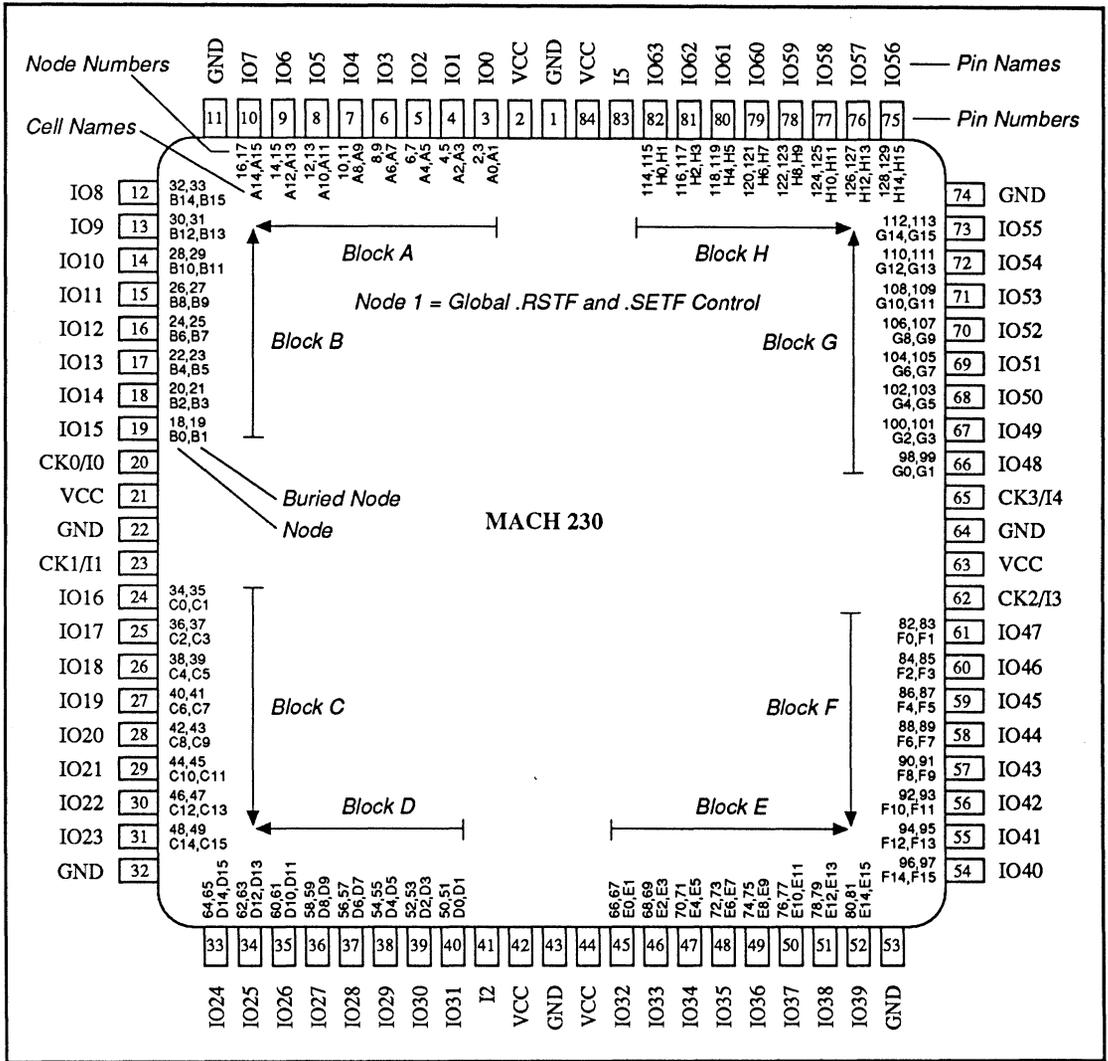
³⁰ Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

MACH 230 DEVICE

An illustration of the MACH 230 node numbers and cell names is shown next. Each I/O pin of the device has an associated node, designated by a number. For example, pin 3 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 3. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.

Important: Pin and cell names have been assigned for reference purposes and, where appropriate, reflect functionality. You can assign your own names in the pin declaration segment of the design file.

MACH 230 DEVICE

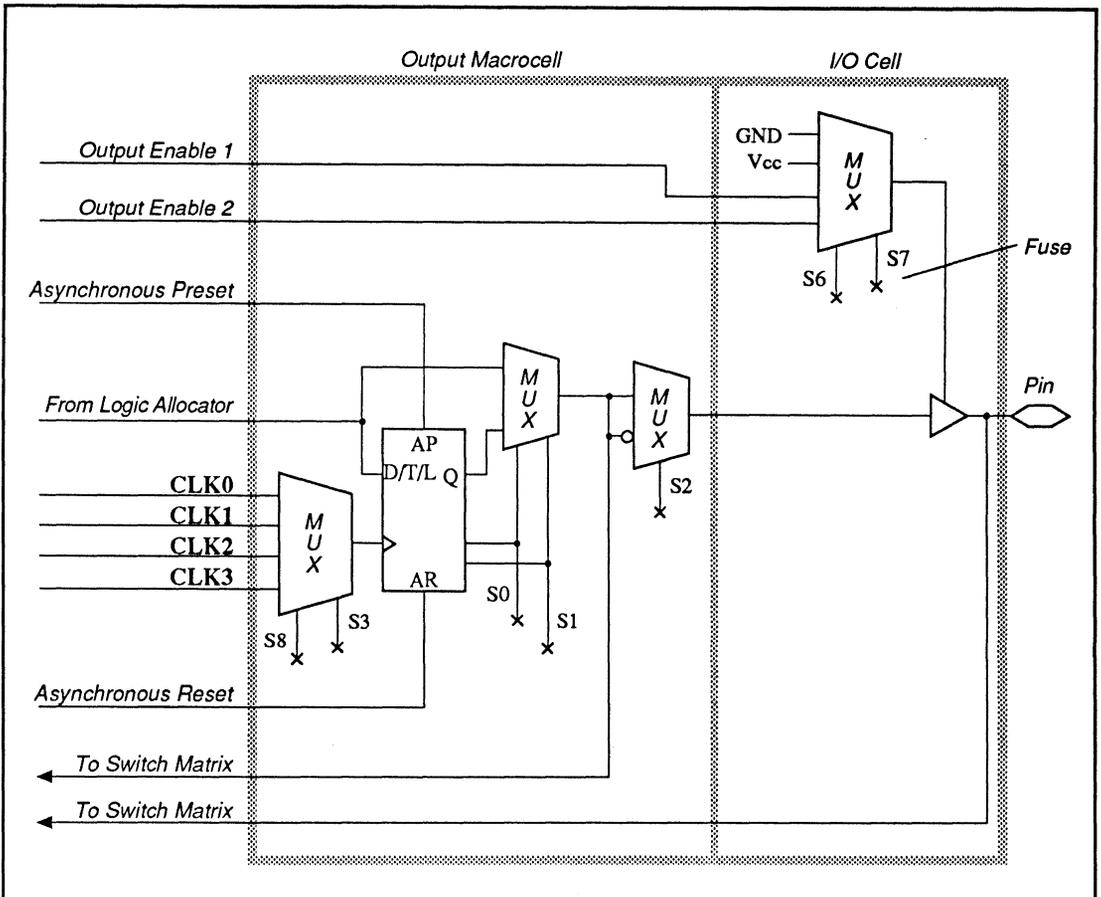


MACH 230 Node Numbers and Cell Names

MACH 230 DEVICE

The logic for a MACH 230 output macrocell and I/O cell is shown next.

The output of the macrocell can be combinatorial, D flip-flop, T flip-flop, or latch.³¹ You define the configuration in the pin declaration or equation segment of the design.



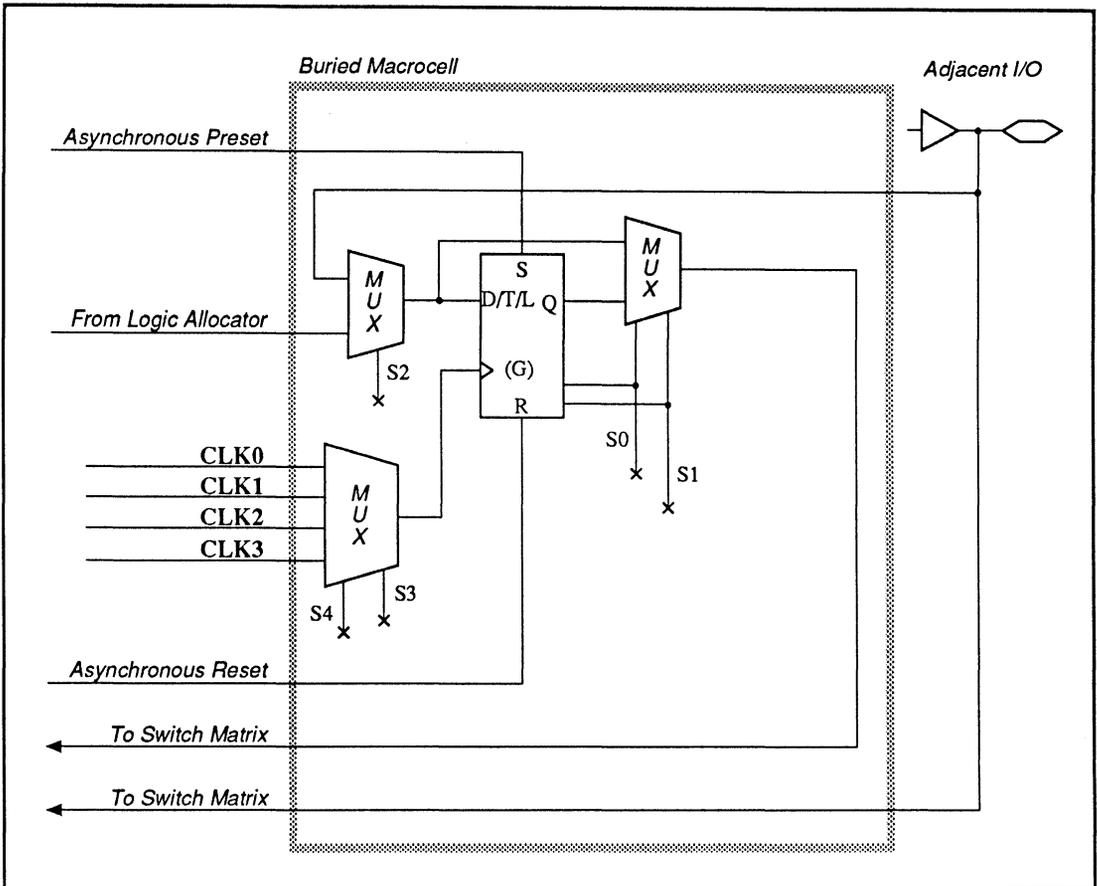
MACH 230 Output Macrocell and I/O Cell

³¹ Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

MACH 230 DEVICE

The output of the macrocell can be combinatorial, D flip-flop, T flip-flop, or latch.³² You define the configuration in the pin declaration or equation segment of the design.

The next figure illustrates the MACH 230 buried macrocell layout.



MACH 230 Buried Macrocell

³² Refer to Section II, Chapter 6, for a function table of illegal latch states. The macrocell latch has an active-low latch input.

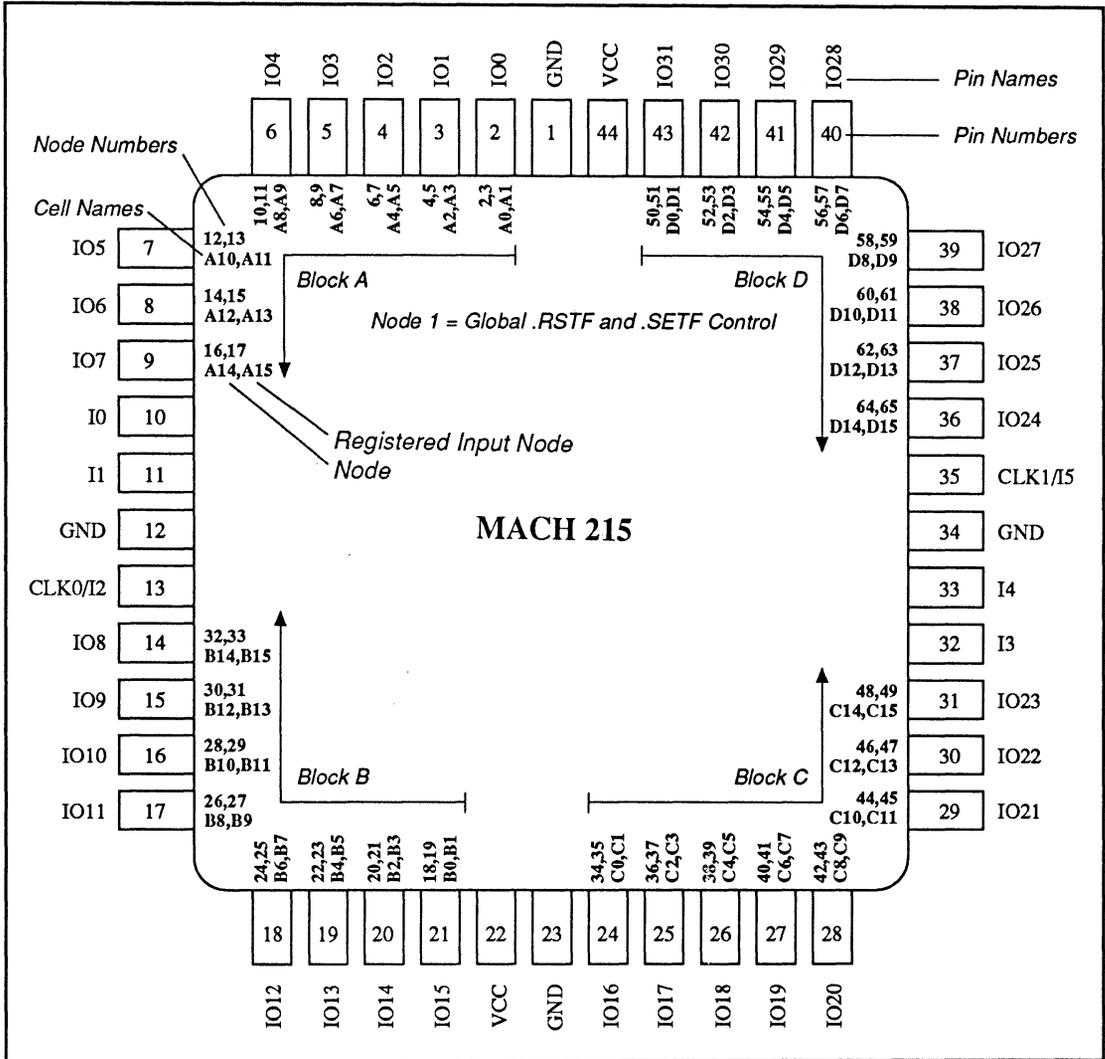
MACH 215 DEVICE

An illustration of the MACH 215 node numbers and cell names is shown next. *Each I/O pin of the device has an associated node, designated by a number. For example, pin 3 corresponds to node 2. For the MACH 2 series devices, there is also a buried node associated with each I/O pin. The signals at these buried nodes do not go to the pin. For example, node 3 is associated with pin 3. You use these numbers to fix pin and node locations in the pin declaration segment of the design file.*

The MACH 215 device has no buried feedback; instead it has dedicated input registers.

Important: Pin and cell **names** have been assigned for reference purposes and, where appropriate, reflect functionality. You can assign your own names in the pin declaration segment of the design file.

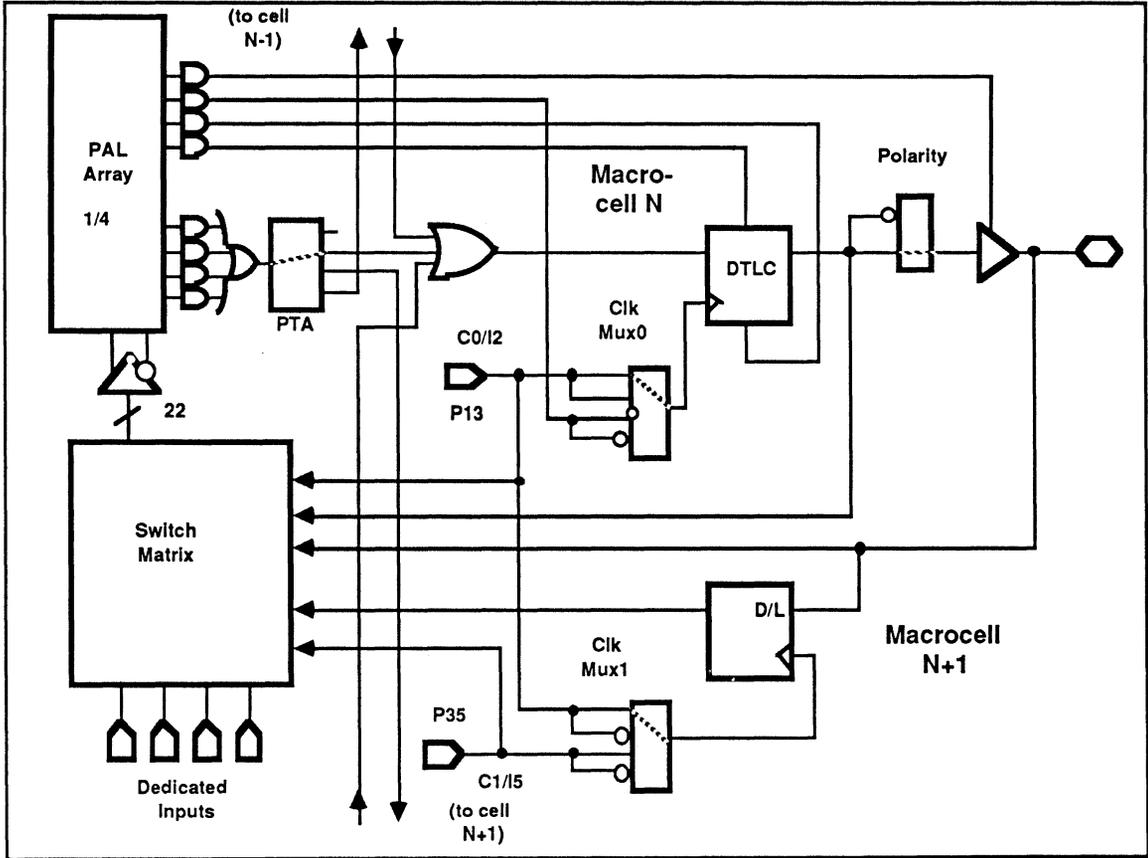
MACH 215 DEVICE



MACH 215 Node Numbers and Cell Names

MACH 215 DEVICE

The logic for a MACH 215 macrocell architecture is shown next.



MACH 215 Macrocell Architecture

SECTION V

APPENDICES

Appendix A: PLD Text Editor

APPENDIX A

PLD TEXT EDITOR

CONTENTS

PLD TEXT EDITOR	1
A.1 FILE MENU	2
A.2 WINDOW MENU	3
A.3 BLOCK MENU	4
A.4 SEARCH MENU	5
A.5 PRINT MENU	6
A.6 MACRO MENU	7
A.7 EDITING MENU	8
A.8 OTHER MENU	9
A.9 QUIT MENU	11

The software includes a text editor to create and edit PALASM design specification (PDS) files. The information presented here includes each command that's available, listed by menu.

- File menu, A.1
- Window menu, A.2
- Block menu, A.3
- Search menu, A.4
- Print menu, A.5
- Macro menu, A.6
- Editing menu, A.7
- Other menu, A.8
- Quit menu, A.9

A.1 FILE MENU

With the text editor, you can switch easily between multiple files, which are inserted into a **ring** in memory as you edit them. When you quit a file, it is deleted from the ring and the previous file in the ring becomes the new current file. Commands to switch between multiple files, and other file commands, are listed in the table below.

FILE MENU COMMAND	COMMAND DEFINITION
Load	Load the named file(s) into the ring.
File	Quit and save the current file.
Save	Write the current file to disk.
Quit file	Quit the current file without saving changes.
Next	Make the next file in the ring the new current file.
Prev	Make the previous file in the ring the new current file.
Read	Insert the specified file into the current file.
Change name	Change the name of the current file.
Write block	Write the current marked block to a specified file.
OS shell	Return to the operating system environment.
Global file	Quit and save all files which have been loaded.

A.2 WINDOW MENU

You can display up to eight windows on the screen at one time. Each window can contain a separate file, and the same file can be viewed in multiple windows. Window commands are listed in the table below.

WINDOW MENU COMMAND	COMMAND DEFINITION
Close	Close the current window, unless it is the only window on the screen, in which case do nothing.
Grow	Increase the size of the current window if there are multiple windows on the screen.
Split	Split the current window horizontally to create a new window.
Next	Make the next window the new current window.
One	Close all windows except the current window and expand it to occupy the entire screen.
Prev	Make the previous window the new current window.
Shrink	Reduce the size of the current window by expanding the window above or below it.
Zoom	Toggle the current window between occupying the entire screen and sharing the screen with other windows.

A.3 BLOCK MENU

Block commands allow you to identify a contiguous portion of text that you can copy, or cut and move, to a new area in the same file or to a different file. They are listed in the table below.

BLOCK MENU COMMAND	COMMAND DEFINITION
mark block Begin	Mark the beginning of a block of characters. The block is not shown until you also mark the end.
mark block End	Mark the end of a block of characters. The block is not shown until you also mark the beginning.
Copy block	Copy the current marked block to the new cursor position.
Move block	Copy the current marked block, delete it from its present location, and insert it in the new cursor position.
Delete block	Delete the current marked block.
Unmark	Remove the mark from the currently blocked characters.
mark Line	Mark the complete line where the cursor is located. Moving the cursor to different lines adds those lines to the block.

A.4 SEARCH MENU

You use the search commands to look for a specified string of characters in the current file. After choosing one of the commands, you are prompted for the string and given several search options. The search is then performed; you can cancel the command by pressing [Return].

The search commands are listed in the table below.

SEARCH MENU COMMAND	COMMAND DEFINITION
Find	Search for a specified string of characters within the current file.
Replace	Search for and replace a specified string of characters with a different string.
Again	Repeat the previous Find or Replace command.

A.5 PRINT MENU

You can specify how you want the current file printed. The table below lists the available print commands.

PRINT MENU COMMAND	COMMAND DEFINITION
print All	Send the entire current file to the printer.
print Block	Send the current marked block to the printer. The marked block must be in the current file.
send Formfeed	Send a form-feed character (ASCII 12) to the printer.
set Left margin	Set the number of spaces to be printed at the beginning of each line. The default is 0 spaces.
set Page size	Set the number of lines to be printed per page. A value of 0 will allow continuous printing.

A.6 MACRO MENU

You can capture keystrokes to create a macro using the Macro record command. For each macro you create, you are asked to assign a unique command key. Every time you press that key, the recorded keystrokes are repeated.

You can establish different libraries of macros by saving them under a file name and loading them into the text editor when you want to use them. The macros are then assigned to the keys to which they were bound when they were saved.

The table below lists the available macro commands.

MACRO MENU COMMAND	COMMAND DEFINITION
Macro record	Toggle Macro record between ON and OFF. You assign a command key, and all following keystrokes are recorded until you execute Macro record a second time.
Read macro	Load the named macro file, with all the macros it contains, from disk into the text editor's internal macro buffer. You are prompted for the name of the file.
Write macro	Save all current defined macros to a disk file. You are prompted to name the file.

A.7 EDITING MENU

The text editor allows you to use the editing commands listed in the table below.

EDITING MENU COMMAND	COMMAND DEFINITION
Add line	Add a blank line below the current line, and place the cursor on the new line. The cursor column does not change.
Delete line	Delete the current line and place the cursor on the following line.
delete to End of line	Delete text on the current line from the cursor position to the end of the line.
Insert line	Insert a blank line above the current line and place the cursor on that line.
Join line	Join the end of the current line to the line below it.
Split line	Split the current line at the cursor position.
Undelete line	Recover the last line deleted from the current file.

A.8 OTHER MENU

Four different operating modes and two tab-setting options are available with the text editor. Those commands, and several other function commands, are listed in the table below.

OTHER MENU COMMAND	COMMAND DEFINITION
set Autoindent OFF (ON)	<p>Toggle between Autoindent ON and OFF. The default is ON.</p> <ul style="list-style-type: none">• In auto-indent mode, the left margin is maintained through word-wrap, paragraph-reformatting, and the [Return] key, and is aligned with the first non-space character.
set Insert OFF (ON)	<p>Toggle between Insert ON and OFF. The default is ON.</p> <ul style="list-style-type: none">• In insert mode, text is shifted to the right as you enter new text. Backspacing shifts text to the left as it deletes the character to the left of the cursor.• In overwrite (non-insert) mode, text you enter overlays text at the cursor position. Backspacing deletes the character to the left of the cursor but does not shift the remainder of the line.
set Wordwrap OFF (ON)	<p>Toggle between Wordwrap ON and OFF. The default is ON.</p> <ul style="list-style-type: none">• In word-wrap mode, text you enter wraps around to the next line when the cursor goes past the right margin and a non-space character is typed. The left margin is determined by the function of the auto-indent mode.

continued on next page...

OTHER MENU COMMAND	COMMAND DEFINITION
set Ptabwidth	Set the physical tab width to a value of 2, 4, or 8. This determines the width of physical tab characters found in files. You are prompted to set the value.
set Ctabwidth	Set the cursor tab width to any value from 2 through 12. This defines the actual screen position to which the cursor will move each time a tab key is pressed. You are prompted to set the value.
set Right margin	Set the right margin for use in the word-wrap mode.
set Backups OFF (ON)	<p>Toggle between Backups ON and OFF. The default is ON.</p> <ul style="list-style-type: none"> • The backup function keeps backup copies of files that are written to disk.
set Enter matching ON (OFF)	<p>Toggle between Enter matching ON and OFF. The default is OFF.</p> <ul style="list-style-type: none"> • The match function automatically enters the matching right character when the left double quote, paren, or square bracket is typed.

A.9 QUIT MENU

Leaving the text editor, and saving all the work you've done, is easily accomplished by using the commands listed in the table below.

QUIT MENU COMMAND	COMMAND DEFINITION
Quit all files	Quit all files without saving changes.
Exit all files	Conditionally quit all files. If the file has been modified, the editor prompts for whether the changes should be saved.

SECTION VI

GLOSSARY / INDEX

Glossary

Index

GLOSSARY

- Active Edge** A low-to-high or high-to-low signal transition that initiates an action.
- Assemble** A transparent software process that generates a JEDEC formatted fuse map and test vectors.
- Back Annotation** A user-selectable software process that takes pin assignment names determined by the fitting process and writes them back into the PDS file.
- Bank** A collection of I/O or buried macrocells within a block. In the MACH 110 device, each block has two banks with 8 I/O cells in each bank. In the MACH 210 device, each block has one bank with 8 I/O cells and one bank with 8 buried macrocells. See Block.
- Block** A collection of PAL-like structures that function as independent PAL devices on a single chip. Each block contains a product-term array, a logic allocator, macrocells, and I/O cells. The blocks communicate with each other only through the switch matrix.
- Buried Macrocell** A buried macrocell allows the designer to use registered inputs. The input register is a D-type flip-flop. Once configured as a registered input, the buried macrocell can not generate logic from the product term array. A buried macrocell does not send its output to an I/O cell. The output of a buried macrocell is a feedback signal to the switch matrix. This allows the designer to generate additional logic without requiring additional pins.
- Circuit Simulation** A software breadboard to verify design functionality and performance. Software that logically emulates a circuit's functions to ensure proper design.

Combinatorial Macro	A macro that performs a logical function and has no storage capability.
Combined files	A PDS file created by merging two separate PDS files. Certain variables may be automatically renamed or reassigned to different pins.
Controllability	The degree to which signals in a part of a circuit can be made to take on specific values through manipulation of primary inputs; used in testability analysis.
Critical Path Evaluation	The identification and analysis of signal paths whose delays could limit the speed of the circuit.
Current Design File	The design file that you specified to work on.
Default Value	The value used unless you specify a different one.
Design File	A file containing the PALASM description or schematic representation of a design.
Disassemble	The process of translating a .TRE file or a JEDEC fuse map back into a PDS.
EEPROM (E² PROM)	See Electrically Erasable and Programmable ROM.
Electrically Erasable and Programmable ROM	Similar to an EPROM, but it stores the charge on a floating gate. Newer EEPROMs can erase individual data bytes.
EPLD	Erasable Programmable Logic Device, such as an EPROM.
EPROM	See Erasable Programmable ROM.
Erasable Programmable ROM	Usually refers to the UV erasable, 2764 device type. Generally, EPROMs are erased by shining ultraviolet light on the chip through a quartz window on the package.

Expand	A transparent software routine that performs one of two functions depending on your design. Converts Boolean equations to sum-of-product-terms form, and state-machine constructs to Boolean Exclusive-OR.
Field	An area in a form where you enter information by typing or selecting from a list of options. Also, an area in a PALASM language construct where you enter specific information.
Field Programmable Logic	Standard products that the user can configure to a specific application, such as PAL and FPLA devices.
Fit	A software process that places pins and nodes after compilation. The placement is automatic if pins and nodes are left floating. This process generates the JEDEC fuse map.
FITR	The FITR software automatically manages the internal arrangement of resources. The software automatically distributes product terms to the macrocells and adjusts the distribution as required by the design.
I/O cell	The I/O cell provides a three-state output buffer. The three-state buffer can be left permanently enabled for use only as an output; permanently disabled for use as an input; or it can be controlled by one of two product terms, for bidirectional signals and bus connections. The two product terms provided are common to a bank of eight I/O cells.
Float	A feature that allows pin and node placement to occur automatically during the fitting process. To leave a pin or node floating, you use a question mark, ?, in the location field of a pin or node statement.
Form	Forms provide specifications to the software. Each form provides one or more text, status, or option fields.

Functional Primitives

Design building blocks, such as adders, shifters, decoders, and memory. A functional primitive differs from a gate-level primitive only in the individual element's degree of functional complexity.

Gate Splitting

An automatic software process that routes feedback through the switch matrix, AND array, and logic allocator to the assigned pin. You can use product terms from nonadjacent macrocells in a MACH 110 or 210 device, including other blocks. Gate splitting uses product terms in multiples of four. However, unlike product-term steering, gate splitting adds one propagation delay for each pass through the AND logic array.

Initial Value

The preset value for an option when the software is invoked. After the software starts executing, the option's value can be changed. See Default Value.

Initialize

The process of establishing an initial condition or starting state. For example, setting logic elements in a digital circuit, or the contents of a storage location, to known states so that subsequent application of digital test patterns drive the logic elements to another known state. Initialization sets counters, switches, and addresses to zero or other starting values at the beginning of, or at prescribed points in, a computer routine.

Keyword

A word used in a language to indicate a specific construct or structure.

Library

See Macro Library.

Log File

A file containing all processes and messages generated during a software processing session.

Logic Allocator

The logic allocator is a block within which different product terms are allocated to the appropriate macrocells.

MACH Device

The MACH is a new 15 ns CMOS EE PLD available in 44-, 68-, and 84-pin packages. The MACH device provides programmable logic capabilities from approximately 900 to 3600 gates. The MACH 1 and 2 families consist of PAL blocks interconnected by a programmable switch matrix. Each family member is differentiated by the number of pins, the number of macrocells, and the amount of interconnect. The MACH 1 family has only output macrocells. The MACH 2 family has output and buried macrocells. Otherwise the families are the same.

MACH 110

The MACH 110 consists of two PAL blocks interconnected by a programmable switch matrix. The MACH 110 has 32 macrocells, 44 pins and 38 inputs.

MACH 120

The MACH 120 consists of four PAL blocks interconnected by a programmable switch matrix. It has 48 macrocells, 68 pins and 58 inputs.

MACH 130

The MACH 130 consists of four PAL blocks interconnected by a programmable switch matrix. It has 64 macrocells, 84 pins and 70 inputs.

MACH 210

The MACH 210 consists of four PAL blocks interconnected by a programmable switch matrix. It has 64 macrocells, 44 pins and 38 inputs. The MACH 210 also has dedicated buried macrocells.

MACH 220

The MACH 220 consists of six PAL blocks interconnected by a programmable switch matrix. It has 96 macrocells, 68 pins and 48 inputs. The MACH 220 also has dedicated buried macrocells.

MACH 230

The MACH 230 consists of eight PAL blocks interconnected by a programmable switch matrix. It has 128 macrocells, 84 pins and 70 inputs. The MACH 230 also has dedicated buried macrocells.

Macro

The elements in a library that you can retrieve to create a schematic.

Macrocells

There are two fundamental types of macrocells: an output macrocell and a buried macrocell. A buried macrocell is found only in the MACH 2 family of devices. The use of buried macrocells effectively doubles the number of macrocells available without increasing the pin count. Both macrocell types can generate registered or combinatorial output. If used, the register can be as a T- or D-type flip-flop. Programmable polarity (for output macrocells) and the T-type flip-flop give the software a way to minimize the number of product terms needed. All macrocells have internal feedback, allowing a pin to be used as an input if the macrocell signal is not needed externally. See also Output Macrocell.

Macro Library

A collection of macros, organized or classified by function, that contain the macros required for schematic capture.

Merge

Combine two or more text files into one description or PDS file. A synonym for Combine.

Minimize

A transparent software routine that reduces a set of Boolean equations to a simpler sum-of-products form, usually involving fewer product terms or literals.

Netlist

A list of circuit elements and their interconnections.

Node

An identifiable point in a design. A node can be associated with a specific device, geographic location, or signal.

Nominal Delay

The mean time signals take to propagate through a logic element or a wire. The effect of an input change to an element on the output does not occur until after the nominal delay.

Observability

In addition to the control offered by preload, testing requires observability of the internal state of the device following a sequence of vectors. The MACH device offers an observability feature that allows the user to send hidden buried register values to observable output pins. For macrocells that are configured as combinatorial, the observability function suppresses the selection of the combinatorial output by forcing the macrocell output multiplexer into registered output mode. The observability function allows observation of the associated registers by overriding the output enable control and enabling the output buffer.

Option List

A list that appears when you press [F2] in an option field of a software form. You select an option from the list or change the specification in the selected field.

Output Macrocell

The output macrocell sends its output back to the switch matrix, via internal feedback, and to the I/O cell. The feedback is always available regardless of the configuration of the I/O cell. This allows for buried combinatorial or registered functions, freeing the I/O pins for use as inputs if not needed as outputs.

PAL

See Programmable Array Logic.

PAL Blocks

The PAL blocks can be viewed as independent PAL devices on the chip. Each PAL block contains a product-term array, a logic allocator, macrocells, and I/O cells. PAL blocks communicate with each other only through the switch matrix. Additionally, each PAL block contains an asynchronous reset product term and an asynchronous preset product term. This allows the flip-flops within a single PAL block to be initialized as a bank.

PALASM	The PALASM software development system runs on PC/AT compatible and 386-based systems. The package provides low-cost CAD capabilities for the following design phases: design entry, implementation, verification, programming and testing. The software operates with an easy-to-use pull-down menu interface that allows most operations to be performed with a single keystroke. Designs can be entered using mixed schematic, state machine, and Boolean input formats.
Programmable Array Logic	A programmable logic architecture having two levels of logic with a programmable AND array.
Parse	A transparent software routine that checks the syntax of the design file and creates the intermediate design.TRE and design.pin files.
PDS File	A textual representation of a design file using PALASM language constructs.
PLA	See Programmable Logic Array.
PLD Device	A general category of programmable logic devices that contains the two subcategories of PALs and PLAs.
Power-up Reset	All flip-flops power-up to a logic LOW for predictable system initialization. The actual values of the outputs of the MACH devices depend on the configuration of the macrocell. The VCC rise must be a monotonic and the reset delay time is 1000 ns maximum.
Product-Term Array	A product-term array consists of of a number terms that form the basis of the logic being implemented. The product terms drive the logic allocator, which allocates the product terms to the appropriate macrocells. The number of product terms allocated to each array is not fixed and the full sum of products is not realized in the array. The inputs to the AND gates come from the switch matrix, and are provided in both true and complement forms for efficient logic implementation. There are several three-state product terms that provide three-state control to the I/O cells.

Product-Term Steering	An automatic software process that borrows supplemental terms from adjacent macrocells, in addition to the four product terms available to each macrocell. This may occur when the maximum gate width is set to 12 and you include more than four product terms in an equation. In MACH 110 designs, up to eight product terms can be borrowed from adjacent macrocells for a total of 12; four product terms can be borrowed from the adjacent macrocell above and below. In MACH 210 designs, up to 12 product terms can be borrowed from adjacent macrocells for a total of 16.
Programmable Logic Array	A rectangular array of AND and OR gates used to generate logic functions in sum-of-products form.
Programmable Read-Only Memory	A ROM that can be programmed by the customer.
PROM	See Programmable Read-Only Memory.
Reference Designator	The reference designator is used to create unique net and block names. See Macro.
Register Preload	All registers on the MACH devices can be preloaded from the I/O pins to facilitate functional testing of complex state machine designs. This feature allows direct loading of arbitrary states, making it unnecessary to cycle through long test vector sequences to reach a desired state. In addition, transitions from illegal states can be verified by loading illegal states and observing proper recovery.
Reserved Word	See Keyword.
Schematic Diagram	A circuit diagram in which components are represented by standard, simple, easily drawn symbols.

Security Bit	A security bit is provided on the MACH device as a deterrent to unauthorized copying of the array configuration patterns. Once programmed, this bit defeats readback of the programmed pattern by a device programmer.
Signal Contention	Conflicts between signal assignments that arise when you merge one PDS file with another. For instance, input signals may have the same location; this may or may not be appropriate for your design.
Spike	The output condition where the inputs are being manipulated faster than the element's propagation delay.
Switch Matrix	The switch matrix provides communication between PAL blocks and routes inputs to these blocks. The switch matrix takes all dedicated inputs, I/O feedback signals, and buried feedback signals and routes them as needed to the various PAL blocks. The switch matrix makes the MACH device more than just multiple PAL devices on a single chip.
Unit-Delay	A simulation technique used to verify functionality. In this technique all the delays of the elements are set to one time unit.
Unit-Delay Simulation	A simulation that assumes equal propagation delays.
Zero-Delay Simulator	A simulator used for functional validation. The signals have no delay.
Zero-/Unit-Delay Simulator	Combination of the zero-delay and unit-delay simulation elements and with unit-delay circuits inserted into the feedback lines and memory elements.

INDEX

A

- Abandon input command, 9–20
- Active-high polarity, 11–16
- Active-low polarity, 11–15
- Adjacent macrocell use, 5–14
- Analyze device resources, 5–7
- Annotated
 - Datasheet, 7–11
 - Schematic, 7–3
- Assigning logic to a block, 4–53
- Assigning pin and node locations, 5–11
- Assigning resources, 5–28
- Assigning state bits, 4–38
 - Automatic state-bit assignment, 4–38
 - Choosing state-bit assignments, 4–39
 - Manual state-bit assignment, 4–39, 4–41
- ASSIGNMENT OPERATOR, 10–14
- AUTHOR, 10–18
- AUTOEXEC, 1–12
- Automatic state-bit assignment, 4–38
- Auxiliary simulation file command, 9–51

B

- Back annotate signals command, 9–56
- Bank output enable, 4–11
- Banking set and reset in MACH devices, 4–17
- Begin new design command, 9–8
- Bind Pins/Nodes command, 9–26
- Bind signals together, 4–79
- Block and macrocell diagrams, 11–39
- Block partitioning, 5–3, 5–25
- BOOLEAN EQUATION, 10–20
- Boolean design strategies, 4–7
- Boolean-equation elements, 10–4
- Both command, 9–55

- Buried register with /Q feedback, 11–28
- Buried register with Q feedback, 11–30

C

- Capturing a schematic, 7–6
 - Defining preset and reset functions, 7–9
 - Grouping signals into a block, 7–7
 - Interpreting reference designators, 7–10
 - Manually splitting product terms, 7–8
 - Naming signals, 7–10
 - Specifying pin and node numbers, 7–6
 - Terminating unused inputs and outputs, 7–8
 - Turning minimization off, 7–7
- CASE, 4–22, 10–24
- Change directory command, 9–32
- Changes after successful fitting, 5–58
 - Changing logic, 5–59
 - Changing the pin out, 5–58
- Changing logic, 5–59
- Changing the pin out, 5–58
- CHECK, 10–30
- CHECKQ, 10–34
- CHIP, 10–38
- Choosing a larger MACH device, 5–57
- Choosing state-bit assignments, 4–39
- .CLKF, 10–40
- CLKF, 10–44
- Clock control, 11–10
 - Common external clock control, 11–10
 - Individual product term clock control, 11–11
- CLOCKF, 10–46
- Clock signals, 5–8
- Clocking a state machine, 4–44
- .CMBF, 10–48
- COMBINATORIAL, 10–50
- Combinatorial logic, 11–19
- Combinatorial output with I/O feedback, 11–32
- Combined entry methods, 4–6

Combining schematic and text descriptions, 4–64

Command conventions, 9–5

Commands

Abandon input, 9–20

Auxiliary simulation file, 9–51

Back annotate signals, 9–56

Begin new design, 9–8

Bind pins/nodes, 9–26

Both, 9–55

Change directory, 9–32

Compilation options, 9–37

Compile, 9–53

Control file for schematic design, 9–50

Convert schematic to text, 9–55

Current disassembled file, 9–66

Delete specified files, 9–33

Design file, 9–61

Disassemble from, 9–56

Edit a file, 9–21

Edit pin/node list, 9–28

Execute, 9–59

Execution log file, 9–61

Get next input file, 9–18

Go to system, 9–48

JEDEC data, 9–63

List combined files, 9–19

Logic synthesis options, 9–44

Merge design files, 9–16

Merge files, 9–19

Options, 9–30

Other file, 9–51

Other operations, 9–55

Quit, 9–20, 9–48

Recalculate JEDEC checksum, 9–58

Reports, 9–62

Resolve detectable conflicts, 9–22

Retrieve an existing design, 9–15

Save, 9–20

Schematic file, 9–50

Set renaming strategy, 9–31

Set up, 9–34

Simulation, 9–54

Simulation data, 9–64

Simulation options, 9–43

Text file, 9–49

Translate from PLPL, 9–59

View the output buffer, 9–22

Waveform display, 9–65

Working environment, 9–34

Commands and options, 9–7

COMMENT, 10–52

Common external clock control, 11–10

Common external or individual product term control,
11–8

Common external output, 11–7

COMPANY, 10–54

Compatibility, 4–67

Compilation / fitting, 5–1/5–4

Compilation options command, 9–37

Compile command, 9–53

Complement array, 11–35

Condition equations, 4–31

CONDITIONS, 10–56

CONFIG.SYS, 1–12

Configuration, 1–11

Conflicts table, 4–75

Connection problem (wiring congested), error F590,
5–51

Connection status, 5–35

Control file for schematic design command, 9–50

Controlling

Banking set and reset in MACH devices,
4–17

Clocks with .CLKF, 4–16

Logic reduction, 4–25

Minimization, 4–54

Output buffers using .TRST, 4–11

Bank output enable, 4–11

Grouped output enable, 4–14

Individual output enable, 4–12

Polarity

From the equation, 4–8

From the pin or node statement, 4–9

Set/reset using .SETF and .RSTF, 4–17,
4–56

Conversion, existing schematics to MACH-device
designs, 4–62

Convert schematic to text command, 9–55

Creating a simulation file, 6–3

Creating equivalent logic, 4–10

Creating state-machine equations, 4–30

Condition equations, 4–31

Output equations, 4–32

State-machine example, 4–32

Transition equations, 4–31

Current disassembled file command, 9–66

D

- D flip-flop, 11–20
- DATE, 10–60
- DECLARATION SEGMENT, 10–62
- Default branches, 4–34
 - Example, 4–36
 - Global defaults, 4–35
 - Local defaults, 4–35
- DEFAULT_BRANCH, 10–66
- DEFAULT_OUTPUT, 10–70
- Defining
 - Moore and Mealy machines, 4–29
 - Preset and reset functions, 7–9
- Delete specified files command, 9–33
- Deleting unused logic, 4–59
- Design documentation, 4–60
- Design evaluation, 4–67
- Design file command, 9–61
- Design flow, 4–3
- Designing to fit, 5–5
 - Analyze device resources, 5–7
 - Assigning pin and node locations, 5–11
 - Grouping logic, 5–15
 - Methodology, 5–6
 - Setting compilation and fitting options, 5–16
- Device polarity, 11–15
 - Active-high polarity, 11–16
 - Active-low polarity, 11–15
 - Programmable polarity, 11–18
- Device resource checks, 5–24
- Disassemble from command, 9–56
- Documentation menu, 9–69
 - Help on errors, 9–71
 - Index of topics, 9–69
 - Language reference, 9–70
- Download menu, 9–68

E

- Edit a file command, 9–21
- Edit combined data, 4–83
- Edit menu, 9–49
- Edit Pin/Node List command, 9–28
- Editor menu, 9–21
- Electronic signature, 11–36
- Entry, 4–1

EQUATIONS SEGMENT, 10–72

Errors

- F580, 5–39
 - F590, 5–51
 - F600, 5–47
 - F610, 5–41
 - F620, 5–54
- Execute command, 9–59
- Execution log file command, 9–61
- Expand
 - All PT spacing, 5–19
 - Small PT spacing, 5–19
- EXPRESSION, 10–74

F

- [F1] for help, 9–72
 - Feedback map, 5–32
 - Feedback, 11–23
 - Programmable, 11–23
 - Output with I/O feedback, 11–23
 - Output with /Q feedback, 11–25
 - Output with I/O and /Q (dual) feedback, 11–26
 - Buried register with /Q feedback, 11–28
 - Buried register with Q feedback, 11–30
 - Registered input with /Q output, 11–31
 - Non-programmable
 - Combinatorial or registered output with I/O feedback, 11–32
 - Registered output with /Q feedback, 11–32
- FILE Menu, 9–8
- Files menu, 9–18
- Fitting options, 5–56
- Fitting process, 5–3
 - Block partitioning, 5–3
 - Initialization, 5–3
 - Resource assignment, 5–4
- Fixing node locations, 4–51
- Fixing pin locations, 4–50
- Flags used, 5–22
- Flip-flops, 6–8
- FLOATING PINS AND NODES, 10–78

FOR loop, 6–15
FOR-TO-DO, 10–82
Forms

 Schematic-based design, 9–12
 Text-based design, 9–10

FUNCTIONAL EQUATIONS, 10–86
Fuse data only, 9–63
Fuse map, 9–62

G

Gate splitting, 5–16
General PLD language syntax, 11–6
 Clock control, 11–10
 Combinatorial logic, 11–19
 Complement array, 11–35
 Device polarity, 11–15
 Electronic signature, 11–36
 Feedback, 11–23
 Observability product term control, 11–34
 Output-enable control, 11–7
 Preset control, 11–11
 Reset control, 11–13
 Registered or latched logic, 11–20
 Preload control, 11–33
Get next input file command, 9–18
Global defaults, 4–35
Global product term control, 11–12, 11–14
GND, 10–90
Go to system command, 9–48
Group statements with MACH block names, 4–85
GROUP, 10–92
Grouped output enable, 4–14
Grouping
 Logic, 5–15
 Signals into a block, 7–7

H

Hardware requirements, 1–2
Help on errors, 9–71
History file, 6–11

I

IF-THEN-ELSE, 4–21, 6–16
IF-THEN-ELSE, EQUATIONS, 10–96
IF-THEN-ELSE, SIMULATION, 6–16, 10–100
Individual output enable, 4–12
Individual product term clock control,
 11–7, 11–11/11–13
Initializing a state machine, 4–43
Input files, 4–66
Inputs, clock signals, and set/reset control, 4–69
Installation, 1–4
 File updates, 1–12
 Requirements
 Hardware, 1–2
 Software, 1–3
 Steps, 1–4
Interconnection resources, 5–10
Internal nodes, 6–8
Interpreting reference designators, 7–10

J

.J EQUATION, 10–102
JEDEC data command, 9–63

K

.K EQUATION, 10–104

L

Large functions at the end of a block, 5–14
Large logic functions, 5–13
Latch, 11–22
LATCHED, 10–106
Latches, 6–9
List combined Files command, 9–19
LOCAL DEFAULT, 10–108
Local defaults, 4–35
Logic map, 5–33
Logic synthesis options command, 9–44

M

- MACH 1 and MACH 2 series devices, 11–156
 - Device features, 11–156
 - PALASM programming features, 11–157
 - Pin and node descriptions, 11–157
 - Sample equations, 11–168
- MACH 110 device, 11–174
- MACH 120 device, 11–178
- MACH 130 device, 11–182
- MACH 210 device, 11–186
- MACH 215 device, 11–198
- MACH 220 device, 11–190
- MACH 230 device, 11–194
- MACH device features, 11–156
- MACH error messages, 5–36
 - Connection problem (wiring congested), error F590, 5–51
 - Mapping difficulty – no feasible solution, error F620, 5–54
 - Marginal block partitioning measure, warning F120, 5–37
 - Not all input signals were connected, error F600, 5–47
 - Partitioning could not place all signals into blocks, error F580, 5–39
 - Procedures for reducing logic complexity, 5–55
 - Product term distribution, error F610, 5–41
- MACH I/O cell and macrocell, 11–169
- MACH library, 4–48, 7–2
 - Quick reference, 8–1/8–17
- MACH output files, errors and warnings, 5–35
- MACH pin and node
 - Descriptions, 11–157
 - Feedback, 11–170
- MACH registered and latched inputs, 11–171
- MACH report, 5–21, 9–62
 - Assigning resources, 5–28
 - Block partitioning, 5–25
 - Connection status, 5–35
 - Device resource checks, 5–24
 - Feedback map, 5–32
 - Flags used, 5–22
 - Logic map, 5–33
 - Pair analysis, 5–23
 - Pin map, 5–34
 - Preplacement & equation usage checks, 5–23
 - Signals, equations, 5–30
 - Signals, tabular, 5–29
 - Timing analysis for signals, 5–23
 - Utilization, 5–27
- MACH sample equations, 11–168
 - I/O cell and macrocell, 11–169
 - Pin and node feedback, 11–170
 - Registered and latched inputs, 11–171
- MACH_SEG_A, 10–110
- Macrocells and I/O pins, 5–9
- Manual state-bit assignment, 4–39, 4–41
- Manually splitting product terms, 7–8
- Mapping difficulty – no feasible solution, error F620, 5–54
- Marginal block partitioning measure, warning F120, 5–37
- MASTER_RESET, 10–114
- Maximize packing of logic blocks, 5–18
- MEALY_MACHINE, 10–116
- Menus, 9–7
 - Documentation menu, 9–69
 - Download menu, 9–68
 - Edit menu, 9–49
 - Editor menu, 9–21
 - [F1] for help, 9–72
 - File menu, 9–8
 - Files menu, 9–18
 - Resolution menu, 9–22
 - Run menu, 9–52
 - Setup menu, 9–30
 - View menu, 9–60
- Merge design files command, 9–16
- Merge files command, 9–19
- Merge files, 4–81
 - Edit combined data, 4–83
 - Save combined data, 4–84
- Merge guidelines, 4–69
- Merging multiple PDS files, 4–66
 - Compatibility, 4–67
 - Design evaluation, 4–67
 - Guidelines, 4–69
 - Input files, 4–66
- MINIMIZE_OFF, 10–118
- MOORE_MACHINE, 10–120

N

- Naming signals, 7–10
- Netlist report command, 9–67
- NODE, 10–122
- Non-programmable feedback, 11–32
 - Combinatorial or registered output with I/O feedback, 11–32
 - Registered output with I/O feedback, 11–33
 - Registered output with /Q feedback, 11–33
- Not all input signals were connected, error F600, 5–47

O

- Observability product term control, 11–34
- OPERATOR, 10–126
- Options, 9–30
- Other file command, 9–51, 9–68
- Other operations command, 9–55
- .OUTF, 10–128
- Output enable, 6–9
- Output equations, 4–32
- Output files, errors and warnings, 5–35
- Output polarity, 4–7
 - Controlling polarity from the equation, 4–8
 - Controlling polarity from the pin or node statement, 4–9
 - Creating equivalent logic, 4–10
 - The two components of polarity, 4–7
- Output with I/O and /Q (dual) feedback, 11–26
- Output with I/O feedback, 11–23
- Output with /Q feedback, 11–25
- Output-enable control, 11–7
 - Common external or individual product term control, 11–8
 - Common external output, 11–7
 - Individual product term control, 11–7
- OUTPUT_ENABLE, 10–132
- OUTPUT_HOLD, 10–134

P

- Pair analysis, 5–23
- PAIR, 10–136
- PALASM programming features, 11–157

- PAL16RA8, 11–48
- PAL16V8HD, 11–52
- PAL20EG8, 11–62
- PAL20EV8, 11–66
- PAL20RA10, 11–70
- PAL22V10, 11–80
- PAL32VX10, 11–130
- PALCE22IP6, 11–74
- PALCE23S8, 11–84
- PALCE26V12, 11–92
- PALCE29M16, 11–96
- PALCE29MA16, 11–104
- PALCE610, 11–142
- Partitioning could not place all signals into blocks, error F580, 5–39
- PATTERN, 10–140
- Pin and Node Descriptions, 11–38
- Pin map, 5–34
- PIN, 10–142
- Pinout, 9–67
- PLD device feature cross-reference table, 11–4
- PLD Naming Conventions, 11–2
- PLS30S16, 11–110
- PLS105, 11–40
- PLS167/168, 11–44
- PRELOAD, 10–146
- Preload control, 11–33
 - Product term control, 11–33
 - Supervoltage, 11–33
- Preloaded registers, 6–9
- Preplacement & equation usage checks, 5–23
- Preset control, 11–11
 - Global product term control, 11–12
 - Individual product term control, 11–13
- .PRLD, 10–148
- PRLDF, 10–150
- Procedures for reducing logic complexity, 5–55
- Product term control, 11–34
- Product term distribution, error F610, 5–41
- Product terms, 5–9
- Programmable feedback, 11–23
 - Buried register with Q feedback, 11–30
 - Buried register with /Q feedback, 11–28
 - Output with I/O feedback, 11–23
 - Output with I/O and /Q (dual) feedback, 11–26

- Output with /Q feedback, 11–25
- Registered input with /Q output, 11–31
- Programmable polarity, 11–18

Q

- Quit command, 9–20, 9–48

R

- .R EQUATION, 10–152
- Radix notation, 4–20
- Re-engineer the combined design, 4–84
 - Group statements with MACH block names, 4–85
 - Shared resources, 4–85
 - Simulation commands, 4–85
- Recalculate JEDEC checksum command, 9–58
- REGISTERED, 10–154
- Registered input with /Q output, 11–31
- Registered or latched logic, 11–20
 - D flip-flop, 11–20
 - Latch, 11–22
 - SR flip-flop, 11–21
- Registered output with I/O feedback, 11–32
- Registered output with /Q feedback, 11–33
- Rename signals in the input buffer, 4–76
- Reports command, 9–62
- Reset control, 11–13
 - Global product term control, 11–14
 - Individual product term control, 11–13
- Resolution menu, 9–22
- Resolve conflicts, 4–74
 - Bind signals together, 4–79
 - Rename signals in the input buffer, 4–76
 - Review detectable conflicts table, 4–75
- Resolve Detectable Conflicts, 9–22
- Resource assignment, 5–4
- Retrieve an existing design command, 9–15
- Retrieve files, 4–72
- REVISION, 10–156
- .RSTF, 10–158
- Run menu, 9–52

S

- .S EQUATION, 10–160

- Sample equations, 11–168
- Save combined data, 4–84
- Save command, 9–20
- Schematic entry, 4–4
- Schematic file command, 9–50
- Schematic parameters, 4–49
 - Assigning logic to a block, 4–53
 - Controlling minimization, 4–54
 - Controlling set/reset, 4–56
 - Deleting unused logic, 4–59
 - Fixing node locations, 4–51
 - Fixing pin locations, 4–50
- Schematic versus text entry, 4–47
- Schematic-based design form, 9–12
- .SETF, 10–162
- SETF, 10–164
- Set Renaming Strategy, 9–31
- Set Up command, 9–34
- Set/reset signals, 5–8
- Setting compilation and fitting options, 5–16
 - Expand all PT spacing, 5–19
 - Expand small PT spacing, 5–19
 - Gate splitting, 5–16
 - Maximize packing of logic blocks, 5–18
- Setup Menu, 9–30
- Shared resources, 4–85
- Signals, equations, 5–30
- Signals, tabular, 5–29
- SIGNATURE, 10–166
- SIMULATION, 10–170
- Simulation, 6–1
 - Command, 9–54
 - Commands, 4–85
 - Considerations, 6–7
 - Flip-flops, 6–8
 - Internal nodes, 6–8
 - Latches, 6–9
 - Output enable, 6–9
 - Preloaded registers, 6–9
 - Verified signal values, 6–10
 - Summary, 6–3
- Simulation data command, 9–64
- Simulation design examples, 6–17
 - Boolean equation design, 6–17
 - State-machine design, 6–20
- Simulation options command, 9–43
- Simulation segment versus auxiliary file, 6–5

Software requirements, 1–2
Special programming features, 11–39
Specifying outputs in IF-THEN-ELSE and CASE statements, 10–6
Splitting gates, 5–16
SR flip-flop, 11–21
Standard PLD Devices versus non-standard PLD devices, 11–3
START_UP, 10–172
STATE, 10–176
STATE ASSIGNMENT EQUATION, 10–180
STATE EQUATIONS, 10–184
State-machine constructs, 10–5
State-machine design strategies, 4–26
State-machine example, 4–32
STATE OUTPUT EQUATION, 10–188
STATE TRANSITION EQUATION, 10–192
State segment overview, 4–28
Steering product terms, 11–178
Strategies, designs that don't fit, 5–21
STRING, 10–194
Supervoltage, 11–33
Syntax and examples, 10–12

T

.T EQUATION, 10–202
.T1 EQUATION, 10–204
.T2 EQUATION, 10–206
Terminating unused inputs and outputs, 7–8
TEST, 10–198
Text-based design form, 9–10
Text entry, 4–4
Text file command, 9–49
Timing analysis for signals, 5–23
TITLE, 10–208
Trace file, 6–12
TRACE_OFF, 10–210
TRACE_ON, 10–212
Transition equations, 4–31
Translate from PLPL command, 9–59
.TRST, 10–214
Turning minimization off, 7–7

U

Using high-level constructs, 4–18
CASE statement, 4–22
IF-THEN-ELSE statement, 4–21
Radix notation, 4–20
Vector notation, 4–18
Using simulation constructs, 6–15
FOR loop, 6–15
IF-THEN-ELSE, 6–16
WHILE loop, 6–15
Using state bits as outputs, power-up and clock equations, 4–42, 4–45
Utilization, 5–27

V

VCC, 10–218
Vector notation, 4–18
VECTOR, 10–220
Vectors + fuse data, 9–63
Verified signal values, 6–10
View menu, 9–60
View the output buffer command, 9–22
Viewing simulation results, 6–11
History file, 6–11
Trace file, 6–12

W

Warning F120, 5–37
Waveform display command, 9–65
WHILE loop, 6–15
WHILE-DO, 10–222
Working environment command, 9–34

NOTES

NOTES



**ADVANCED
MICRO
DEVICES, INC.**

901 Thompson Place
P.O. Box 3453
Sunnyvale
California 94088-3453
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL-FREE
(800) 538-8450

**APPLICATIONS HOTLINE &
LITERATURE ORDERING**
(800) 222-9323
(408) 749-5703

**SUPPORT PRODUCTS
ENGINEERING HOTLINE**
JAPAN 0-031-11-1129
UK 0-800-89-1131
USA (800) 222-9323



RECYCLED &
RECYCLABLE

Printed in USA
CPS-3M-6/92-0
12379C