IS—68K CPU

INTEGRATED SOLUTIONS 68K CPU HARDWARE REFERENCE MANUAL

REVISION 1.0

March, 1984

INTEGRATED SOLUTIONS, INC.
2240 Lundy Ave.
San Jose, Ca. 95131

Preliminary Edition, January 1983
1st Edition, March 1984

The material in this manual is for information only and is subject to change without notice.

Integrated Solutions, Inc., assumes no responsibility for any errors which may appear in this manual.

- Class A Computing Devices:

NOTICE:

This equipment generates, uses and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference in which case the user at his own expense may be required to take measures to correct the interference.

DEC, LSI-11, and RL02 are trademarks of Digital Equipment Corporation.

PREFACE

This manual describes the Integrated Solutions IS-68K, LSI-11 Bus Compatible 68000/68010-based CPU Board. It contains specifications, installation procedures, a technical description and limited troubleshooting information for the IS-68K. It also contains initialization and programming procedures for the on-board serial ports and counter/timer chips. Part of the technical description assumes that the user is somewhat familiar with the architecture of the 68000/68010 processors. For more information on the architecture and programming of the 68000 family, see the appropriate Motorola, Hitachi, Rockwell or Signetics reference manuals.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

1.1 Features

The IS-68K card is a high performance, highly integrated CPU on the LSI-11 Bus. It is specifically designed for those users who want to combine the processing speed, large instruction space and sophisticated instruction set of the Motorola 68000/68010 processors with the large variety of peripherals available on the LSI-11 bus. In addition, the card offers a higher level of integration than the existing LSI-11 processors, combining up to 1 megabyte of parity memory, up to 32k bytes of PROM, and two high speed serial ports with programmable baud rates on a single Quad form factor card.

1.2 Performance

The IS-68K has been designed to maximize the performance available from the 68000/68010 processors. It will operate at 10MHz (12MHz -12 option) with no wait states out of the on-board or local bus expansion memory. Dual porting the on-board and local bus memory permits all on-board and local bus memory to be accessed by DMA devices on the LSI-11 Bus and also permits the 68000/68010 to continue operating while DMA transfers are underway. The 68000 continues executing at full speed while DMA transfers are occuring to LSI-11 bus memory. It continues executing at reduced speed if DMA accesses to on-board/local bus memory are made, while memory accesses are automatically arbitrated between the 68000 and the DMA bus master.

1.3 Specifications

1.3.1 Processor

The processor of the IS-68K CPU board is a Motorola 68000 or 68010. There are no hardware differences between the 68000 and 68010 versions of the IS-68K. All IS-68K boards have hardware support for demand paging.

1.3.2 Bus

The system bus of the IS-68K is LSI-11 compatible. It meets all requirements of DEC STD 160 WITH 22-bit addressing. The "B" revision of the processor board (released March, 1984) supports block mode transfers as defined in DEC STD 160.

1.3.3 Address Space

A 22-bit logical address space is mapped to a 22- bit physical address space through the memory management subsystem. The memory management subsystem together with the user mode of the 68000/68010 processors provides the protection features required to support a large multiuser environment.

1.3.4 Memory

Configurations:

128k, and 256k	implemented with 64k RAMS.
512k, and 1024k	implemented with 256k RAMS.

Byte parity generation and checking implemented on all versions.

1.3.5 Memory Management

The IS-68K board contains two levels of memory management, a segmentation front end and a paging back end. Each process can have from 1 to 64 segments associated with it. Segments contain from 1 to 16 4k byte pages. Support for demand paging through page accessed and page modified bits is associated with each page. These bits are automatically updated by hardware on each page reference.

1.3.6 Local Memory Bus

The on-board memory is expandable to 4 megabytes over the local memory bus. Local memory, as well as on-board memory runs with no wait states. All local memory is accessible from the LSI-11 bus through the two port memory arbitrator.

1.3.7 LSI-11 Bus Memory

All memory accesses above the local memory are routed over the LSI-11 bus. The transition between local and LSI-11 bus memory is controlled by the local memory limit switch on the IS-68K (see Section 1.3.12).

1.3.8 PROM Sockets

The IS-68K contains two on-board PROM sockets configurable for 2716, 2732, 2764 or 27128 ROMS or EPROMS. The IS-68K is optionally available with two sets of EPROMs. On the IS-68K, the PROMs are located starting at address C00000H in the 68000/68010 on-board I/O space. In addition some special hardware has been added to the CPU board to map the bottom eight bytes of system mode logical address space (addresses 000000H to 000007H) into the bottom eight bytes of the PROM space (addresses C00000H to C00007H). In this manner when the IS-68K board powers up, the initial system stack pointer and initial PC are always taken as long words from the bottom eight bytes of the PROM.

MACS PROMs - The MACS PROMs are a set of 2732 EPROMs containing the Motorola MACSBUG monitor. A description of the commands supported under this monitor is contained in Appendix B.

UNIPS PROMs - The UNIPS PROMs are a set of 2764 EPROMs containing the UNIX bootstraps required to boot System 3 or System 5 Unix from a variety of different DEC compatible devices. For BSD 4.2 Unix the PROMs are 27128s because of the larger number of drivers and higher complexity of the bootstrap device drivers.

1.3.9 Serial Ports

The two serial ports on the board both support either RS423 (RS232C compatible) asynchronous communication or RS422 balanced synchronous communication with external clocking. Both channels offer full modem control support while in asynchronous mode and limited modem control support in synchronous mode. Channels 0 and 1 can be independently configured for synchronous or asynchronous communications.

1.3.10 Electrical Interface

The IS-68K interfaces with the 22 bit LSI-11 bus. A Quad form factor board is used with the bus interface implemented on A and B connectors. There are no signals on the C and D

connectors. DEC approved bus drivers, receivers and transceivers are used for all bus signals. All AC and DC requirements of DEC Standard 160 are adhered to. The board supports BPOK and BDCOK power up/down protocol, as well as BEVNTL and the generation of SRUN (pin AF1) signal. By using the CD connectors for power and ground only, the board is compatible with all LSI-11 bus backplanes.

1.3.11 Indicators

4 LEDs at the top of the board:

1 Motorola 68000 HALT indicator
1 latched PWR FAIL
1 latched LOCAL PARITY ERROR
1 latched BUS PARITY ERROR

1.3.12 Configuration Switches (8 switches)

SW 1-4 - Local memory limit switch: All memory accesses above this limit go to the LSI-11 bus.

SW 5-6 - Default baud rate of console port: settable to 300, 1200, 2400 or 9600 baud. Settable to any other four standard baud rates (up to 38.4K), at the time of order upon customer request. Moving the jumper associated with pins E28-E29-E30 will also halve the baud rates produced. (See section 2.1.5.). These switches set the baud rate according to the above definition with both the MACS and UNIPS PROMs. However, the switches are software readible and the on-board firmware must make the conversion between switch setting and programmed baud rate.

SW 7-8 - Used by the UNIPS PROMs to control autobooting and the type of SMD disk from which to boot. See the appropriate Unix Installation notes for a more detailed description.

1.3.13 Power Requirements

+ 5v - 4.0 amps typical, 5 amps maximum
+ 12v - .1 amps maximum

1.3.14 Environmental

Temperature: 0 ° C to 50 ° C (operating)
-40 ° C to 65 ° C (non-operating)

Humidity: 10% to 90% (non-condensing)

1.3.15 Options

- MACS two 2732 EPROMs containing Motorola MACSBUG power up diagnostics and autoboot capability (no PROMs standard)

- UNIPS two 2764 E Proms containing self test diagnostics MMU initialization logic, and Unix bootstrap loaders (required for Unix)

- 10 10 MHz board (8MHz standard)

- 12 12 MHz board

- 10V 10 MHz board with 68010 virtual memory processor

- 12V 12 MHz board with 68010 virtual memory processor (available 2nd quarter 1984)

- 128 128k bytes of parity memory (256k standard)

- 512 512k bytes of parity memory (available 2nd qtr 84)

- 1024 1024k bytes of parity memory(available 2nd qtr 84)

1.4 Memory Management

The memory management unit is based on 5 high speed static 1kx4 RAMS and consists of a segmentation front end followed by a paged back end. This is the same memory management style used on many current generation mainframe computers. The segmentation unit takes bits 21-16 of the 68000 logical address and uses it as an index into the segment register bank along with the contents of the context register (4 bits) to be described in more detail later. The segment register array consists of 1024 registers each 8 bits wide. The combination of 6 logical address bits and 4 context register bits select one unique segment register on each memory access. Out of the segment register bank come 8 bits, the upper two of which are used for to encode four protection modes and the bottom six of which are used as an index into the page table. The two protection bits are encoded as follows:

00 - no access

01 - read and execute access only

10 - read/write access

11 - execute access only in user mode. Any access permitted in System mode

1.4.1 Page map

The six segment number bits coming out of the segment register are concatenated with bits 15-12 of the logical address to form a 10 bit entry into the page table. The page table is 1024 registers, each 12 bits wide. Out of the page table comes a 10 bit physical page number which is combined with bits 11-0 of the 68000 logical address to a form a 22 bit physical memory address. From the programmers point of view, each process consists of from 1-64 segments, each segment containing from 1 to 16 4k pages. This page size should be optimal for a large number of processes to be simultaneously memory resident without wasting memory space due to unused partial pages and at the same time minimizing the number of memory management registers that must be changed to load a new process. The 3FCH-3FFH number pages have special meanings, two pages (3FCH and 3FDH) being used as the non-existant page and two pages (3FEH and 3FFH) being used as the I/O page for the LSI-11 bus. The other two bits coming out of the page register are used as page

accessed and page dirty bits for demand paging. They are automatically updated by the hardware on each memory access.

1.4.2 Context register

In an effort to minimize context switching overhead, there are actually 16 sets of 64 segments present in the segment map. A four bit context register, settable only by the system, selects which user context (1 -15) is to be used when a user program is running. Whenever the 68000 is in supervisor mode, the system context (0) is automatically used. The user context, however, is preserved when a switch is made from user mode to system mode and back to user mode such as when servicing interrupts. Thus, when 15 or fewer processes are simultaneously memory resident, context switching only requires modifying the context number in the context register.

1.5 On-board/Local Memory

The IS-68K CPU board comes equipped with 256k bytes (128k bytes optional) of on-board memory. This memory is implemented with high speed 64k dynamic RAMS and accesses from the 68000 are accomplished with no wait states. This type of performance is available because the on-board memory is dual-ported; the 68000 accesses it via a different path than the LSI-11 bus and does not have to follow LSI-11 bus protocol. For high performance applications, the on-board memory can be extended to the full 4 megabyte address space of the 22 bit LSI-11 bus via the two 34 pin local bus expansion connectors at the top of the IS-68K card. When configured this way, the IS-68K allows a full 4 megabyte 68000 system to run with no wait states at 8, 10 or 12 MHz.

All on-board local bus memory is fully accessible from the LSI-11 bus when DMA devices are bus master. Further, because of the performance limitations of the LSI-11 bus only 50% of the on- board memory bandwidth can be used even if a device transfers at the ideal maximum LSI-11 bus rate. The dual-ported memory system allows the remaining memory bandwidth to be used by the 68000.

NOTE

> The first 256k bytes of memory always appears at the bottom of the LSI-11 bus physical memory space. If more memory is present on the local bus, it appears at the next ascending physical memory locations up to the limit of the local bus memory present.

Users of the first IS-68K boards with only 128k bytes (no longer available) also have a further limitation. Because the address limit switch selects addresses with a granularity of 256k bytes, the IS-68K always thinks that it has at least 256k of on-board memory. This means that with 128k byte boards, a 128k hole appears above the on-board memory where the other 128k bank of memory normally sits. Accesses to this memory space do not result in bus errors but no information can be read or written from this space. The effect of this problem can be negated to a large extent by programming the memory management unit never to access this physical memory space.

1.5.1 Local Memory Bus

The local memory bus is implemented as a means of extending the on-board memory with its fast access characteristics and dual- ported nature. The local memory bus is implemented with 38 control, data and signal lines on 2 34 pin ribbon cable connectors. The connectors are designated J3 and J4 at the top of the IS-68K.

NOTE

On early versions of the IS-68K, the J4 connector is designated J1.

The following signals are implemented on the J4 connector:

J4-1 XBUFD10 (external buffered data 10)
J4-2 XBUFD09
J4-4 XBUFD15
J4-6 XBUFD14
J4-7 XBUFD11
J4-8 XBUFD13
J4-9 XBUFD12
J4-12 XBUFD08
J4-14 XBPAROUTH (external buffered parity out -most sig byte)
J4-16 XBUFD02
J4-17 XBUFD00
J4-19 XBUFD07
J4-21 XBUFD04
J4-22 XBUFD06
J4-24 XBUFD05
J4-26 XBUFD01
J4-28 XBUFD03
J4-30 XBPAROUTL (external buffered parity out-least sig byte)
J4-32 XBPINH* (external buffered parity in -most sig byte)
J4-34 XBPINL* (external buffered parity in -least sig byte)

The following signals are implemented on the J3 connector:

J3-1 XBAD17 (buffered non-multiplexed address 17)
J3-3 XBCAS* (buffered column address strobe)
J3-5 XBREF (buffered refresh signal)
J3-7 XBRAS* (buffered row address strobe)
J3-9 XBMUX1 (buffered multiplexed address 1)
J3-11 XBMUX5
J3-13 XBMUX0
J3-15 XBMUX7
J3-17 XBMUX6
J3-19 XBMUX3
J3-21 XBMUX2
J3-23 XBMUX4
J3-25 XBAD21
J3-27 XBAD18
J3-29 XBAD19
J3-31 XBAD20
J3-33 XBWRL (buffered write low byte)
J3-34 XBWRH (buffered write high byte)

The signals on the local memory bus can be grouped into four functional groups:

1.5.1.1 The Data Group

The data group consists of the 16 bidirectional data lines, the two byte parity in lines and the two byte parity out lines. The parity in and out lines are unidirectional.

1.5.1.2 The Multiplexed Address Group

There are eight unidirectional multiplexed address lines. These lines contain a physical addresses 1 through 8 at RAS time and physical addresses 9 through 16 at CAS time. The source of these physical addresses depends on whether the cycle is a 68000 cycle or an LSI-11 bus cycle. In addition, during refresh cycles, these lines contain the refresh address.

1.5.1.3 The Non-multiplexed Address Group

These five lines contain non-multiplexed physical addresses 17- 21. The non-multiplexed addresses are guaranteed valid from 35ns before RAS is asserted and are valid through the entire RAS cycle.

1.5.1.4 The Control Group

The five signals in the control group are as follows:

XBRAS* - This signal is asserted to indicate that a memory cycle has begun and that row addresses are valid on XBMUX0-7.

XBCAS* - This signal always follows 40ns after XBRAS* and indicates that the column addresses are valid on XBMUX0-7. In the case of refresh cycles, XBCAS* still occurs but must be supressed at the memory chips themselves.

XBREF - This signal is asserted at least 40ns in advance of RAS to indicate that the forthcoming memory cycle is a memory refresh cycle. The RAS lines of all memory devices must be asserted low at RAS time and CAS must be suppressed to all memory devices.

XBWRH - This signal indicates that a write is to be performed on the most significant byte of the addresses memory location. This signal is always stable at least 0ns before the assertion of XBCAS* and is stable until the negation of XBRAS*.

XBWRL - This signal indicates that a write is to be performed on the least significant byte of the addressed memory location. This signal is always stable at least 0ns before the assertion of XBCAS* and is stable until the negation of XBRAS*.

1.5.2 Local Bus Timing

Figure 1.5 shows the timing requirements of typical read and write signals on the Local Bus. The significant parameters are:

TARasSu - address set up time before RAS* is asserted - 35ns (this parameter applies to both the multiplexed and non- multiplexed addresses)

TARasHd - multiplexed address hold time after RAS* is asserted - 25ns

TACasSu - multiplexed address (column addr) set up time before CAS* is asserted - 0ns

TACasHd - multiplexed address (column addr) hold time after CAS* is asserted - 100ns

TRASPr - minimum RAS precharge time - 85ns (12 MHz) 100ns (10 MHz) 125ns (8 MHZ)

TRASCy - minimum RAS cycle time - 130ns (12 MHz) 150ns (10 MHz) 187ns (8 MHz)

TDRAS - data access time from RAS - 100ns (12 MHz) 120ns (10 MHz) 158ns (8 MHz)

TRC - time interval between RAS assertion and CAS assertion - 45- 55ns

TWCASSu - stable WRITE HIGH and WRITE LOW signals before CAS* is asserted - 0ns

TWDCASSu - stable write data high and low bytes before CAS* is asserted including parity data - 25ns

TWDCASHd - stable write data, high, low bytes and parity, after CAS is asserted - 100ns

1.6 Serial Ports

A combination of a Zilog CTC and a Zilog DART are used to implement two serial ports with programmable baud rate. The serial ports can be used in asynchronous communications at rates up to 38.4 kilobits per second and in synchronous communications up to 1 megabit per second. Almost all characteristics of the serial ports are programmable including the number of bits/word, the number of start bits, the number of stop bits, the number of bits per character, whether interrupts occur on various conditions, the interrupt vector, etc. (See Appendix A, and the Zilog application notes in Appendix E for more information on programming the DART and CTC.). The DART and CTC are properly initialized for operation at the baud rate set in switches 5 and 6 by both the MACS and UNIPS PROMs.

1.7 Traps and Interrupts

1.7.1 Interrupts

The IS-68K uses the seven level interrupt scheme of the 68000 to service interrupts from sources on-board and off-board. The off- board interrupts include the four LSI-11 bus interrupt levels and four hardware related sources of level seven non-maskable interrupts. The 68000 level seven interrupt differs from the other six levels in that it is non-maskable, i.e., no matter what priority level is set, it will still be serviced.

1.7.1.1 Level 1-4 Interrupts

The LSI-11 Bus Interrupts. The four LSI-11 bus interrupts are mapped into the following 68000 interrupt levels:

LSI-11 BUS	Motorola
IRQ4	Level 1
IRQ5	Level 2
IRQ6	Level 3
IRQ7	Level 4

These four interrupts are vectored with the vector coming from the LSI-11 bus. The LSI-11 vector is converted to an address in the 68000 vector table by multiplying it by four. For example, the LSI-11 bus RL02 interrupt at 160(8) will vector through Motorola location 1C0h. The IS-68K does not have the same restriction that the LSI-11 bus does that vectors must have even addresses. The effect of this restriction, however, is to permit the LSI-11 bus to access only every other location in the vector table.

A second restriction requires the LSI-11 bus vectored interrupts to appear in the 68000 user interrupt space from 100h to 3FCh. This means that LSI-11 bus vectors from 100(8) to 377(8) are permitted. This range includes almost all LSI-11 bus standard vectors and

the bottom of the floating vector table.

1.7.1.2 Level 5 Interrupt

The BEVNTL line. The only interrupt at level 5 is the LSI-11 bus BEVNTL line. The BEVNTL line contains a 60Hz TTL level compatible signal generated by DEC or DEC compatible power supplies that is received by the IS-68K and latched on the falling edge. Because the BEVNTL line has no interrupt associated with it, the BEVNT interrupt is autovectored at level 5. This means that it always traps through location 074h.

1.7.1.3 Level 6 Interrupts

The On-Board Serial Ports. The on board serial ports and optionally the counter/timer chip interrupt at level 6. A vector is provided by the serial I/O ports and counter/timer chip at interrupt acknowledge time. Like the LSI-11 bus vectors, these vectors are multiplied by four to get the vector address at the bottom of the 68000 logical address space. Because the Zilog DART chips used for the serial ports and the Zilog CTC chips can modify their base vectors to respond to different interrupt sources, the user must be careful that the on-board level 6 interrupts do not conflict with the LSI-11 bus vectors. (See Appendix A for more information on programming the Zilog DART.)

1.7.1.4 Level 7 Interrupts

Level 7 interrupts are reserved for hardware related problems that must be brought to the immediate attention of the 68000. Because no vectors are associated with level seven interrupts, they are all autovectored through the same location, 07CH.

BPOKH falling edge - The falling edge of the BPOKH signal indicates that the power supply has only four milliseconds of power left. This falling edge generates a level 7 interrupt to get the processors immediate attention. In addition, this occurence is latched into the on-board STATUS register at location C30000H, bit 9.

On-board Parity Error - A parity error in on-board memory causes a level 7 interrupt when the memory access is made from the 68000. The occurence of the parity error is also latched into the STATUS register bit 7. If another device were bus master when the parity error occured in on-board memory, the response would be different. In this case, the response would be to suppress the bus REPLY signal and cause a bus error in the DMA device to indicate to it that the parity error had occured.

Bus Parity Error - The LSI-11 bus defines a protocol where the assertion of both address lines 17 and 18 (BDAL 17 and 18) at data in time indicates that a parity error has occured in the device being accessed. The IS-68K recognizes this protocol and immediately asserts a level 7 interrupt. This occurence is also latched into the STATUS register bit 8.

BHALT line - The falling edge of the LSI-11 bus BHALTL line causes a level 7 interrupt. This occurence is also latched into the STATUS register bit 6.

1.8 DMA Arbitration

The IS-68K board serves as the DMA arbitrator for the LSI-11 system into which it is inserted. DMA arbitration is not handled by the 68000 but is instead handled by a completely separate piece of TTL logic for performance reasons. The arbitrator conforms to LSI-11 bus specifications with a maximum DMA latency (request to grant

time) of 200 ns (150 ns average) except in the situation where the 68000 is using the LSI-11 bus. In this case, the DMA arbitrator cannot issue the bus grant until the 68000 is through with its current bus cycle. If, however, the 68000 is using on-board/local bus memory at the time of the request, the GRANT is always issued with an average latency of 150 ns.

The second case where DMA latency may be affected is when one device has the bus AND another is requesting its use AND the 68000 is requesting its use concurrently. In this case, the 68000 will be given the bus for one cycle only at the conclusion of the first master's use of the bus. After completing one cycle, the second requester will be granted bus mastership. Note that when the 68000 is executing out of on-board/local bus memory, instruction execution and servicing of on-board interrupts can still occur concurrently with DMA transfers.

In the case of off-board interrupts, the 68000 will be suspended in the IACK (interrupt acknowledge) cycle until the current master relinquishes the bus. The 68000 will then conclude the interrupt acknowledge cycle, immediately after which the second master will be granted the bus if another bus request is pending.

1.9 Memory Arbitration

The memory arbitration unit allocates memory cycles among the three possible requesters of the on-board memory in the following priority order:

- 68000 memory requests
- refresh requests (one refresh request occurs every 12.8 microseconds)
- LSI-11 bus requests

The priority order is important only when two or more requesters simultaneously request the use of on-board memory. In all other cases, memory cycles are awarded on a first come-first served basis. Further it is important to note that the on-board memory always runs at the maximum possible rate of a cycle every 260ns (12 MHz board, 300ns - 10MHz, 375ns - 8 MHz) when any requests are pending.

Even though the LSI-11 bus is at the minimum priority level, it is almost always guaranteed an access within one memory cycle. This is because the 68000, even if accessing memory at the maximum rate possible, does not request the use of the bus for one clock cycle after completing each memory cycle. Any LSI-11 memory request pending is honored at this time. The only exception to this is the case where a refresh request was also pending. In this case, the refresh request would be granted and then the LSI-11 bus memory request would be honored. Thus it would theoretically be possible for the LSI-11 bus and the 68000 to both request bus use constantly and alternate memory cycles. This would allow one LSI-11 cycle every 520ns for a memory transfer rate of 3.85 megabytes/second. Unfortunately, the LSI-11 bus protocol will not support memory requests that frequently. Empirical studies of the IS-68K have shown that what actually happens with a DMA device that performs fairly close to the upper limit of the LSI-11 bus bandwidth is that the LSI-11 DMA device will perform a memory cycle every two 68000 cycles for a total transfer rate of 2.4 megabytes per second. In situations where this level of performance is unacceptable, the performance can be improved a little by stopping the 68000 processor while doing DMA transfers. When this is done, the maximum transfer rate increases to 2.7 megabytes per second and is limited only by the memory arbitration time and the LSI-11 bus protocol itself.

The disadvantage of this method of increasing the DMA bandwidth is that the 68000 no longer can perform meaningful work while DMA transfers are underway. However, if the 68000 has nothing meaningful to do while a DMA transfer is pending, it is certainly better to STOP it (by executing the STOP instruction) rather than executing a tight code loop waiting for completion.

Average BDOUTL to REPLYL time on memory writes is 140ns (97ns min - 172ns max 12mhz board). Average BDINL to REPLYL time on memory reads is 315ns (12 mhz board). This time is much longer than the memory access time and is required in order to check the parity of the information coming out of the memory in order to suppress REPLY when a parity error occurs. The impact of slow READ operations is somewhat minimized by the fact that in most operating systems, writes from disk to memory are four times as frequent as reads from memory to disk.

1.9.4 Block Mode Memory Operation

Revision 2.0 of the IS68K board supports block mode memory operation as defined in the latest LSI-11 Bus Technical Specification. Under block mode operation, a single address serves as the starting address for multiple data transfers. With the IS68K block mode design, up to 16 words will be transferred in a single block transfer. Under block mode writes from the disk into memory (again four times as frequent as data reads from memory), one word transfer can occur approximately every 570ns for an effective transfer rate of just under 3.5 megabytes per second. With block mode read from memory to the disk, a word transfer can occur every 695ns for an effective transfer rate of 2.8 megabytes per second.

Mode	Dual-Ported Memory Maximum Transfer Rate (MBytes/sec)	
	Disk to Memory	Memory to Disk
Non-Block Mode	2.4*	2.2*
Block Mode (Rev 2.0)	3.5	2.8

Figure 1.9.4 LSI-11 Bus Memory DMA Bandwith

*In these two cases, maximum transfer rate is almost completely a function of LSI-11 bus protocol rather than the response time of the dual-ported memory.

SECTION 2

CONFIGURATION

2.1 Jumpers

2.1.1 Jumpers E1,E2,E3

These jumpers control the number of wait states when accesses are made to the on-board EPROMS.

E2-E3 - two wait states are inserted on 68000 accesses. EPROM access times of 260 ns or better are required fo 12 MHz operation, 320 ns or better for 10 MHz operation, 400 ns or better for 8 MHz operation.

E1-E3 - four wait states are inserted on 68000 accesses. EPROM access times of 430 ns or better are required for 12 MHz operation, 520 ns or better for 10 MHz operation, 650 ns or better for 8 MHz operation.

The factory configuration is E1-E3.

Jumper	12MHz	10MHz
E1-E3	430ns	520ns
E2-E3	260ns	320ns

Figure 2.1.1 EPROM Access Time (nano seconds)

2.1.2 Jumpers E4–E7

E4, E5, E6 and E7 control the LSI-11 bus timeout timing. There are three possible timeout timings that can be set by tieing jumper post E4, E5 or E6 to E7. Only one post of these three should be attached to E7. The board is shipped from the factory with the default of E5-E7 and it should be left in that configuration for most applications.

Jumper	12MHz	10MHz	8MHz
E6-E7	11.0	12.8	16
E5-E7	22.0	25.6	32
E4-E7	44.0	51.2	64

Figure 2.1.2 Bus Timeout Timing (microseconds)

2.1.3 Jumpers E31–36

Jumpers E31,E32,E33,E34,E35,E36 control the configuration of the EPROM sockets to allow the use of 2716,2732,2764,28128 type PROMS or EPROMS.

E32-E33 E35-E36 -2716 type EPROMS (2kx8)
E32-E33 E34-E36 -2732 type EPROMS (4kx8)
E32-E33 E34-E36 -2764 type EPROMS (8kx8)
E31-E33 E34-E36 -27128 type EPROMS (16kx8)

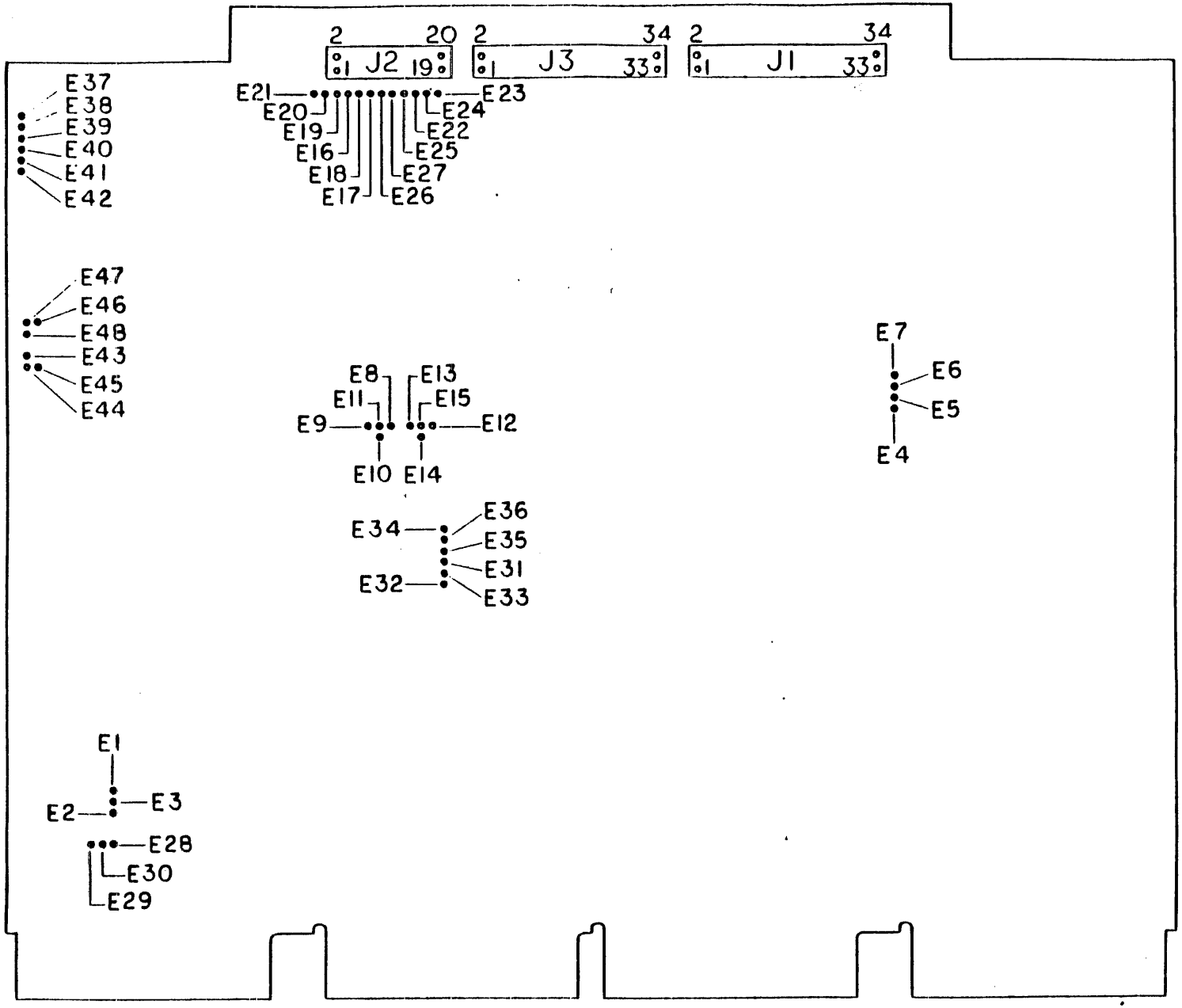


FIG.2.1 CONFIGURATION

2.1.4 Jumpers E8-E11,E16-E21,E43-E48

Jumpers E43,E44,E45,E46,E47,E48 configure serial port #0 to operate in RS423 (RS232C compatible) asynchronous mode or RS422 balanced synchronous mode. In synchronous mode, an external clock must be provided on the CLK and CLK RET lines. Jumpers E16-E21 are used in conjunction with the above jumpers to allow each receiver to have its own RETURN line in synchronous mode or to tie all receivers together to a common ground point in asynchronous mode.

For asynchronous operation on Channel 0,the following connections should be made:

E8-E10 - This ties the internally generated counter timer channel 0 baud rate output to the transmitter and receiver clock inputs on the SIO/DART chips.

E9-E11 This ties the CTS0 signal received on Pin 4 of the J2 connector into the SIO/DART chip.

E16-E17, E19-E21 This ties the return path for all serial channels to a common point. In addition, E49-E21 ties these return paths to Board Ground.

E43-E44, E47-E48 This provides the proper voltages to the Channel 0 26LS30 bus driver to allow it to operate in RS423 mode.

For synchronous operation on Channel 0, the following connections should be made:

E10-E11 This allows a baud clock generated externally to operate the SIO chip.

E16-E18, E20-E21 This configures each signal line to have its own RETURN. Since E49 (Board Ground) is not used in this configuration, these returns are not connected to ground for differential mode operation.

E44-E45, E46-E47 These provide the proper voltages to operate the 26LS30 chip in RS422 balanced synchronous mode.

For channel 1, a similar set of connections is made. For asynchronous operation:

E12-E14 - This ties the internally generated counter timer channel 1 baud rate output to the transmitter and receiver clock inputs for channel 1 on the SIO/DART chips.

E13-E15 This ties the CTS1 signal received on Pin 14 of the J2 connector into the SIO/DART chip.

E22-E23,E25-E26 This ties the return path for all serial receivers to a common point. In addition, these points must be tied to Board Ground. If channel 0 is wired up in asynchronous mode, this can be accomplished by tying E16-E17; otherwise, E16 should be tied directly to E49 (Board Ground).

E37-E38,E41-E42 This provides the proper voltages to the 26LS30 bus driver to allow it to operate in RS423 mode

For synchronous operation on Channel 1, the following connections should be made:

E14-E15 This allows a baud clock generated externally to operate the SIO/DART chip.

E22-E24,E25-E27 This configures each signal to have its own RETURN. These returns are not connected to ground for differential mode operation.

E38-E39, E40-E41 These provide the proper voltages to operate the Channel #1 26LS30 chip in RS422 balanced synchronous mode.

The factory default configuration is both channels wired for asynchronous operation.

ERRATA

On Rev 1.0 boards, there is no jumper post to tie the input RETURN lines to Board Ground. This is necessary for reliable operation of the asynchronous channels. This is accomplished on Rev 1.0 boards by tieing E17-E16 and E23 to the GND pin of chip J17 (pin 10). The board is factory configured in this manner.

2.1.5 Jumpers E28–E30 CTC Input

The CTC input clock can be programmed to be 1/4 or 1/8 the 68000 clock rate.

E28-E30 The CTC input clock is 1/4 the 68000 clock rate. This is the factory default position.

E29-E30 The CTC input clock is 1/8 the 68000 clock rate. The baud rates produced by the CTC are all exactly 1/2 the factory default rates. This jumper can effectively be used to support baud rates that are exactly 1/2 the four baud rates provided by MACSBUG. See Section 2.2.

2.2 Dipswitch

The single eight pin dipswitch at position B10 controls three independent functions.

Switch positions 1-4 control the transition between local bus memory and LSI-11 bus memory. These switches allow from 1 to 16 256k byte segments of physical memory to be placed on the local bus. When all four positions are closed, it indicates that only 256k bytes of memory are present on the local bus.

The following table indicates the switch settings for various amounts of memory present on the local bus:

Memory on Local Bus Including CPU Resident Memory	Switch 1	Switch 2	Switch 3	Switch 4
256k bytes	close	close	close	close
512k bytes	close	close	close	open
768K bytes	close	open	close	close
1024k bytes	close	open	close	open
1.25 meg	close	close	open	close
1.5 meg	close	close	open	open

1.75 meg	close	open	open	close
2.00 meg	close	open	open	open
2.25 meg	open	close	close	close
2.50 meg	open	close	close	open
2.75 meg	open	open	close	close
3.00 meg	open	open	close	open
3.25 meg	open	close	open	close
3.50 meg	open	close	open	open
3.75 meg	open	open	open	close
4.00 meg	open	open	open	open

ERRATA

On boards Rev A2 and earlier, setting all four switches open will put all 4.00 megabytes on the local bus. This has the problem that the upper 8 kbytes of the four megabyte address space (the I/O page of the LSI-11 bus) cannot be accessed by the IS68K. There are two solutions to this problem. For those users who must have the capability of putting a full 4 megabytes of memory on their system, a one wire change is available from Integrated Solutions to fix this problem. The other solution is to limit high speed memory to 3.75 megabytes or less. The problem has been fixed on boards REV A3 and later. On these boards, setting all the switches open will result in all four megabytes of memory minus the I/O page being available on the high speed bus. All I/O page references will go out on the LSI-11 bus.

Switch positions 5,6,7,8 are software readable switches that appear in the STATUS register at bit positions 3,2,1,0:

SW5 - bit position 3
SW6 - bit position 2
SW7 - bit position 1
SW8 - bit position 0

Under MACSBUG the switches have the following functions:

SW5 and SW6 set the console baud rate at power up according to the following table:

SW5 - closed, SW6 - closed - 9600 baud
SW5 - closed, SW6 - open - 2400 baud
SW5 - open, SW6 - closed - 1200 baud
SW5 - open, SW6 - open - 300 baud

If the jumper between E29-E30 is installed, the baud rates will be 4800 baud, 1200 baud, 600 baud, and 150 baud respectively.

SW7 and SW8 have no function under MACSBUG.

2.3 I/O Connector Pinout

Connector J2 contains the I/O connections for the two serial ports. The signals on the connector are different depending on whether one or both ports are configured for synchronous operation.

NOTE - Pin 1 is marked by a square pad on the IS68K board for all ribbon cable

connectors.

Port #0 - Asynchronous Configuration

J2-1 TXD0 out
J2-2 RXD0 in
J2-3 RTS0 out
J2-4 CTS0 in
J2-5 not used
J2-6 CH0 COMMON RET
J2-7 DCD0 in
J2-8 DTR0 out
J2-9 not used
J2-10 not used

Port #0 - RS422 Synchronous

J2-1 TXD0 out
J2-2 RXD0 in
J2-3 RTS0 out
J2-4 CLK0 in
J2-5 RTS0 RET out
J2-6 DCD0 RET in
J2-7 DCD0 in
J2-8 TXD0 RET out
J2-9 CLK0 RET in
J2-10 RXD0 RET in

Port #1 - Asynchronous Configuration

J2-11 TXD1 out
J2-12 RXD1 in
J2-13 RTS1 out
J2-14 CTS1 in
J2-15 not used
J2-16 CH1 COMMON RET
J2-17 DCD1 in
J2-18 DTR1 out
J2-19 not used
J2-20 not used

Port #1 - Synchronous

J2-11 TXD0 out
J2-12 RXD0 in
J2-13 RTS0 out
J2-14 CLK0 IN in
J2-15 RTS0 RET out
J2-16 DCD0 RET in
J2-17 DCD0 in
J2-18 TXD0 RET out
J2-19 CLK0 RET in
J2-20 RXD0 RET in

Both ports are factory configured for asynchronous operation. One common use of port 1 is for loading information to/from another computer system via a serial link. In this mode, a cable that reverses TXD and RXD must be used to make both computers look like they are talking to terminals. This is commonly called a NULL MODEM. See the section on configuration for instructions on making the serial ports operate in synchronous mode.

SECTION 3

OPERATION/PROGRAMMING

3.1 On-Board I/O Addresses

The 68000 logical address space from 400000H to FFFFFFH is dedicated to on-board I/O. On-board I/O is only accessible when the 68000 is in Supervisor mode; an attempt to access on-board I/O while in user mode will result in a bus-error trap. Note that this restriction is controlled by the PROTECTION PROM on a socket in position D10. Contact Integrated Solutions directly if more information on the PROTECTION PROM is required.

3.1.1 Segment Registers

There are 16 sets of of 64 segment registers; one set is assigned to each of the 16 contexts. Each segment register is eight bits long, is readable and writable, and is accessed over the least significant half of the bus (ie, the least significant byte of a word operation or an odd byte address).

The segment registers are loaded through the following I/O addressing mechanism:

Address	Contents
bits 23,22	01 - indicates that the segment register bank is selected
bits 21-16	0H-3FH - select one of the 64 registers in the context
bits 15-12	0H-FH - selects one of the 16 contexts
bits 11-1	not decoded/don't care
bit 0	the 8 bits of the segment register are located on the least significant half of the bus

NOTE

The segment registers are 8 bits long and are located on the least significant half of the bus. They can either be accessed via a byte access to the corresponding odd address or a word access to the corresponding even address. In the case of a word access, the upper eight bits of the bus are a don't care.

The following are examples of two typical segment register access addresses:

420001H - segment register #2 of context #0, the system context

536001H - segment register #19 of context #6

Each segment register contains 6 address translation bits, bits 5-0, which select one of

64 sets of 16 page registers and two protection bits 7 and 6. The protection bits are encoded for the following four access modes. Only mode 11, execute only, makes any distinction between supervisor and user mode in its functioning.

bit 7	bit 6	
0	0	no access, any attempt to access this segment (supervisor or user mode) will result in a bus error trap.
0	1	read only and execute access, any attempt to write this segment (supervisor or user mode) will result in a bus error trap.
1	0	read/write access, all forms of access are permitted in both supervisor and user modes.
1	1	all forms of access are permitted in supervisor mode but program references only are permitted in user mode.

The following examples give typical segment register entries:

80H - select page register set 0 (page registers 000H-00FH) for read/write access. The 68000 logical address bits 15 -12 select which page register is used to complete the translation.

7EH - select page register set 62 (3E0H-3EFH) for read only access. Again, logical address bits 15-12 select which page register in the set are used to complete the translation.

3.1.2 Page Registers

There are 1024 page registers which are accessed from 800000H- BFF000H in the 68000 logical address space. The page registers are each 12 bits long and are accessed by word references to even addresses according to the following mechanism:

Address	Contents
bits 23-22	10 - indicates that the page register bank is selected
bits 21-12	000H-3FFH - selects one of the 1024 page registers
bits 11-1	not decoded/don't care
bit 0	0 - page registers must be accessed via word accesses; byte accesses are not supported

The addresses of the first and second page registers are 800000H and 801000H respectively and the addresses of the last two page registers are BFE000H and BFF000H. Each

page register contains 12 bits which are used as follows:

Bit	Function
bits 11-2	the 10 address translation bits which select one of the 1024 physical pages (each page is 4 kbytes long) of memory that are to be accessed
bit 1	the page accessed bit
bit 0	the page written bit

To form a 22 bit physical address, the 10 tranlated address bits from the page register concatenated with the bottom 12 bits of logical address from the 68000:

selected page register contents bits 11-2	logical address bits 11-0	==> physical address bits 21-0

Four page register contents have special meaning:

3FCH and 3FDH - page register contents 3FCH and 3FDH always indicate the invalid page. Any attempt to access the invalid page either in supervisor mode or user mode will result in a bus error trap.

3FEH and 3FFH - page register contents 3FEH and 3FFH always refer to the LSI-11 bus I/O page. When these two pages are accessed, the LSI-11 bus BBS7 signal is asserted. In addition, the I/O page is always assumed to be located on the LSI-11 bus even if all 4 megabytes of physical memory are located on the local bus (see section 2.2). The upper 16 kilobytes of memory even if present in the system are not accessible by the 68000.

3.1.2.1 Page Accessed Bit (bit 1 of the page register)

The page accessed bit can be loaded as either 0 or 1 when the page register is loaded and it is changed to '1' whenever an access to that page is made. In normal operation, the page register accessed bits are written to zero as a process or part of a process is loaded into memory by the operating system. When any access to that page is made in the course of executing the process, the bit is converted to a '1'. The operating system, in the course of swapping out pages will examine the page accessed bits as part of deciding which pages to page out.

3.1.2.2 Page Written Bit (bit 0 of the page register)

The page written bit can be loaded as either 0 or 1 when the page register is loaded and it is changed to '1' whenever any change to that page is made. In normal operation, the page written bits are written to zero as a process or part of a process is loaded into memory by the operating system. When any change to that page is made in the course of executing the process, the bit is converted to a '1'. Pages which have not been changed can be paged out by simply loading the new page over them; the old page does not have to be written to the swapping device.

3.1.3 Context/Exception Register - C10001H -

The context/exception register is eight bits long is writable and readable and is located on the least significant half of the data bus. A special access method has been implemented to save hardware. The context/exception register MUST be written with a word refernce to location C10000H (the upper byte of data is discarded) and it must be read with a byte reference to location C10001H. Attempting to write the register with a byte write to location C10001H will result in no update. The context/exception register has two four bit fields with the following functions:

Context bits - bits 7-4 -

The context register allows the user to select which context, i.e., which set of segment registers is used when the processor is in user mode. When the processor is in supervisor mode, context 0 is always used regardless of the contents of the context register but the contents of the context register are preserved. In this manner, a user process can be interrupted, a switch to context 0 made automatically by the hardware and the interrupt serviced and a return to the previous user context made without touching the context register.

Exception bits - bits 3-0 -

The exception register has 4 read/write bits, three of which allow the user to determine the source of a bus error and the fourth of which serves as a parity enable bit to enable on-board parity checking.

bit 3 - bus timeout

A bus timeout can occur for one of several reasons:

- an attempt was made in either supervisor or user mode to access memory which is not existent. In this case, bits 2 and 1 will be reset.

- an attempt was made in user mode to access the on/board I/O space. In this case, the bus timeout bit will be set as well as bit 1, the protection violation bit.

- an attempt was made to access segments which have been set no access in the segment register, an attempt was made to write segments which are read only, or an attempt was made to access pages which have been marked invalid and the physical memory was located on the LSI-11 bus. If the physical memory is present on the local bus, only the protection violation bit is set; no bus timeout occurs.

bit 2 - invalid page

This bit is set when an effort is made to access a page that has been marked invalid in the page register. The invalid page bit will occur by itself when the physical page is located in local bus memory and in conjunction with the bus timeout bit when the page is located on the LSI-11 bus. In neither case is any memory write performed even if the memory is physically present in the system.

bit 1 - protection violation

A protection violation can occur either when an access is made to a protected segment, or in user mode, an attempt is made to access on-board I/O (a 68000 logical address above 3FFFFFH).

bit 0 - parity enable

When this bit is set to 1, parity is enabled for on-board/local bus memory. When cleared, parity is not checked, but parity is generated for all on-board memory writes. Parity errors will occur if the memory is not written to before being accessed if parity is enabled. The memory must initially be purged of all parity errors before parity is enabled.

NOTE

The four exception bits of the context/exception register are cleared to zero on system reset/power up.

3.1.4 Serial Ports/Counter-Timer - C20001-C21007

The IS-68K board has two serial ports and four counter/timer channels. Two of the counter/timer channels are dedicated to providing programmable baud rate for the two serial channels and the remaining two channels are concatenated to form a single programmable timer. This timer can be used to generate interrupts every 1/60th of a second if the IS-68K is used in a system that does not have a power supply that generates a BEVNT every 1/60th of a second or it can be used as a general purpose programmable timer in control applications.

The counter/timer chip is a Zilog CTC chip and is accessed at the following addresses:

C20001 - Channel 0
C20003 - Channel 1
C20005 - Channel 2
C20007 - Channel 3

The serial chip is either a Zilog DART chip or a Zilog SIO chip. The SIO provides SDLC and HDLC capabilities that are not present in the DART. Both the DART and the SIO support asynchronous and high speed synchronous communications. All IS-68Ks are shipped with DARTS unless the SIO is specified. There are four ports associated with the DART:

C21001 - DART Channel 0 DATA
C21003 - DART Channel 1 DATA
C21005 - DART Channel 0 CONTROL
C21007 - DART Channel 1 CONTROL

3.1.5 Status Register - C30000H

The status register is located at word location C30000h and has ten bits which are read only. The four latched error bits, bits 5 through 9, can be cleared by any write operation to the status register. The data written is irrelevant; all four bits are always cleared by the write operation. The bits in the status register have the following functions:

Bit	Function
9	PWR FAIL - The falling edge of the BPOK signal is latched. This indicates that a DEC compatible power supply has a minimum of 4

ms. of power remaining.

8 BUS PARITY ERROR - An error has been detected and latched on an LSI-11 bus read

7 ON-BOARD PARITY ERROR - A 68000 read access to on-board/local bus memory has resulted in a parity error.

6 HALT line - The falling edge of the HALT line has been detected and latched.

5 Inverted BPOKH - This signal goes low whenever BPOK is high and high whenever BPOKH is low. This signal could be used to resume processing after a power failure only when BPOKH has become active again.

4 Inverted BHALTL - This line is high when BHALTL is asserted on the LSI-11 bus and low when BHALTL is not asserted (processor enabled). This line could be used to emulate the DEC HALT function by inhibiting the processor from running when the HALT line is asserted.

3 Dipswitch bit 5

2 Dipswitch bit 6

1 Dipswitch bit 7

0 Dipswitch bit 8

Once an error is latched into the one of the upper four bits of the status register, it remains there until cleared by one of the following events:

- another error occurs in which case the new error will be latched into the appropriate bit of the STATUS register and a second level 7 interrupt will occur.

- a system RESET is executed externally by dropping BDCOKH on the LSI-11 bus or internally by executing the 68000 RESET instruction.

- any write operation to the STATUS register

APPENDIX A

INITIALIZING AND PROGRAMMING THE SERIAL PORTS

A.1 Initialising

The Zilog DART on the IS-68K board is a two channel, full duplex asynchronous/synchronous serial control chip, with full modem control. Both channels are completely independent, so that one may be operating in synchronous mode at one clock rate while the other is operating in asynchronous mode at another clock rate. Baud rates in asynchronous mode are programmable through the programming of two channels of the associated counter/timer chip. In synchronous mode, the clock must be externally generated for both transmission and reception. Baud rates up to 1 megabit/second can be supported in this mode by the hardware.

Before data can be transmitted and received via the DART, it must be initialized to define the operating characteristics of each of its two channels. Initialization of the DART consists of writing a string of control bytes to each of its two control registers which are located at C21005H and C21007H. The control register at C21005H corresponds to channel 0 and the control register at C21007H corresponds to channel 1. There are actually 5 write registers and 2 read registers associated with each channel of the DART. In addition, there is an additional read/write register associated with channel 1 which is not present in the channel 0 write register array. Of these 16 registers only WRITE REGISTER 0 and READ REGISTER 0 of the two channels are accessible directly. The remaining 12 registers must be accessed by writing a pointer to WRITE REGISTER 0 of the corresponding channel. The next read or write operation will then cause the *pointed–to* register to be accessed.

The following is a detailed description of the functions of the WRITE and READ REGISTER bits:

- **WRITE REGISTER 0**

WRITE REGISTER 0 has two functions; it has bits 5,4,3 which implement commands directly and bits 2,1,0 which serve as a pointer to allow accessing of other write and read registers. Bits 7 and 6 of write register 0 are not used in this application.

bits 5,4,3

000 - null code - no function directly implemented

010 - reset external status/interrupts

Performing this function has two effects. The piece of READ REGISTER 0 called external status is updated to agree with the current state of the lines entering the DART. This is important because once a change in one of the external status lines (including a BREAK condition on the data receive line) is detected by the DART, bits 7-3 are permanently latched in that condition until the next RESET EXTERNAL STATUS/INTERRUPTS is received.

The RESET EXTERNAL STATUS/INTERRUPTS, for example, must be issued repeatedly after a BREAK condition is detected by the DART in order to detect the end of the BREAK. If not issued, the BREAK bit in READ REGISTER 0 will stay latched

forever.

The second function of this command is to reset pending interrupts that are associated with this change in external status.

011 - channel reset

This command should always be the first command issued to each channel of the DART when the DART is initialized.

100 - enable interrupt on next received character

This command has meaning only when the DART is set up only to interrupt on the first received character. It resets the DART so that it will interrupt on the next received character and must be issued in the received character interrupt service routine when the DART is set up in this mode.

101 - reset transmitter interrupt pending

A transmitter interrupt will occur when the transmitter input buffer (a two deep SILO) goes empty. This command allows this interrupt to be reset without being serviced.

110 - error reset

This command should be issued when one of the three error conditions - parity error, data overrun error, or framing error (bits 4,5,6 in read register 1) is detected. These error bits have the same characteristic that the EXTERNAL STATUS bits in READ REGISTER 0 have, i.e., they are latched until cleared by this command.

111 - return from interrupt (channel 0 only)

This command is the last command issued before returning from the interrupt service routine. Its effect is to clear the DART interrupt that was serviced and setup the DART for subsequent interrupts.

The remaining combinations of bits 5,4,3 have no meaning in the DART.

bits 2,1,0

These bits are used to point to one of the other five write registers or one of the two directly inaccessible read registers of the DART. The next read or write operation to the DART will cause the pointed to register to be accessed. One of the commands in bits 5,4,3 is often simultaneously executed with the pointing operation. Often this command is the RESET EXTERNAL STATUS/ INTERRUPTS command.

- **WRITE REGISTER 1**

bits 7,6,5 - not used in this application

bits 4,3 -

00 - disable receiver interrupts

01 - interrupt on first received character only
10 - interrupt on all received characters (vector is affected by parity error)
11 - interrupt on all received characters (vector is not affected by parity error)

bit 2

Enable STATUS affects vector (this bit present in channel 1 only but affects both channels). If this bit is enabled, bits 3,2,1 of the vector programmed into the DART will be modified to a value that depends on the source of the interrupt.

bit 1 - enable transmitter interrupts

bit 0 - enable interrupts on changes in external conditions (CTS,DCD)

• **WRITE REGISTER 2** (Present in channel 1 only but affects both channels)

bits 7-0

These 8 bits form the base interrupt vector. Remember that the raw interrupt vector must be multiplied by four to give the entry in the 68000 interrupt vector table at the bottom of physical memory. In addition, if the STATUS AFFECTS VECTOR bit in WRITE REGISTER 1 is on, bits 3-1 of the interrupt vector will be modified depending on the source of the interrupt.

• **WRITE REGISTER 3**

bits 7,6

00 - 5 bits/char on receiving characters
01 - 7 bits/char on receiving characters
10 - 6 bits/char on receiving characters
11 - 8 bits/char on receiving characters

bit 5

Enables a feature called AUTOENABLE which automatically performs the following modem control functions:

DATA CARRIER DETECT (DCD on the serial I/O connector) must be asserted before reception of characters will take place and CLEAR TO SEND (CTS on the serial I/O connector) must be asserted before transmission of characters can begin. This bit should be programmed with caution because once set, the serial channel will hang if transmission or reception of characters is attempted without the appropriate signal line assertion.

bits 4-1 not used - must be 0s

bit 0 - receiver enable

This bit must be a '1' to enable reception of characters.

• **WRITE REGISTER 4**

bits 7,6-

00 - straight thru mode. The incoming clock is used directly as the baud rate. This clock can ONLY be used if the incoming data is synchronized to be valid during the rising edge of the clock, i.e. only for synchronous operation.

01 - divide by 16 clock mode. The incoming clock from the CTC (counter timer chip) is divided by 16 before being used as the baud rate.

10 - divide by 32 clock mode. The incoming clock from the CTC is divided by 32 before being used as the baud rate.

11 - divide by 64 clock mode. The incoming clock from the CTC is divided by 64 before being used as the baud rate.

- **WRITE REGISTER 5**

bit 7

DTR - This bit directly controls the DTR line on the serial interface. Setting the bit to 1 results in an active DATA TERMINAL READY line.

bits 6,5

00 - transmit with 5 bits/character
01 - transmit with 7 bits/character
10 - transmit with 6 bits/character
11 - transmit with 8 bits/character

bit 4

SEND BREAK - causes a continuous break to be sent on the TxD line of the corresponding channel until the bit is cleared.

bit 3

Tx ENABLE - This bit must be set to enable transmission of characters.

bit 2

Not used.

bit 1

RTS - This bit directly controls the RTS line on the serial interface. A '1' causes an active READY TO SEND signal.

- **READ REGISTER 0**

bits 7,5,3

These bits are the EXTERNAL STATUS bits. Bit 7 monitors a BREAK condition in the received data line while bits 5 and 3 monitor CTS and DCD respectively. Any

BREAK or change in the CTS or DCD lines will cause the corresponding bit to go to one and the other two bits latched in their zero state. This condition can be reset by issuing the RESET EXTERNAL STATUS/ INTERRUPTS command.

bits 6,4

Not used.

bit 2

Tx Buffer Empty - This bit goes to '1' whenever the transmit buffer can receive another character. Because, the transmit path is double-buffered, it does not necessarilly mean that the last character has been fully transmitted. See READ REGISTER 1 bit 0 for this function.

bit 1

Interrupt Pending - (implemented in channel 0 only) This bit indicates that some interrupt is currently pending in the DART.

bit 0

Rx Character Available - This bit goes to '1' whenever a character has been assembled in the receive SILO and is ready for pickup.

- **READ REGISTER 1**

bit 7

Not used.

bit 6,5,4

Error Bits - These are the three latched error bits that can occur on character reception. Bit 6 corresponds to Framing Error, bit 5 corresponds to Receiver Overrun and bit 4 corresponds to Receiver Parity Error. Once one of these bits is latched, the three bits can be reset by issuing an ERROR RESET command.

bits 3,2,1

Not used.

bit 0

All Sent - This bit goes to one whenever all the characters put in the transmitter SILO have been transmitted.

- **READ REGISTER 2** (Channel 1 only)

bits 7-0

Interrupt Vector - This register allows the programmed interrupt vector to be read. If STATUS AFFECTS VECTOR is set, this register will have the current modified vector in it.

A.2 Interrupt Vectors

The Zilog DART has a versatile interrupt scheme in which the DART chip itself will automatically vary the interrupt vector under the different interrupt conditions. This is a useful construct when the speed of interrupt response is critical. The alternative of having only one interrupt vector and having the interrupt service routine sort out the source of the interrupt causes a speed degradation. There are eight possible vectors which may be issued by the DART, four for each channel. A bit in WRITE REGISTER 1 when set allows the modified vectors to be generated. Unfortunately, the DART modifies vectors by simply changing bits 3-1 of the vector word. This places two constraints on the vectors that can be generated by the DART:

• The base vector must be xxxx000xb (binary) in order for eight unique vectors to be generated. The eight vectors generated are the permutations of bits 3-1 with bits 7-4 and 0 remaining unchanged.

• The vectors, when multiplied by four, should remain within the Motorola user interrupt space. This means that vectors from 40H to FFH are permitted. Note that there is no protection in the 68000 against vectors below 40H but they should be used with caution so that they do not overlap the system traps.

The modified vectors than can be generated are:

xxxx000x - channel 1 Transmit Buffer Empty - must have Tx Interrupt enabled.

xxxx001x - channel 1 External Status Transition - must have interrupt on changes in external conditions enabled.

xxxx010x - channel 1 Character Received - must have one form of receiver interrupts enabled

xxxx011x - channel 1 Special Receive Condition - the special receive condition interrupt always occurs if receiver interrupts have been enabled.

xxxx100x - Channel 0 Transmit Buffer Empty

xxxx101x - Channel 0 External Status Transition

xxxx110x - Channel 0 Character Received

xxxx111x - Channel 0 Special Receive Condition

A.3 Special Receive Condition

A special receive condition occurs if in the process of receiving a character one of the following erroneous conditions occurs:

- a parity error occurs if parity checking is enabled.
- a data overrun occurs. The three character SILO of the DART has overflowed.
- a framing error occurs.

Once a special receive condition has been detected, it will be latched in READ REG 1

bits 6,5,4 until the ERROR RESET command is issued.

A.4 Typical DART Initialisation Sequence

The DART is typically initialized in a short program loop which writes a string of bytes into the control register of each channel. The following code is the MACSBUG PROM resident code which initializes the DARTS.

```
        LEA     SIOTAB,A1       ;LOAD ADDRESS OF PARAMETER TABLE
        MOVL    #$C21005,A0     ;CHANNEL 0 CONTROL PORT
        MOV     #8,D0
SIO0LP: MOVB    (A1)+,A0
        DBRA    D0,SIO0LP

                                ;NOW PROGRAM CHANNEL 1
        ADDQ    #2,A0           ;POINT TO CHAN 1 CONTROL PORT
        LEA     SIOTAB,A1       ;POINT TO BEGINNING OF TABLE
        MOV     #8,D0           ;SET UP LOOP AGAIN
SIO1LP: MOVB    (A1)+,A0
        DBRA    D0,SIO1LP
SIOTAB: .BYTE   18,14,44,13,0C1,
                15,0EA,11,00,00
```

A.5 Typical Operation of the DART:

Once the DART is initialized, it can be programmed by writing or reading 8 bit parallel data directly from the data port for the appropriate channel. To transmit data to the DART, the user would poll the TRANSMIT BUFFER EMPTY bit of READ REGISTER 0 before stuffing a character into the DART.

```
TXLOOP: BTST.B  #2,$C21005      ;CHK IF XMIT BUFFER IS EMPTY
        BEQ     TXLOOP
        MOV.B   D0,$C21001      ;SEND DATA OUT
```

To read from channel 0 of the DART, the user would poll bit zero of READ REGISTER 0 until a character was received.

```
RXLOOP: BTST.B  #0,$C21005
        BEQ     RXLOOP
        MOVB    $C21001,D0      ;SAVE DATA IN D0
```

A more complete routine would look at the break bit in READ REGISTER 0

```
RXLOOP: MOVB    $C21005,D0
        BTST    #7,D0           ;CHK THE BREAK BIT
        BNE     BREAK           ;BRANCH IF BREAK SET
        BTST    #0,D0
        BEQ     RXLOOP
        MOVB    $C21001,D0      ;SAVE DATA IN D0
        .
```

```
            .
            .
BREAK:      MOVB    #$18,$C21005    ;ISSUE RESET EXTERNAL
                                    ;STATUS/INTERRUPTS
            BTSTB   #7,$C21005      ;IS BREAK STILL SET
            BNE     BREAK           ;YES KEEP LOOPING

;NOW ENTER THE BREAK SERVICE ROUTINE
```

APPENDIX B

MACSBUG COMMANDS

B.1 Overview

The MACSBUG option provides a resident firmware monitor for the 68000. This monitor supports a variety of commands for debugging and downloading progams, including commands to:

- display or change registers and memory
- control program execution through branching, breakpoints, and single and multiple stepping
- selectively display tracing information at breakpoints and while stepping
- communicate with a host computer
- use a limited form of relative addressing with certain of the above commands.

In addition, programming effort is eased somewhat by commands to perform arithmetic mode conversions and to allow symbolic access to numbers and memory locations.

These commands are discussed in detail in the following sections.

The MACSBUG command prompt is an asterisk (*). This prompt is shown in the examples in this Appendix; it is not to be entered by the user.

Input to MACSBUG is buffered. A control X (^X) cancels the line being entered; ^H, RUBOUT, and DEL delete the last character entered (but do not erase it from the screen - to redraw the line enter ^D). A ^W will stop output to the console; entering any other character will start it again. The BREAK key will stop almost anything MACSBUG is doing (but will not kill a user program running under MACSBUG).

B.2 Displaying or Changing Registers and Memory

B.2.1 Displaying Registers

The hex contents of any of the 68000's registers can be displayed by entering the name of the register in response to the MACSBUG prompt. For example,

```
*A3<cr>
```

might produce

```
A3=0000146A.
```

The names of the registers are as follows:

D0, ..., D7	data registers
A0, ..., A7	address registers
PC	program counter
SR	status register
SS	supervisor stack pointer
US	user stack pointer.

Recall that A7 is SS in supervisor mode, US in user mode.

All of the address or data registers can be displayed by entering A or D, respectively. For example,

```
*D<cr>
D0=00000000 D1=FFFFFFF2 D2=1479630A D3=00000001
D4=FFFFFFFF D5=00000000 D6=00000000 D7=A0369741
```

Registers can also be displayed as part of a trace display. (See B.4.)

B.2.2 Changing Registers

Data can be entered into registers in either hex or ASCII. Hex is the default; to enter ASCII data, enclose it in single quotes. The simplest way to change a register is to enter the register name followed by the new data; thus

```
    *SR 0<cr>
```

will clear the status register, while

```
    *D3 'Fred'<cr>
```

will put 46726564 (ASCII 'Fred') in data register 3. An attempt to put more than four characters of ASCII data in a register will result in a SYNTAX ERROR message, while an attempt to enter more than eight hex digits will result in simply ERROR. (MACBUG's error messages are sometimes slightly obscure.) Data items shorter then eight hex digits or four ASCII characters will be padded on the left with zeroes.

To display the contents of a register and optionally change it, enter the name of the register followed by a colon. For example,

```
    *A4:<cr>
```

might produce

```
    A4=00001A47 ?
```

To change the value in A4, enter the new value after the question mark; e.g.,

```
    A4=00001A47 ? 1A43
```

If the old value is satisfactory, simply enter a carriage return.

It is also possible to cycle through the address or data registers, examining and optionally changing them one at a time, by entering A: or D: and responding to the ensuing question marks with new data or carriage returns.

B.2.3 Displaying Memory

Memory is displayed in chunks of sixteen (hex 10) bytes. Hex and ASCII representations are shown side-by-side. To display the contents of an address, use the DM command followed by the hex address. Thus,

```
    *DM 3000
```

might produce

```
003000  58 50 71 00 00 FF 00 FF 00 00 00 00 00 FF 00 FF XPq.....
```

The contents of the indicated address will always be the first data displayed, so with the data above

```
    *DM 3001
```

might give

```
003001  50 71 00 00 FF 00 FF 00 00 00 00 00 FF 00 FF 00 Pq.....
```

To display more than sixteen bytes starting at a given address, enter

*DM start n

where start is the hex address from which to start displaying data and n is either the hex number of bytes desired or the hex location at which to stop the display. This is, of course, ambiguous. The ambiguity is resolved by the convention that if n is greater than start it is the ending address, while if n is less than or equal to start it is an item count. In either event, the number of bytes displayed will be a multiple of sixteen. Thus

*DM 100 100
*DM 100 1FF
*DM 100 1F0
*DM 100 F1

will all display the 256 (hex 100) locations 100 to 1FF.

It is possible to send display output to port 2 on the board rather than port 1 by using the DM2 command in place of DM in the above.

B.2.4 Changing Memory

B.2.4.1 The SM Command

To set the contents of memory, use the SM command followed by an address and one or more data items (separated by blanks). Data items can be hex or ASCII; ASCII data must be enclosed in single quotes.

The SM command is strictly byte-oriented. This requires caution when entering data, particularly hex data.

Hex data can be from one to eight digits, plus as many leading zeroes as desired. (Although it is possible, as shown in the example below, to enter numbers larger than eight digits by the addition of leading zeroes, such techniques might be invalidated by future versions of the firmware.) Leading zeroes are significant; the entire number, including leading zeroes, is right justified in the smallest number of bytes possible, with an extra leading zero if necessary to fill out the leftmost byte. There is an exception to this: if the data item is eight hex digits with the most significant an 8 or greater (i.e., if the 32-bit hex number is negative) all nibbles that would have been filled by leading zeroes are instead filled with Fs. (It is simplest to think of the numbers as being sign-extended, but this may be confusing: leading zeroes are still significant in determining the number of bytes to be changed.) Perhaps an example will clarify things:

*SM 2000 1 002 00012345678 0FFFFFFFF 4567
*DM 2000
002000 01 00 02 00 00 12 34 56 78 FF FF FF FF FF 45 67

Here the 1 occupies one byte, the 2 occupies two bytes (one extra byte for its second leading zero and an extra leading zero to pad out the extra byte), the FFFFFFFF occupies five bytes (one extra nibble for its leading zero and a second nibble to pad, both nibbles filled with Fs), and the 4567 occupies two bytes.

ASCII data can be from one to 256 characters in length. (At present slightly longer strings are handled correctly, but there is no guarantee that they will be accepted by future versions of the firmware.) An attempt to enter longer strings can result in either an error message or in silent truncation of the data.

ASCII and hex data can be mixed in one SM command. For example,

```
    *SM 2000 'cat' 416 'dog' 417 'turkey'
    *DM 2000
002000  63 61 74 04 16 64 6F 67 04 17 74 75 72 6B 65 79
cat..dog..turkey
```

When putting more than one data item on a line it is wise to restrict the total length of the data (hex digits plus characters) to no more than 256.

B.2.4.2 The OP Command

Memory can also be changed by using the OP command to open memory at a given address. This command enters a subcommand mode in which MACSBUG displays an address and its contents, then waits for the user to reply. Either hex or ASCII data can then be entered (or no data can be entered and the location will be left unchanged), followed by one of the following subcommands:

(CR)	go to the next location
^	go to the previous location
=	stay at the same location
.	exit subcommand mode (end OP)

One disadvantage of this method is that it is strictly one byte at a time. An attempt to enter more than one character or two hex digits results in all but the rightmost character or two digits being discarded. This may be seen in the following example:

```
*OP 2000

MACSBUG prompts    User responds
002000  00 ?       21
002001  00 ?       716
002002  00 ?       'J'=
002002  4A ?       'K'^
002001  16 ?       ^
002000  21 ?       .

Now

    *DM 2000
yields
    002000  21 16 4B ....
```

B.2.5 Accessing Memory Through Windows

In addition to the methods of sections B.2.3 and B.2.4, memory can be displayed and changed through "windows." A window is simply an effective address that has been given a special name. Windows are named W0 through W7; their corresponding memory locations are M0 through M7. The following effective addressing modes are available for windows:

Example	Mode
20F4	absolute
(A5)	register indirect

2F(A5)	register indirect with displacement
2F(A5,D4)	indexed register indirect with displacement
(*)	PC relative
2F(*)	PC relative with displacement
2F(*,A5)	PC relative with index and displacement

The term displacement is used rather than the usual offset because "offset" has special meaning in MACSBUG (see B.6).

Windows are defined by the W command:

```
*Wn.l EA
```

where n is the window number (from 0 to 7), l is the length of the window in bytes (from one to four - a length of zero "closes" or deactivates the window), and EA is notation for an effective address. It is assumed that the reader is familiar with the addressing modes of the 68000; the syntax of the available effective addressing modes should be clear from the example above. If W followed by a window number is entered alone, MACSBUG will respond with the effective address of that window. For instance,

```
*W3.4 10(A3,D2)
*A3 2000
*D2 20
*W3
```

would result in

```
W3.4  10(A3,D2)=2030
```

This is a laborious way to do simple addition, but much can happen in practice after the definition of W3; the bare W3 might be a useful reminder.

Once windows have been opened, the associated effective addresses can be treated as if they were registers. In the above example,

```
*M3 1A47
```

will put 1A47 into memory location 2030; the result can be checked by entering

```
*M3
```

to which the computer should respond

```
M3=00001A47
```

The result could also be checked by a

```
*DM 2030
```

as long as the values in A3 and D2 have not changed. Observe that the effective address M3 changes as the values in A3 and D2 change. For instance, if we now enter

```
*D3 40

*W3
```

the effective address becomes 2050, not 2030.

The window lengths defined by *Wl.n EA are enforced; an error will result from an attempt to put more data into memory than can fit in the specified length.

Data in windows can also be printed as part of a trace display. (See B.4.)

B.3 Controlling Program Execution

B.3.1 Starting Execution

Execution of a program can be started by using the G command. This command has

three forms -

*G	start execution at address in PC
*G address	start execution at this address
*G TILL address	set a temporary breakpoint at this address, then begin execution at address in PC

In the last of these, the temporary breakpoint is cleared as soon as any breakpoint is encountered; otherwise, it behaves as any other breakpoint.

B.3.2 Stopping Execution – Breakpoints

Once a user program begins executing under MACSBUG, it will continue until it completes execution (it may stop as the result of an interrupt, but it must process the interrupt itself - MACSBUG will not halt a user program on receiving a BREAK), or encounters a breakpoint.

MACSBUG allows a user to set up to eight breakpoints, each of which may have an associated count. To set a breakpoint at an address, enter

*BR address

to remove it, enter

*BR -address

For a list of all active breakpoints, simply enter

*BR

Breakpoints in MACSBUG may have an associated count. A breakpoint set with a count of n will not stop execution until the nth time it is hit. When a breakpoint with a count greater than one is encountered, the program does not halt; instead, the count is decremented and certain trace information is displayed (see B.4 - the trace display is also printed when execution is stopped by a breakpoint). To enter a breakpoint with a count, simply follow its address with a colon and the count in the BR command; for example,

*BR 2000:4

There is no practical difference between a breakpoint with a count of one and one with a count of zero (no count) except that the former will show its count in response to a *BR.

All breakpoints in a program can be removed by entering

*BR CLEAR

B.3.3 Stepping Through Execution – the T Command

MACSBUG provides convenient features for stepping and tracing through program execution. To step through the next instruction (i.e., to execute the instruction pointed to by PC), simply enter

*T

The instruction will be executed and the trace display (B.4) will be printed.

To execute the next n instructions, printing the trace display after each, enter

*T n

To trace until a certain address is reached, enter

*T TILL address

This will step through one instruction at a time, printing the trace display after each, until either the specified address or some breakpoint is reached.

All of these commands put MACSBUG in trace mode - the normal * prompt is replaced by :* and simply hitting (CR) will cause the next instruction to be traced. (Any other MACSBUG command can also be entered; anything but a (CR) or one of the trace commands will automatically end trace mode.)

B.4 The Trace Display

The user can exercise control over the trace display (the information displayed while tracing and when breakpoints are hit) by using the TD command. The basic form of this command is

*TD reg.format

where reg is one of the following:

- registers D0,...,D7, A0,...,A7, PC, SR, US, and SS
- register classes A and D
- window addresses W0,...,W7
- window contents M0,...,M7

and format is one of the following:

- 0 to remove the item from the display
- 1, 2, 3, or 4 to display one to four bytes in hex (with leading zeroes if necessary)
- Z or D to display four bytes (a long word) in hex or decimal, respectively - signed, with no leading zeroes

Observe that reg.format pairs can be concatenated on the command line. For example, one might enter

*TD D1.1 A5.3 PC.3 M1.D M2.0

The A and D register classes are an exception to the formatting rules. A.1 and D.1 simply put all the address and data registers, respectively, in the display in four byte unsigned hex format. A.2, A.3, A.4, A.Z, A.D, A.R, and A.S have the same effect as A.1; D.2, D.3, D.4, D.Z, D.D, D.R, and D.S act the same as D.1. (See below for R and S formats.)

If a window has been defined to be less than four bytes wide, printing it with Z or D format will print only as many bytes as the width of the window; in this case it will be treated as unsigned. For example,

```
*W1.4 2000
*M1 FFFFFFFF
*TD M1.Z
*TD
```

will print

```
M1=-1
```

(TD by itself prints the trace display), while

```
*W1.3 2000
*TD
```

would now print

```
M1=FFFFFF
```

It is possible to separate blocks in successive trace displays by defining a line separator. To use, say, '!' as the line separator, enter

*TD L.!

This will cause a row of exclamation points to be printed after each trace display, thus making it somewhat easier to read the output. To remove the line separator enter

*TD L.0

To clear the trace display (remove everything, including the line separator) enter

```
*TD CL
```

If no TD commands have been entered to MACSBUG, the default trace display format is

```
*TD PC.3 SR.2 US.4 SS.4 D.1 A.1 L.-
```

The display can be reset to this default by entering

```
*TD AL
```

There are two other options for the format parameter - R and S. These will be discussed here although both refer to material from later sections of this Appendix.

If the user has defined an offset (see B.6), s/he may wish to make part or all of the display relative to this offset. This is accomplished by using R as the format character. Thus

```
*TD A1.R
```

will cause the offset to be subtracted from A1 and the result printed in Z format followed by the letter R. For example,

```
*OF 2000
*A1 1FFF
*TD
```

would now print

```
A1=-1R
```

Finally, if the user has defined symbols with the SY command (see B.7), the S format may be useful. For instance, if we enter

```
*TD A3.S
```

then each time the display is to be printed the value in A3 will be compared with the (four byte) value in the symbol table. If the value in A3 is found in the table, the corresponding name from the symbol table will be printed (eight characters). If not, the value itself will be printed as eight hex digits.

B.5 Communicating with a Host Computer

B.5.1 Console/Host Communication – Transparent Mode

It is often necessary for the console to communicate directly with a host computer (for example, to do compilations and assemblies). This can be done by putting MACSBUG in transparent mode. In transparent mode commands and data entered on the console go directly to the host computer, while the host's transmissions go directly to the console. To enter transparent mode, enter

```
*P2 endchar
```

where endchar is a character which, when entered on the console, will end transparent mode. Obviously, the endchar should be one that is not needed in communication with the host. If no endchar is entered, the default is ^A (control A).

If no reply from the host is needed, entering * in response to MACSBUG's * prompt will send the remainder of the command line to the host (port 2); thus

```
**cc -O file.c &
```

might be used to start a C compilation on a host Unix system without leaving MACSBUG.

B.5.2 Downloading and Verifying

The commands in B.5.1 provide a means of direct communication between the console and the host; those in this section provide for communication between the host and

the 68000 (needed for downloading programs).

The RE command is used to download 'S' records from the host (port 2). The general form of the command is

*RE;=text

where the text after the = is sent to the host to encourage it to begin transmission. For example,

*RE;=LOAD FILE.DATA

where LOAD is a program that converts FILE.DATA into 'S' records and controls the host's end of the download.

If an illegal character or bad checksum is encountered during the download, an error message will be issued. (As there is no handshaking or other control of the download by the 68000, it is likely that some subsequent records will be lost while the error message is being printed.) To ignore checksums, use

*RE;-C =text

To display data as it is being read, use

*RE;-X =text

To do both,

*RE;-CX =text

Although the 'S' records contain address information, the data can be loaded at a different address by using a global offset. (Observe that this will not generally accomplish a relocation of a program.)

Downloaded data can be checked using the VE command:

*VE;=text

can be used to read the same 'S' records again and print any that are different.

For more information on downloading and 'S' records, refer to Appendix C.

B.6 Using Relative Addressing – the Offset

MACSBUG allows the user to do a limited form of relative addressing by defining a global offset. This hex value will automatically be added to the values in the BR, G, SM, and DM commands. In addition, the offset is added to the address of data downloaded by the RE command and can be used with the TD command as discussed in B.4.

To set the offset, enter

*OF n

where n is the hex value of the offset. To clear the offset, enter

*OF 0

To display its current value, enter simply

*OF

An alternate offset can be used with any command that uses the offset by following the data with a comma and the alternate offset. For example,

*OF 2000

*BR 10,3000

will set the breakpoint at 3010, not 2010. To use no offset at all with one of these commands, use the comma with no alternate offset. In the above,

*BR 10,

would set the breakpoint at absolute 10.

Commands that do not use the offset (such as SY) can be forced to do so by following the data with an R. For example,

```
*OF 1000

*SY ADD 1000R
```

will define ADD to have the value 2000 (see B.8).

B.7 Performing Arithmetic Mode Conversions

MACSBUG allows easy conversion from decimal to hex and vice versa. To convert from decimal to hex, enter

```
*CV n
```

To convert from hex to decimal, enter

```
*CV $n
```

To convert the value of a symbol from hex to decimal, enter

```
*CV name
```

where name is the name of the symbol in question.

To calculate an offset or displacement, enter

```
*CV value,offset
```

This adds value and offset together and prints the result in hex and decimal. This command is tricky; value and offset are both assumed to be decimal unless preceded by dollar signs. To add the global offset to a value, enter

```
*CV value,0R
```

or

```
*CV valueR
```

Here are some examples of the CV command.

```
*CV $10
$10=&16
*CV 16
$10=&16
*CV 10,10
$14=&20
*CV 10,$10
$1A=&26
*OF 1000
*CV 10,10R
$1014=&4116
```

In the last example, both tens are decimal, while the global offset 1000 is hex.

B.8 Defining and Using Symbols

Frequently used or significant addresses or other numbers can be defined symbolically with the SY command. To define a symbol, enter

```
*SY name value
```

where name is from one to eight characters chosen from A,...,Z, 0,...,9, period, and dollar sign. The name must begin with a letter or a period. To undefine a symbol, enter

```
*SY -name
```

To print the value of a symbol, enter

```
*SY name
```

(Observe that if name is an address this prints the address, not its contents.)

To print the first symbol (alphabetically) with a given value, enter

```
*SY value
```

To print the entire symbol table, sorted alphabetically, enter

```
*SY
```

Symbols can be used most places ordinary constants can be used. The global offset is not used in defining or printing symbols. For example,

```
*OF 1000
*SY MIX 1009
*SY MIX
MIX    =1009
*BR MIX
*BR
BRKPTS= 2009
```

Here the offset is not used with SY, but is used with BR.

MACSBUG will not accept TILL, ALL, or CLEAR as symbols.

B.9 Summary

Command	Meaning
^X	cancel line being entered
^H, RUBOUT, DEL	delete last character entered
^D	redraw command line
^W	halt output until another character entered
BREAK	stop everything (except user program)
^A	exit transparent mode (default endchar)
reg	display register
A or D	display all A or D registers
reg data	enter hex data in register
reg 'data'	enter ASCII data in register
reg:	display old value; request new one
A: or D:	cycle through A or D registers, displaying and requesting new values
DM start n	display memory in hex and ASCII - from start to n if n > start, otherwise n bytes
SM address data	enter hex data into memory (bytewise)
SM address 'data'	enter ASCII data into memory
OP address	open memory at address for changing
Wn.l EA	define window
Mn	look through window n into memory
G	start execution at address in PC
G address	start execution at given address
G TILL address	set temporary breakpoint at address, start execution at address in PC
BR address	set breakpoint at address
BR -address	remove breakpoint at address
BR	list all active breakpoints
BR address:count	set a breakpoint with a count
BR CLEAR	clear all breakpoints
T	single step and print trace display
T n	step n instructions, trace display after each
T TILL address	like G TILL address, but print trace display after each instruction
:*(CR)	step one instruction in trace mode
TD reg.format	put (format) register, register class, or window in trace display
TD L.char	define char as trace display line separator
TD CL	clear trace display
TD AL	put all registers in trace display (default)
P2 endchar	enter transparent mode; get out with endchar
*text	send text to host
RE;=text	read 'S' records
VE;=text	verify 'S' records
OF address	define address as global offset
CV n	convert n from decimal to hex
CV $n	convert n from hex to decimal
CV m,n	add m and n; print result in hex and decimal
SY name value	define symbol
SY name	print symbol
SY	print all symbols

APPENDIX C

Downloading and 'S' Record Format

Object data to be downloaded to the IS-68K using MACSBUG's RE command must be in 'S' record format. The 'S' record format was devised by Motorola for the purpose of encoding data files in printable form for transportation between computer systems. 'S' records are character strings made up of five fields which define:

- the type of 'S' record
- the record length
- the load address
- a data field
- a checksum

Each 'S' record begins with the letter 'S' followed by the number 0,1,2,8,9. Each of these numbers indicates a specific type of 'S' record. The types of 'S' records are as follows:

S0	The header record for a block of 'S' records. The header record contains a valid record length field, has a 16 bit starting address of all zeroes, a code/data field containing any kind of information desired and a valid checksum. The S0 record is ignored by the MACSBUG 'S' record loader and is optional.
S1	The S1 record is a data record which contains a valid record length field, a valid 16 bit starting address, a code/data field containing data to be loaded at the starting address and a valid checksum.
S2	The S2 record is a data record which contains a valid record length field, a valid 24 bit starting address, a code/data field containing data to be loaded at the starting address and a valid checksum.
S8	The S8 record is a terminating record which contains a valid record count, an optional 24 bit address of the location to which MACSBUG is to transfer control after loading the 'S' record and a checksum. There is no data field. The S8 record is optional and causes MACSBUG to return control to the keyboard monitor if there is no start address.
S9	The S9 record is a terminating record which contains a valid record dount, an optional 16 bit address of the location to which MACSBUG is to transfer control after loading the 'S' record and a checksum. There is no data field. The S9 record is optional and causes MACSBUG to return control to the keyboard monitor if there is no start address.

characters 1-2	S0 if this is the first 'S' record S1 if address field is two bytes S2 if address field is three bytes S8,S9 if this is the last 'S' record
characters 3-4	number of character pairs remaining in this 'S' record (in hex)
characters 5-8 (or 5-10)	two- or three-byte (depending on second character) hex address at which to load following data
rest of characters (except last two)	data
last two characters	checksum of address and data fields

NOTE

The checksum is the least significant byte of the one's complement of the sum of the *values* represented by the character pairs in the record length, address and data fields.

Shown below is a typical 'S' record module consisting of an S0 record, four S1 records and an S9 terminating record. Remember that the S0 and S9 records are optional.

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The first three S1 records contain 13H (19 decimal) character pairs and the final S1 record contains seven character pairs. Note that the starting addresses for the four S1 records are 0000H, 0010H, 0020H, and 0030H respectively.

For more information on downloading and verifying with the RE and VE commands, see Appendix B.

APPENDIX D

SETTING UP THE MEMORY MANAGEMENT UNIT

On the IS-68K, all memory references both in supervisor and user modes go through the memory managment unit. Therefore, before RAM memory can be accessed, the memory management unit must be initialized to permit address translations to occur in a valid manner. The following code sets up the SUPERVISOR (context #0) memory management unit up in transparent mode, ie, 68000 logical addresses are translated to the same physical addresses.

```
MMUSET:  MOVE.W   #$80,D0       ;GIVE ALL SEGMENTS READ/WRITE
                                ;ACCESS
         MOVE.L   $400001,A0    ;THE ADD OF FIRST SYSTEM
                                ;MODE SEGMENT
         MOVE.W   #$3F,D1       ;A LOOP COUNT, THERE ARE 64
                                ;SYSTEM MODE SEGMENTS
SEGLP:   MOVE.B   D0,(A0)       ;WRITE INTO SEGMENT REG
         ADDQ.W   #1,D0         ;POINT TO NEXT PAGE REG SET
         ADD.L    #$10000,A0    ;POINT TO NEXT SEGMENT REG
         DBRA     D1,SEGLP
         MOVE.W   #$3FF,D1       ;THERE ARE 1024 PAGE REGS
         CLR.W    D0             ;POINT TO THE BOTTOM OF
                                 ;PHYSCIAL MEMORY. NOTE THAT
                                 ;THE BOTTOM TWO BITS OF PAGE
                                 ;REGISTER ARE PAGE WRITTEN AND
                                 ;PAGE ACCESSED
         MOVE.L   $800000,A0     ;FIRST PAGE REG POINTED TO
PAGLP:   MOVE.W   D0,(A0)        ;WRITE THE PAGE REG
         ADD.W    #4,D0          ;REMEMBER BOTTOM TWO BITS
         ADD.W    #$1000,A0      ;POINTS TO NEXT PAGE REG
         DBRA     D1,PAGLP       ;PROGRAM ALL 1024 PAGE REGS
```

Using the Z80® SIO In Asynchronous Communications

Zilog

Application Note

July 1980

SECTION 1

Introduction.

The Z80 Serial Input/Output (SIO) controller is designed for use in a wide variety of serial-to-parallel input and parallel-to-serial output applications. In this application note, only asynchronous applications are considered. The emphasis is almost completely on software implementation, with only modest reference to hardware considerations.

While reference is made only to the Z80 SIO, the entire text also applies to the Z80 DART, which is functionally identical to the Z80 SIO in asynchronous applications.

Protocol

Communication, either on an external data link or to a local peripheral, occurs in one of two basic formats: synchronous or asynchronous. In synchronous communication, a message is sent as a continuous string of characters where the string is preceded and terminated by control characters; the preceding control characters are used by the receiving device to synchronize its clock with the transmitter's clock. In asynchronous communication, which is described in this application note, there is no attempt at synchronizing the clocks on the transmitting and receiving devices. Instead, each fixed-length character (rather than character string) is preceded and terminated by "framing bits" that identify the beginning and end of the character. The time between bits within a character is approximately constant, since the clocks or "baud rates" in the transmitter and receiver are selected to be the same, but the time between characters can vary.

Thus, in asynchronous communication, each character to be transmitted is preceded by a "start" framing bit and followed by one or more "stop" framing bits. A start bit is a logical 0 and a stop bit is a logical 1. The receiver will look for a start bit, assemble the character up to the number of bits the SIO has been programmed for, and then expect to find a stop bit. The time between the start and stop bits is approximately constant, but the time between characters can vary. When one character ends, the receiving device will wait idly for the start of the next character while the transmitter continues to send stop or "marking" bits (both the stop bits and the marking bits are logical 1). Figure 1 illustrates this. A very common application of asynchronous communication is with keyboard devices, where the time between the operator's keystrokes can vary considerably.

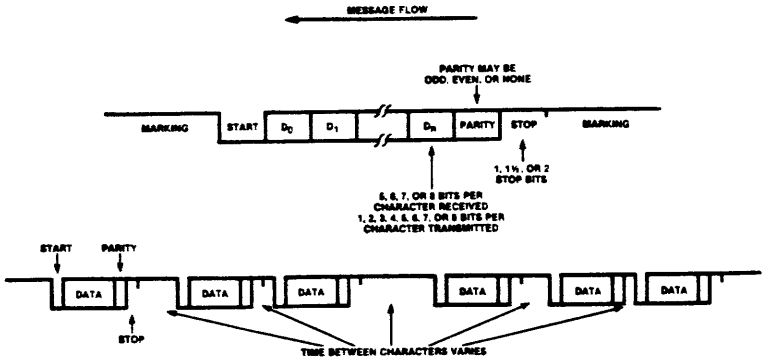


Figure 1. Asynchronous Data Format

Protocol (Continued)

If the transmitter's clock is slightly faster than the receiver's clock, the transmitter can be programmed to send additional stop bits, which will allow the receiver to catch up. If the receiver runs slightly faster than the transmitter, then the receiver will see somewhat larger gaps between characters than the transmitter does, but the characters will normally still be received properly. This tolerance of minor frequency deviations is an important advantage of using asynchronous I/O. Note however that errors, called "framing errors," can still occur if the transmitter and receiver differ substantially in speed, since data bits may then be erroneously treated as start or stop bits.

Modes

The SIO may be used in one of three modes: Polled, Interrupt, or Block Transfer, depending on the capabilities of the CPU. In Polled mode the CPU reads a status register in the SIO periodically to determine if a data character has been received or is ready for transmission. When the SIO is ready, the CPU handles the transfer within its main program.

In Interrupt mode, which is far more common, the SIO informs the CPU via an interrupt signal that a single-character transfer is required. To accomplish this, the CPU must be able to check for the presence of interrupt signals (or "interrupt requests") at the end of most instruction cycles. When the CPU detects an interrupt it branches to an interrupt service routine which handles the single-character transfer. The beginning memory address of this interrupt service routine can be derived, in part, from an "interrupt vector" (8-bit byte) supplied by the SIO during the interrupt acknowledge cycle.

In Block Transfer mode, the SIO is used in conjunction with a DMA (direct memory access) controller or with the Z80 or Z8000 CPU block transfer instructions for very fast transfers. The SIO interrupts the CPU or DMA only when the first character of a message becomes available, and thereafter the SIO uses only its Wait/Ready output pin to signal its readiness for subsequent character transfers. Due to the faster transfer speeds achievable, Block Transfer mode is most commonly used in synchronous communication and only rarely in asynchronous formats. It is therefore not treated with specific examples in this application note.

Since Polled mode requires CPU overhead regardless of whether or not an I/O device desires attention, Interrupt mode is usually the preferred alternative when it is supported by the CPU. Note that the choice of Polled or Interrupt mode is independent of the choice of synchronous or asynchronous I/O. This latter choice is usually determined by the type of device to which the system is communicating.

SIO Configurations

The SIO comes in four different 40-pin configurations: SIO/0, SIO/1, SIO/2, and SIO/9. The first three of these support two independent full-duplex channels, each with separate control and status registers used by the CPU to write control bytes and read status bytes. The SIO/9 differs from the first three versions in that it supports only one full-duplex channel. The product specifications for these versions explain this in full.

There are 41 different signals needed for complete two-channel implementation in the SIO/0, SIO/1, and SIO/2, but only 40 pins are available. Therefore, the versions differ by either omitting one signal or bonding two signals together. The dual-channel asynchronous-only Z80 DART has the same pin configuration as the SIO/0.

SIO-CPU Hardware Interfacing

The serial-to-parallel and parallel-to-serial conversions required for serial I/O are performed automatically by the SIO. The device is connected to a CPU by an 8-bit bidirectional data path, plus interrupt and I/O control signals.

The SIO was designed to interface easily to a Z80 CPU, as shown in Figure 2. Other microprocessors require a small amount of external logic to generate the necessary interface signals.

The SIO provides a sophisticated vectored-interrupt facility to signal events that require CPU intervention. The interrupt structure is based on the Z80 peripheral daisy chain. Non-Z80 microprocessors that are unable to utilize external vectored interrupts require some additional external logic to utilize efficiently this interrupt facility. Some non-Z80 system designs do not utilize the vectored interrupt structure of the SIO at all. Instead, these require the CPU to poll the SIO's status through the data bus or to use non-vectored SIO interrupts.

Microprocessors such as the 8080 and 6800 need some signal translation logic to generate SIO read/write and clock timing. CPU signals which synchronize a peripheral device read or write operation are gated to form the proper I/O signals for the SIO. The SIO is selected by some processor-dependent function of the address bus in a memory or I/O addressing space.

Reference Material

In the next section we begin with a discussion of features common to all forms of asynchronous I/O. This is followed by discussions of polled asynchronous I/O and interrupt asynchronous I/O. Next is a series of frequently asked questions about the SIO when used in asynchronous applications. Finally, an example of a simple interrupt-driven asynchronous application is given and discussed in detail. For a complete understanding of the material covered, the following publications are needed:

- *Z80 SIO Product Specification or Z80 DART Product Specification*
- *Z80 SIO Technical Manual*
- *Z80 Family Program Interrupt Structure*
- *Z80 CPU Technical Manual*
- *Z80 Assembly Language Programming Manual*

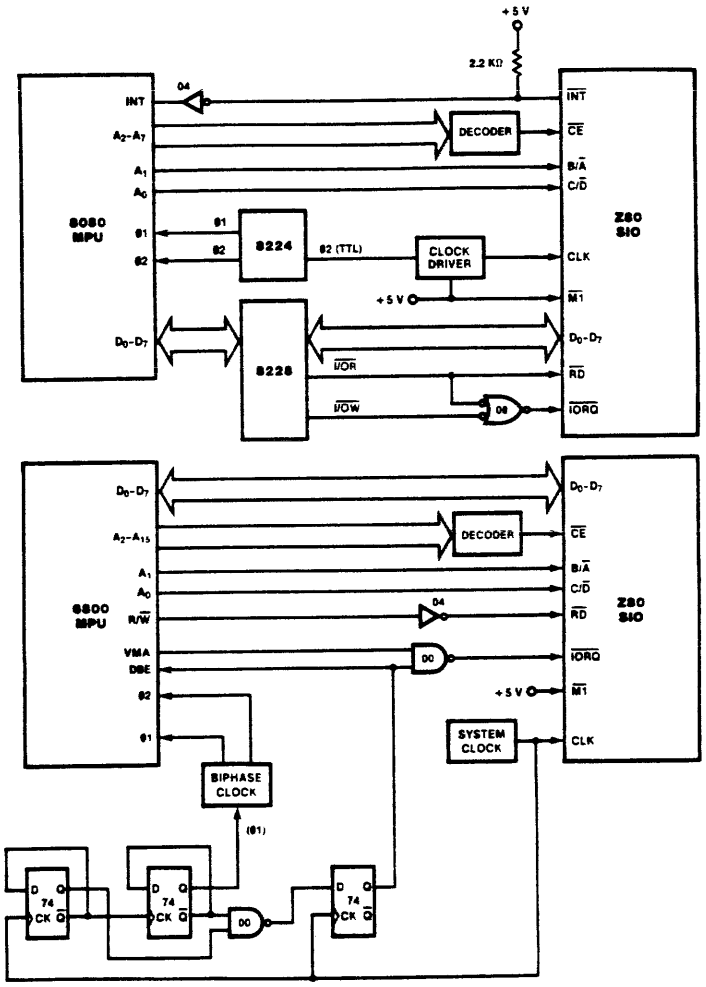


Figure 2. SIO Hardware Interfacing

Operational Considerations.

All of the SIO options to be discussed here are software controllable and are set by the CPU. Thus, use of the SIO begins with an initialization phase where the various options are set by writing control bytes. These options are established separately for each of the two channels supported by the SIO if both channels are used. Before giving an overview of how initialization is done, we will describe some of the basic characteristics of SIO operations that are common to both the Polled and Interrupt-driven modes.

Addressing the SIO

The CPU must have a means to identify any specific I/O device, including any attached SIO. In a Z80 CPU environment, this is done by using the lower 8 bits of the address bus (A_0-A_7). Typically, the A_1 bit is wired to the SIO's B/$\overline{A}$ input pin for selecting access to Channel A or Channel B, and the A_0 bit is wired to the SIO's C/$\overline{D}$ input pin for selecting the use of the data bus as an avenue for transferring control/status information (C) or actual data messages (D). The remaining bits of the address bus, A_2-A_7, contain a port address that uniquely identifies the SIO device. These latter six lines are usually wired to an external decoding chip which activates that SIO's Chip Enable ($\overline{CE}$) input pin when its address appears on A_2-A_7 of the address bus.

The bar notation drawn above the names of certain signal lines, such as B/$\overline{A}$ and C/$\overline{D}$, refer to signals which are interpreted as active when their logic sense—and voltage level—is Low. For example, the B/$\overline{A}$ pin specifies Channel B of the SIO when it carries a logic 1 (high voltage) and it specifies Channel A when it carries a logic 0 (low voltage).

Asynchronous Format Operations

Bits per Character. The SIO can receive or transmit 5, 6, 7, or 8 bits per character. This can be different for transmission and reception, and different for each channel. ASCII characters, for example, are usually transmitted as 7 bits. The SIO can in fact transmit fewer than 5 bits per character when set to the 5-bit mode; this is discussed further in the section entitled "Questions and Answers."

Parity. A parity bit is an additional bit added to a character for error checking. The parity bit is set to 0 or 1 in order to make the total number of 1s in the character (including parity bit) even or odd, depending on whether even or odd parity is selected. The SIO can be set either to add an optional parity bit to the "bits per character" described above, or not to add such a bit. When a parity bit is included, either even or odd parity can be chosen. This selection can be made independently for each channel.

Start and Stop Bits. There are two types of framing bits for each character: start and stop. When transmitting asynchronously, the SIO automatically inserts one start bit (logic 0) at the beginning of each character transmitted. The SIO can be programmed to set the number of stop bits inserted at the end of each character to either 1, 1½, or 2. The receiver always checks for 1 stop bit. Stop bits refer to the length of time that the stop value, a logic 1, will be transmitted; thus 1½ stop bits means that a 1 will be transmitted for the length of clock time that 1½ bits would normally take up. A logic 1 level that continues after the specified number of stop bits is called a "marking" condition or "mark bits."

CPU-SIO Character Transfers

The SIO always passes 8-bit bytes to the CPU for each character received, no matter how many "bits per character" are specified in the SIO initialization phase. If the number of "bits per character" is less than eight, parity and/or stop bits will be included in the byte sent to the CPU. The received character starts with the least-significant bit (D_0) and continues to the most-significant bit; it is immediately followed by the parity bit (if parity is enabled) and by the stop bit, which will be logic 1 unless there is a framing error. The remainder of the byte, if space is still available, is filled with logic 1s (marking). If the "bits per character" is eight, then the byte sent to the CPU will contain only the data bits. In all cases, the start bit is stripped off by the SIO and is not transmitted to the CPU.

Clock Divider

The SIO has five input pins for clock signals. One of these inputs (CLK) is used only for internal timing and does not affect transmission or reception rates. The other four clock inputs ($\overline{RxCA}$, $\overline{TxCA}$, $\overline{RxCB}$, and $\overline{TxCB}$) are used for timing the reception and transmission rates in Channels A and B. Only these last four are involved in "clock dividing." A clock divider within the SIO can be programmed to cause reception/transmission clocking at the actual input clock rate or at 1/16, 1/32, or 1/64 of the input clock rate. The receiver and transmitter clock divisions within a given channel must be the same, although their input clock rates can be different. The x1 clock rate can be used only if the transitions of the Receive clock are synchronized to occur during valid data bit times.

Auto Enables

The SIO has an Auto Enables feature that allows automatic SIO response and telephone answering. When Auto Enables is set for a particular channel, a transition to logical 0 (Low input level) on the respective Data Carrier Detect ($\overline{DCD}$) input will enable reception, and a transition to logical 0 on the respective Clear To Send ($\overline{CTS}$) input will enable transmission. This is described below under the heading "Modem Control."

Special Receive Conditions

There are three error conditions that can occur when the SIO is receiving data. Each of these will cause a status bit to be set, and if operating in Interrupt mode, the SIO can optionally be programmed to interrupt the CPU on such an error. The error conditions are called "special receive conditions" and they include:

- **Framing error.** If a stop bit is not detected in its correct location after the parity bit (if used) or after the most-significant data bit (if parity is not used), a framing error will result. The start bit preceding the character's data bits is not considered in determining a framing error, although character assembly will not begin until a start bit is detected.
- **Parity error.** If parity bits are attached by the external I/O device and checked by the SIO while receiving characters, a parity error will occur whenever the number of logic 1 data bits in the character (including the parity bit) does not correspond to the odd/even setting of the parity-checking function.
- **Receiver overrun error.** SIO buffers can hold up to three characters. If a character is received when the buffers are full (i.e., characters have not been read by the CPU), an SIO receiver overrun error will result. In this case, the most recently received character overwrites the next most recently received character.

Modem Control

Five signal lines on the SIO are provided for optional modem control, although these lines can also be used for other general-purpose control functions. They are:

$\overline{RTS}$ (Request To Send). An output from the SIO to tell its modem that the SIO is ready to transmit data.

$\overline{DTR}$ (Data Terminal Ready). An output from the SIO to tell its modem that the SIO is ready to receive data.

$\overline{CTS}$ (Clear To Send). An input to the SIO from its modem that enables SIO transmission if the Auto Enables function is used.

$\overline{DCD}$ (Data Carrier Detect). An input to the SIO from its modem that enables SIO reception if the Auto Enables function is used.

$\overline{SYNC}$ (Synchronization). A spare input to the SIO in asynchronous applications. This input may be used for the Ring Indicator function, if necessary, or for general-purpose inputs.

In most applications of asynchronous I/O that use modems, the $\overline{RTS}$ and $\overline{DTR}$ control lines and the Auto Enables function are activated during the initialization sequence, and they are left active until no further I/O is expected. This causes the SIO to tell its modem continuously that the SIO is ready to transmit and receive data, and it allows the modem to enable automatically the SIO's transmission and reception of data. Figure 3 illustrates this.

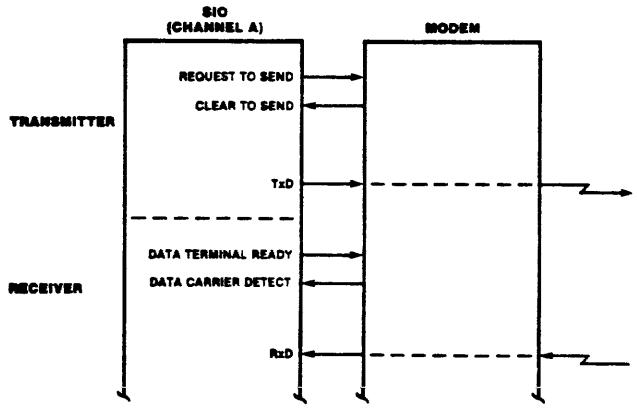


Figure 3. Modem Control (Single Channel)

External/ Status Interrupts

A change in the status of certain external inputs to the SIO will cause status bits in the SIO to be set. In the Polled Mode, these status bits can be read by the CPU. In the Interrupt mode, the SIO can also be programmed to interrupt the CPU when the change occurs. There are three such "external/status" conditions that can cause these events:

- **DCD.** Reflects the value of the $\overline{DCD}$ input.
- **CTS.** Reflects the value of the $\overline{CTS}$ input.
- **Break.** A series of logic 0 or "spacing" bits.

Note that the DCD and CTS status bits are the inverse of the SIO lines, i.e., the DCD bit will be 1 when the $\overline{DCD}$ line is Low.

Any transition in any direction (i.e., to logic 0 or to logic 1) on any of these inputs to the SIO will cause the related status bit to be latched and (optionally) cause an interrupt. The SIO status bits are latched after a transition on any one of them. The status must be reset (using an SIO command) before new transitions can be reflected in the status bits.

Initialization

The SIO contains eight write registers for Channel B (WR0-WR7) and seven write registers for Channel A (all except write register WR2). These are described fully in the *Z80 SIO Technical Manual* and are summarized in Appendix B. The registers are programmed separately for each channel to configure the functional personality of the channel. WR2 exists only in the Channel B register set and contains the interrupt vector for both channels. Bits in each register are named D_7 (most significant) through D_0. With the exception of WR0, programming the write registers requires two bytes: the first byte is to WR0 and contains pointer bits for selection of one of the other registers; the second byte is written to the register selected. WR0 is a special case in that all of the basic commands can be written to it with a single byte.

There are also three read registers, named RR0 through RR2, from which status results of operations can be read by the CPU (see Appendix B). Both channels have a set of read registers, but register RR2 exists only in Channel B.

Let us now look at the typical sequence of write registers that are loaded to initialize the SIO for either Polled or Interrupt-driven asynchronous I/O. Figure 4 illustrates the sequence. Except for step E, this loading is done for each channel when both are used. Steps E and F are described further in the section on "Interrupt-Driven Environments."

Registers WR6 and WR7 are not used in asynchronous I/O. They apply only to synchronous communication.

The related publications on the SIO should be referred to at this point. They will be necessary in following the discussion of functions. In particular, the following material should be reviewed:

Z80 SIO Technical Manual, pages 9-12 ("Asynchronous Operation")

Z80 SIO Technical Manual, pages 29-37 ("Z80 SIO Programming")

A. Load WR0. This is done to reset the SIO.

B. Load WR4. This specifies the clock divider, number of stop bits, and parity selection. Since register WR4 establishes the general form of I/O for which the SIO is to be used, it is best to set WR4 values first.

C. Load WR3. This specifies the number of receive bits per character, Auto Enable selection, and turns on the receiver enabling bit.

D. Load WR5. This specifies the number of transmit bits per character, turns off the bit that transmits the Break signal, turns on the bits indicating Data Terminal Ready and Request To Send, and turns on the transmitter enabling bit.

E. Load WR2. (Interrupt mode only and Channel B only.) This specifies the interrupt vector.

F. Load WR1. (Interrupt mode only.) This specifies various interrupt-handling options that will be explained later.

NOTES:
Steps A through F are performed in sequence.
*Channel B only.
†Interrupt mode only. Polling mode begins I/O after step D

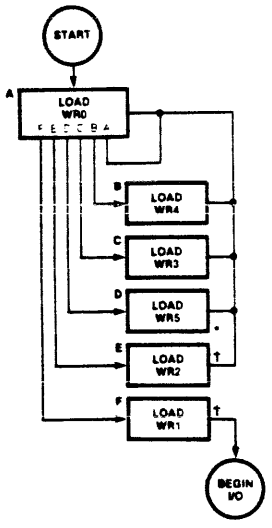


Figure 4. Typical Initialization Sequence (One Channel)

SECTION 3

Polled Environments.

In a typical Polled environment, the SIO is initialized and then periodically checked for completion of an I/O operation. Of course, if the checking is not frequent enough, received characters may be lost or the transmitter may be operated at a slower data rate than that of which it is capable. Initialization for Polled I/O follows the general outline described in the last section. We now give an overview of routines necessary for the CPU to check whether a character has been received by the SIO or whether the SIO is ready to transmit a character.

Character Reception

To check whether a character has been received, and to obtain a received character if one is available, the sequence illustrated in Figure 5 should be followed after the SIO is initialized. We assume that reception was enabled during initialization; if it was not, the Rx Enable bit in register WR3 must be turned on before reception can occur. This must be done for each channel to be checked.

Bit D_0 of register RR0 is set to 1 by the SIO if there is at least one character available to be received. The SIO contains a three-character input buffer for each channel, so more than one character may be available to be received. Removing the last available character from the read buffer for a particular channel turns off bit D_0.

If bit D_0 of register RR0 is 0, then no character is available to be received. In this case it is recommended that checks be made of bit D_7 to determine if a Break sequence (null character plus a framing error) has been received. If so, a Reset External/Status Interrupts command should be given; this will set the External/Status bits in register RR0 to the values of the signals currently being received. Thus, if the Break sequence has terminated, the next check of bit D_7 will so indicate. It may also be desirable to check bit 3 of register RR0 which reports the value of the Data Carrier Detect (DCD) bit.

In any case, if bit D_0 of register RR0 is 0, polled receive processing terminates with no character to receive. Depending on the facilities of the associated CPU, this step may be repeated until a character is available (or possibly a time-out occurs), or the CPU may return to other tasks and repeat this process later.

If bit D_0 of register RR0 is 1, then at least one character is available to be read. In this case, the value of register RR1 should first be read and stored to avoid losing any error information (the manner in which it is read is explained later). The character in the data register is then read. Note that the character must be read to clear the buffer even if there is an error found.

Finally, it is necessary to check the value stored from register RR1 to determine if the character received was valid. Up to three bits need to be checked: bit 6 is set to 1 for a framing error, bit 5 is set to 1 for a receiver overrun error (which occurs when the receive buffers are overwritten, i.e., no character has been removed and more than three characters have been received), and bit 4 is set to 1 for a parity error (if parity is enabled at initialization time). In case of a receiver overrun or parity error, an Error Reset command must be given to reset the bits.

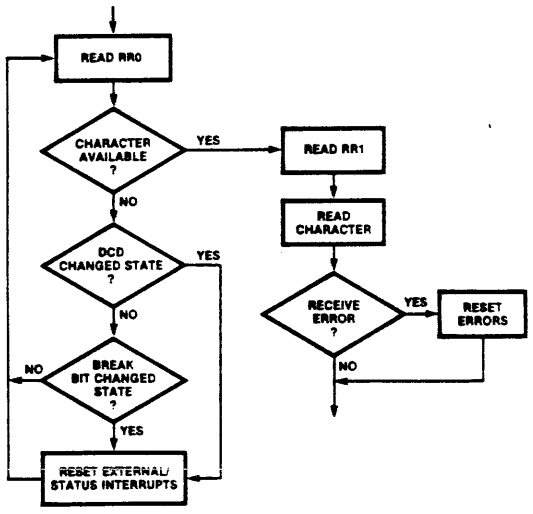


Figure 5. Polled Receive Routine

Character Transmission

To check that an initialized SIO is ready to transmit a character on a channel, and if so to transmit the character, the steps illustrated in Figure 6 should be followed. We assume that the Request To Send (RTS) bit in WR5, if required by the external receiving device, and the Transmit (Tx) Enable bit were set at initialization.

Depending on the external receiving device, the following bits in register RR0 should be checked: bit 3 (DCD), to determine if a data carrier has been detected; bit 5 (CTS), to determine if the device has signalled that it is clear to send; and bit 7 (Break), to determine if a Break sequence has been received. If any of these situations have occurred, the bits in register RR0 must be reset by sending the Reset External/Status Interrupts command, and the transmit sequence must be started again.

Next, bit 2 of register RR0 is checked. If this bit is 0, then the transmit buffer is not empty and a new character cannot yet be transmitted. Depending on the capabilities of the CPU, this is repeated until a character can be transmitted (or a timeout occurs), or the CPU may return to other tasks and start again later.

If bit 2 of register RR0 is 1, then the transmit buffer is empty and the CPU may pass the character to be transmitted to the SIO, completing the transmit processing. On the Z80 CPU, this is done with an OUT instruction to the SIO data port.

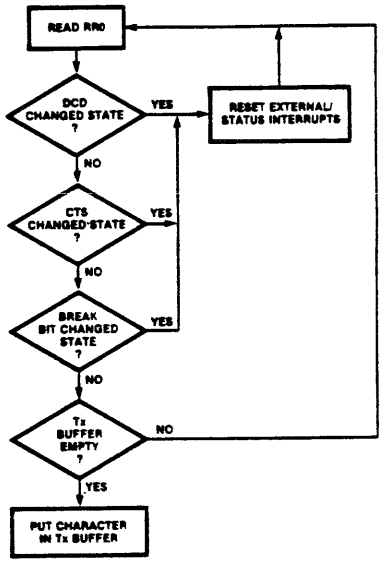


Figure 6. Polled Transmit

Assumptions for an Example

Now let us consider some examples in more detail. We assume we are given an external device to which we will input and output 8-bit characters, with odd parity, using the Auto Enables feature. We will support this device with I/O polling routines following the patterns illustrated in Figures 5 and 6. We assume that the CPU will provide space to receive characters from the SIO as fast as the characters are received by the SIO, and that the CPU will transfer characters as fast as the output can be accomplished by the SIO.

We specify this example by giving the control bytes (commands) written to the SIO and the status bytes that must be read from the SIO. Recall that to write a command to a register, except register WR0, the number of the register to be written is first sent to register WR0; the following byte will be sent to the named register. Similarly, to read a register other than RR0 (the default), the number of the register to be read is sent to register WR0; the following byte will return the register named.

Initialization

We begin with the initialization code for the SIO. This follows the outline illustrated in Figure 4. In the following sample code, each time register WR0 is changed to point to another register, the Reset External/Status Interrupts command is given simultaneously. Whenever a transition on any of the external lines occurs, the bits reporting such a transition are latched until the Reset External/Status Interrupts command is given. Up to two transitions can be remembered by the SIO. Therefore, it is desirable to do at least two different Reset External/Status Interrupts commands as late as possible in the initialization so that the status bits reflect the most recent information. Since it doesn't hurt, we include these commands each time WR0 is changed to point to another register. This is an easy way to code the initialization to insure that the appropriate resets occur.

In the example below, the logic states on the C/$\overline{D}$ control line and the system data bus (D_7-D_0) are illustrated, together with comments.

Initialization (Continued)

Bits sent to the SIO									Effects and Comments
C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	1	1	0	0	0	Channel Reset command sent to register WR0 (D_5-D_3).
1	0	0	0	1	0	1	0	0	Point WR0 to WR4 (D_2-D_0) and issue a Reset External/ Status Interrupts command (D_5-D_3). Throughout the initialization, whenever we point WR0 to another register, we will also issue this command for the reasons noted above.
1	1	1	0	0	1	1	0	1	Set WR4 to indicate the following parameters (from left to right): A. Run at 1/64 the input clock rate (D_7-D_6). B. Disable the sync bits and send out 2 stop bits per character (D_5-D_2). C. Enable odd parity (D_1-D_0).
1	0	0	0	1	0	0	1	1	Point WR0 to WR3.
1	1	1	1	0	0	0	0	1	Set WR3 to indicate the following: A. 8-bit characters to be received (D_7-D_6). B. Auto Enables on (D_5). C. Receive (Rx) Enable on (D_0).
1	0	0	0	1	0	1	0	1	Point WR0 to WR5.
1	1	1	1	0	1	0	1	0	Set WR5 to indicate the following: A. Data Terminal Ready (DTR) on (D_7). B. 8-bit characters to be transmitted (D_6-D_5). C. Break not to be transmitted (D_4). D. Transmit (Tx) Enable on (D_3). E. Request To Send (RTS) on (D_1).

Reset and Error Sequences

In the receive and transmit routines that follow, we treat errors such as a transition on the Data Carrier Detect line by calling for a "reset sequence" to set the values in read register RR0 to reflect the current values found at the pins. This sequence consists of giving the Reset External/Status Interrupts command and beginning the driver over again. The command takes the form of a write to register WR0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	1	0	0	0	0

Permits the status bits in RR0 to reflect current status.

This command does not turn off the latches for such things as parity errors stored in bits 4-6 of register RR1. When such an error occurs and the latches must be reset, we will call for an "error sequence." This sequence consists of giving the Error Reset command and beginning the driver over again. The command also takes the form of a write to register WR0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	1	0	0	0	0

Resets the latches in register RR1.

When specifying the result of reading register RR0 or RR1 or specifying data, we will indicate the values read as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
D	D	D	D	D	D	D	D

Read a byte from the designated register..

Receive and Transmit Routines

Now we will first give an example of the receive routine. This parallels the preceding discussion of "Character Reception."

The framing error in this routine is reported on a character-by-character basis and it is not necessary to execute an "error sequence" if it is the only error received. However, it is not harmful to do so.

Next, we give an example of transmission code that parallels the above discussion on "Character Transmission."

Receive and Transmit Routines (Continued)

C/D	Bits sent and received D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Effects and Comments (Receive Routine)
1	D	D	D	D	D	D	D	D	Read a byte from RR0 (the default read register); if $D_0 = 0$ then no character is ready to be received. In this case, if D_7 (Break) or D_3 (Data Carrier Detect) have changed state, then execute a "reset sequence." If $D_0 = 0$ and D_7 and D_3 have not changed state, then no character is ready to be received; either loop on this read or try again later.
1	0	0	0	0	0	0	0	1	Point WR0 to read from RR1; we will now check for errors in the character read. Note that Reset External/Status Interrupt Commands are not done normally to avoid losing a line-status change.
1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read a byte from RR1; if either bit $D_6 = 1$ (framing error), $D_5 =$ (receive overrun error), or $D_4 = 1$ (parity error), the character is invalid and an "error sequence" should be executed after the following step.
0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read in the data byte received. This must be done to clear the SIO buffer even if an error is detected.

C/D	Bits sent and received D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Effects and Comments (Transmit Routine)
1	D	D	D	D	D	D	D	D	Read a byte from RR0; if either bit D_3 (Data Carrier Detect), D_5 (Clear To Send) or D_7 (Break) have changed state, a "reset sequence" should be executed. If D_3, D_5 and D_7 have not changed state, then if $D_2 = 0$, the transmit buffer is not yet empty and a transmit cannot take place; either loop, reading RR0, or try again later.
0	D	D	D	D	D	D	D	D	Send the data byte to be transmitted.

SECTION 4

Interrupt-Driven Environments.

In a typical interrupt-driven environment, the SIO is initialized and the first transmission, if any, is begun. Thereafter, further I/O is interrupt driven. When action by the CPU is needed, an SIO interrupt causes the CPU to branch to an interrupt service routine after the CPU first saves state information.

In common usage, if I/O is interrupt driven, all interrupts are enabled and each different type of interrupt is used to cause a CPU branch to a different memory address. There is perhaps one frequent exception to this: parity errors are sometimes checked only at the end of a sequence of characters. The SIO facilitates this kind of operation since the parity error bit in read register RR1 is latched; once the bit is set it is not reset until an explicit reset operation is done. Thus, if a parity error has occurred on any character since last reset, bit 4 in register RR1 will be set. It is then possible to set register WR1 so that parity errors do not cause an error interrupt when a character is received. The user then has the obligation to poll for the value of the parity bit upon completion of the sequence.

SIO initialization for Interrupt mode normally requires two steps not used in Polled mode: an interrupt vector (if used) must be stored in write register WR2 of Channel B and write register WR1 must be initialized to specify the form of interrupt handling. It is preferable to initialize the interrupt vector in WR2 first. In this way an interrupt that arrives after the enabling bits are set in WR1 will cause proper interrupt servicing.

Interrupt Vectors

The interrupt vector, register WR2 of Channel B, is an 8-bit memory address. When an interrupt occurs (and note that an interrupt can only occur after interrupts have been enabled by writing to register WR1) the interrupt vector is normally taken as one byte of an address used by the CPU to find the location of the interrupt service routine. It is also possible to cause the particular type of interrupt condition to modify the address vector in WR2 before branching, resulting in a branch to a different memory location for each interrupt condition. This is a very useful construct; it permits short, special-purpose interrupt routines. The alternative, to have one general-purpose interrupt routine which must determine the situation before proceeding, can be quite inefficient. This is usually undesirable since the speed of interrupt-service routines is often a critical factor in determining system performance.

Interrupt Vectors (Continued)

There are at most eight different types of interrupts that the SIO may cause, four for each of the two channels. If bit 1 in register WR1 of Channel B has been turned on so that an interrupt will modify the interrupt vector, the three bits (1-3) of the vector will be changed to reflect the particular type of interrupt. These interrupts follow a hardware-set priority as follows, starting with the highest priority:

Channel A Special Receive Condition sets bits 3-1 of WR1 to 111,

Channel A Character Received sets bits 3-1 to 110,

Channel A Transmit Buffer Empty sets bits 3-1 to 100,

Channel A External/Status Transition sets bits 3-1 to 101.

Channel B Special Receive Condition sets bits 3-1 to 011,

Channel B Character Received sets bits 3-1 to 010,

Channel B Transmit Buffer Empty sets bits 3-1 to 000,

Channel B External/Status Transition sets bits 3-1 to 001.

For example, suppose that the interrupt vector had the value 11110001 and the Status Affects Vector bit is enabled, along with all interrupt-enable bits. When an External/Status transition occurs in Channel A, the three zeros (bits 3-1) would be modified to 101, yielding an interrupt vector of 11111011. The value of the interrupt vector, as modified, may be tained by reading register RR2 in Channel B.

Note that when a character is received, either the Special Receive Condition or Rx Character Available interrupt will occur, depending on whether or not an error occurred; the two will never occur simultaneously. Therefore, these two interrupts have equal priority. Note also that you can select not to be interrupted on some of the eight conditions; in this case, the presence of a particular condition for which interrupts are not desired can be determined by polling.

Suppose that interrupts have been enabled for all possible cases, and that the Status Affects Vector bit has also been enabled, allowing a different routine to handle each possible interrupt. As each interrupt causes a branch to a location only two bytes higher than the last interrupt, it is not possible to place a routine directly at the location where the vectored interrupt branches. In a Z80 CPU environment, these addresses refer to a table in memory which contains the actual starting location of the interrupt service routine. Also, since the state information saved by a CPU is rarely all of the information necessary to properly preserve a computation state, a typical interrupt service routine will begin by saving additional information and end by restoring that information. This is shown briefly in the examples of code in Appendix A.

It is possible to connect several SIOs using the interrupt mechanism and the IEI and IEO lines on the SIO to determine a priority for interrupt service. This mechanism is discussed on page 42 of the *Z80 SIO Technical Manual* and in the *Z80 Family Program Interrupt Structure Manual*. We do not go into it further in this application note.

Initialization

In general, the initialization procedure illustrated in Figure 4 can still be followed. All six steps (A through F) are required here. After completing the first four steps, which are the same as initialization for polled I/O, it is necessary to load an interrupt vector into WR2 of Channel B. Information is then written into register WR1 specifying which interrupts are to be enabled and whether a specific kind of interrupt should modify the interrupt vector.

Now let us give an example. As in the polled example, we assume that we are given a device to which we will input and output 8-bit characters, with odd parity, using the Auto Enables feature. We also assume the CPU will provide space to store characters as received.

We do not discuss the SIO commands and registers in detail. This is done in the *Z80 SIO Technical Manual*. A summary of the register bit assignments taken from the *Z80 SIO Serial Input/Output Product Specification* is included at the end of this note. Recall that to write a register other than register WR0, the number of the register to be written is first sent to register WR0, and the following byte will be sent to the named register. Similarly, to read a register other than RR0 (the default), the number of the register to be read is first written to register WR0 and the next byte read will return the contents of the register named.

In our example below, each time register WR0 is changed to point to another register, the Reset External/Status Interrupts command is also given. Whenever a transition on any of the external/status lines occurs, the bits reporting the transition are latched until the Reset External/Status Interrupts command is given. Up to two transitions can be remembered by the internal logic of the SIO. Therefore, it is desirable to do at least two different Reset External/Status Interrupt commands as late as possible in the initialization so that the status bits reflect the most recent information. Since it doesn't hurt, we give these commands each

Initialization (Continued)

time WR0 is changed to point to another register. This is an easy way to code the initialization to assure that the appropriate resets occur.

The columns below show the logic states on the C/$\overline{D}$ control line and the system data bus (D_7-D_0), together with co nments.

	Bits sent to the SIO								
C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments**
1	0	0	0	1	1	0	0	0	Channel Reset command sent to register WR0 (D_5-D_3).
1	0	0	0	1	0	1	0	0	Point WR0 to WR4 (D_2-D_0) and issue a Reset External/Status Interrupts command (D_5-D_3). Throughout the initialization, whenever we point WR0 to another register we will also issue a Reset External/Status Interrupts command for the reasons noted above.
1	1	1	0	0	1	1	0	1	Set WR4 to indicate the following parameters (from left to right): A. Run at 1/64 the clock rate (D_7-D_6). B. Disable the sync bits and send out 2 stop bits per character (D_5-D_2). C. Enable odd parity (D_1-D_0).
1	0	0	0	1	0	0	1	1	Point WR0 to WR3.
1	1	1	1	0	0	0	0	1	Set WR3 to indicate the following: A. 8-bit characters to be received (D_7-D_6). B. Auto Enables on (D_5). C. Rx Enable on (D_0).
1	0	0	0	1	0	1	0	1	Point WR0 to WR5.
1	1	1	1	0	1	0	1	0	Set WR5 to indicate the following: A. Data Terminal Ready (DTR) on (D_7). B. 8-bit characters to be transmitted (D_6-D_5). C. Break not to be transmitted (D_4). D. Tx Enable on (D_3). E. Request To Send (RTS) on (D_1).
1	0	0	0	1	0	0	1	0	Point WR0 to WR2 (Channel B only).
1	1	1	1	0	0	0	0	0	Set the interrupt vector to point to address 11100000 (which is hex E0 and decimal 224). Once interrupts are enabled, they will cause a branch to this memory location, modified as described above if the Status Affects Vector bit is turned on (which it will be here). This vector is only set for Channel B, but it applies to both channels. It has no effect when set in Channel A.
1	0	0	0	1	0	0	0	1	Point WR0 to WR1.
1	0	0	0	1	0	1	1	1	Set WR1 to indicate the following: A. Cause interrupts on all characters received, treating a parity error as a Special Receive Condition interrupt (D_4-D_3). B. Turn on the Status Affects Vector feature, causing interrupts to modify the status vector—meaningful only on Channel B, but will not hurt if set for Channel A (D_2). C. Enable interrupts due to transmit buffer being empty (D_1). D. Enable External/Status interrupts (D_0).

Special Receive Condition Interrupts

A Special Receive Condition interrupt occurs (a) if a parity error has occurred, (b) if there is a receiver overrun error (data is being overwritten because the channel's three-byte receiver buffer is full and a new character is being received), or (c) if there is a framing error. The processing in this case is the following:

1. Issue an Error Reset command (to register WR0) to reset the latches in register RR1.
2. Read the character from the read buffer and discard it to empty the buffer.

It may be desirable to read and store the value of register RR1 to gather statistics on performance or determine whether to accept the character. In some applications, a character may still be acceptable if received with a framing error.

In specifying the result of reading register RR0, RR1, or specifying data, we will indicate the values as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
D	D	D	D	D	D	D	D

Read a byte from the designated register.

We now present an example of processing a Special Receive Condition interrupt.

	Bits sent and received								
$C/\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments**
1	0	0	0	0	0	0	0	1	If we need to know what kind of error occurred, we point WR0 to read from RR1. Note that the Reset External/Status Interrupts command is not used. This avoids losing a valid interrupt.
1	D	D	D	D	D	D	D	D	Read a byte from RR1; one or more of bit D_6 (framing error), D_5 (receive overrun error), or D_4 (parity error) will be 1 to indicate the specific error.
1	0	0	1	1	0	0	0	0	Give an Error Reset command to reset all the error latches.
0	D	D	D	D	D	D	D	D	Read in the data byte received. This must be done to clear the receiver buffer, but the character will generally be disregarded.

Received (Rx) Character Interrupts

When an Rx Character Available interrupt occurs, the character need only be read from the read buffer and stored. If parity is enabled with character lengths of 5, 6, or 7 bits, the received parity bit will be transferred with the character. Any unused bits will be 1s.

External/ Status Interrupts

To respond to an External/Status Interrupt, all that is necessary is to send a Reset External/Status Interrupts command. However, if you wish to find the specific cause of the interrupt, it is necessary to read register RR0. In this case, the complete processing takes the following form:

	Bits sent and received								
$C/\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments**
1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read register RR0; bit D_7 (Break), D_5 (Clear To Send), or D_3 (Data Carrier Detect) will have had a transition to indicate the cause of the interrupt.
1	0	0	0	1	0	0	0	0	Give a Reset External/Status Interrupts command to set the latches in RR0 to their current values and stop External/Status Interrupts until another transition occurs.

Transmit (Tx) Buffer Empty Interrupts

The final kind of interrupt is a Tx Buffer Empty interrupt. If another character is ready to be transmitted on this channel, a Tx Buffer Empty interrupt indicates that it is time to do so. To respond to this interrupt, you need only send the next character. If no other character is ready to transmit, it may be desirable to mark the availability of the transmit mechanism for future use. In addition, you should send a Reset Tx Interrupt Pending command. This command prevents further transmitter interrupts until the next character has been loaded into the transmitter buffer.

The Reset Tx Interrupt Pending command to WR0 takes the following form:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	0	1	0	0	0

Reset Tx Interrupt Pending command; no Tx Empty interrupts will be given until after the next character has been placed in the transmit buffer.

Z80 Assembler Code

To take these examples further, let us use Z80 Assembler code to implement the routines for a single channel. We assume that the location stored in register WR2 points to the appropriate interrupt service routine. We also assume that the following constants have already been defined:

SIOctrl. The address of the SIO's Channel B control port (we assume Channel B in order to include code to initialize the interrupt vector).

SIOdata. The address of the SIO's Channel B data port.

X. An address pointing to locations in memory that will be used to store various values.

We will write data as binary constants; the "B" suffix indicates this. In most cases, binary constants will be referred to by the command names. We begin with the initialization routine:

```
INIT:   LD      C,SIOctrl       ;place the address of the SIO in the C register for
                                ; use in subsequent output
        LD      A,00011000B     ;load Channel Reset command in A register
        OUT     (C),A           ;give Channel Reset command

        LD      A,00010100B     ;write to register WR0 pointing it to register WR4
        OUT     (C),A
        LD      A,11001101B     ;output basic I/O parameters to WR4
        OUT     (C),A

        LD      A,00010011B     ;write to register WR0 pointing it to register WR3
        OUT     (C),A
        LD      A,11100001B     ;output receive parameters to WR3
        OUT     (C),A

        LD      A,00010101B     ;write to register WR0 pointing it to register WR5
        OUT     (C),A
        LD      A,11101010B     ;output transmit parameters to WR5
        OUT     (C),A

        LD      A,00010010B     ;write to register WR0 pointing it to register WR2
                                ; (Channel B only)
        OUT     (C),A
        LD      A,11100000B     ;output the interrupt vector to WR2; in this case it is
                                ; decimal location 224
        OUT     (C),A

        LD      A,00010001B     ;write to register WR0 pointing it to register WR1
        OUT     (C),A
        LD      A,00010111B     ;output interrupt parameters to WR1
        OUT     (C),A

        RET                     ;return from initialization routine
```

Now let us look first at some sample codes for the Special Receive Condition interrupt routine, following the example above.

This is followed by a simple receive interrupt routine that will fetch the character received and store it in a temporary location.

```
SIOspecint:  PUSH    AF         ;save registers which will be used in this routine
             LD      A,00000001B ;write to register WR0 pointing it to register RR1
             OUT     (SIOctrl),A
             IN      A,(SIOctrl) ;fetch register RR1
             LD      (X),A      ;store result for later error analysis
             LD      A,00110000B ;send an Error Reset command to reset device
                                 ; latches
             OUT     (SIOctrl),A
             IN      A,(SIOdata) ;fetch the character received—we will discard this
                                 ; character since an error occurred during its
                                 ; reception
             POP     AF         ;restore saved registers
             EI                 ;enable interrupts
             RETI               ;return from interrupt
```

Z80 Assembler Code (Continued)

```
SIOrecint:  PUSH  AF          ;save registers which will be used in this routine
            IN    A,(SIOdata) ;fetch the character received
            LD    (X) ,A      ;store result for later use
            POP   AF          ;restore saved registers
            EI                ;enable interrupts
            RETI              ;return from interrupt
```

Of course, this last routine is probably far too simple to be useful. It is more likely that an interrupt routine will fill up a buffer of characters. A more complex example of a receive interrupt routine is contained in the chapter entitled "A Longer Example."

We now give a simple interrupt routine for an External/Status Interrupt, again assuming that the status contents of SIO register RR0 are stored in temporary location X:

```
SIOextint:  PUSH  AF          ;save registers which will be used in this routine
            LD    A,00010000B ;send a Reset External/Status Interrupts command
            OUT   (SIOctrl) ,A
            IN    A,(SIOctrl) ;fetch register RR0
            LD    (X) ,A      ;store result for later analysis
            POP   AF          ;restore saved registers
            EI                ;enable interrupts
            RETI              ;return from interrupt
```

Finally, we give the processing for a transmit interrupt routine in the case where no more characters are to be transmitted.

It is likely that this code would just be a portion of a more general transmit interrupt routine which would transmit a buffer-full of information at a time. A more complex example is included in the section entitled "A Longer Example."

```
SIOtrnint:  PUSH  AF          ;save registers which will be used in this routine
            LD    A,00101000B ;send a Reset Tx Interrupt Pending command
            OUT   (SIOctrl) ,A
            POP   AF          ;restore saved registers
            EI                ;Enable Interrupts
            RETI              ;Return From Interrupt
```

SECTION 5

Hardware Considerations

Questions and Answers.

Q: Can a sloppy system clock cause problems in SIO operation?

A: Yes; the specifications for the system clock are very tight and must be met closely to prevent SIO malfunction. The clock high voltage must be greater than $V_{CC}-0.6V$ but less than +5.5V. The clock low voltage must be greater than −0.3V but less than +0.45V. The transitions between these two levels must be made in less than 30 ns. This does not apply to the $\overline{RxC}$ and $\overline{TxC}$ inputs which are standard TTL levels.

Q: When is a received character available to be read?

A: Data will be available a maximum of 13 system clock cycles from the rising edge of the $\overline{RxC}$ signal which samples the last bit of the data.

Q: What is the maximum time between character-insertion for transmission and next-character transmission?

A: This will vary depending on the speed of the line over which the character is being transmitted.

Q: Are the control lines to the SIO synchronous with the system clock so that noise may exist on the buses any time before setup requirements are satisfied?

A: Yes.

Q: In asynchronous use must receiver and transmitter clock rates be the same?

A: No, the SIO allows receive and transmit for each channel to use a different clock (thus up to four different clocks for receiving and transmitting data can be used on each SIO). However, the clock multiplier for each channel must be the same.

Q: Do Wait states have to be added when using the SIO with other processors other than the Z80 CPU?

A: No, provided that setup times specified for the SIO are met.

Q: If the Auto Enables bit in register WR3 is set, will a change in state on the $\overline{DCD}$ (Data Carrier Detect) or $\overline{CTS}$ (Clear To Send) lines still cause an interrupt?

A: Yes, provided that External/Status Interrupts are enabled (bit 0 in register WR1).

Q: Is the $\overline{M1}$ line used by the SIO if no interrupts are enabled?

A: No, and in this case the $\overline{M1}$ input should be tied high.

Q: Will the SIO continue to interrupt for a condition if the condition persists and the interrupt remains enabled?

A: Yes.

Q: What is the maximum data rate of the SIO?

A: It is 1/5 the rate of the system clock (CLK). For example, if the system clock operates at 4 MHz, the SIO's maximum transfer rate is 800K bits (100K bytes) per second.

Q: What pins are edge sensitive and should be strapped to avoid strange interrupts?

A: The external synchronization ($\overline{SYNC}$) pins and any other external status pins that are not used, including $\overline{CTS}$, and $\overline{DCD}$.

Q: What happens if the transmitter or receiver is disabled, while processing a character, by turning off its associated enable bit (bit 3 in register WR5 for transmit or bit 0 in register WR3 for receive)?

A: The transmitter will complete the character transmission in an orderly fashion. The receiver, however, will not finish. It will lose the character being received and no interrupt will occur.

Register Contents

Q: Does the Tx Buffer Empty (bit 2 in register RR0 get set when the last byte in the buffer is in the process of being shifted out?

A: No. The bit is set when the transmit buffer has already become empty. Similarly, the Tx Buffer Empty interrupt will not occur until the buffer is empty. The same is true for reception: the Rx Character Available bit (bit 0 in register RR0) is not set until the entire character is in the receive buffer, and the Rx Character Available interrupt will not occur until the entire character has been moved into the buffer.

Q: If an Rx Overrun error occurs (and bit 5 of register RR1 becomes latched on) because a new character has arrived, which character gets lost?

A: The most recently received character overwrites the next most recently received character.

Q: Does the Reset External/Status Interrupts command reset any of the status bits in register RR0?

A: No. However, when a transition occurs on any of the five External/Status bits in register RR0, all of the status bits are latched in their current position until a Reset External/Status Interrupts command is issued. Thus, the command does permit the appropriate bits of register RR0 to reflect the current signal values and should be done immediately after processing each transition on the channel.

Special Uses

Q: If the CPU does not have the return from interrupt sequence (RETI instruction on the Z80 CPU), how may the SIO be informed of the completion of interrupt handling?

A: This may be done by writing the Return From Interrupt command (binary, 00111000) to WR0 in Channel A of the SIO.

Q. If the CPU can be interrupted but cannot be used with vectored interrupts, how should processing be done?

A: Immediately after being interrupted, proceed in a manner similar to polling the SIO for both receive and transmit. Alternatively, the Status Affects Vector bit (bit 2 in register WR1) may be set and a 0 byte placed into the interrupt vector (register WR2 in Channel B). Then, the contents of the interrupt vector can be used to determine the cause of the interrupt and the channel on which the interrupt occurred. This can be queried by reading register RR1 of Channel B. Also, $\overline{M1}$ should be tied High and no equivalent to an interrupt acknowledge should be issued.

Q: How can the Wait/Ready ($\overline{W/RDY}$) signal be used by the CPU in asynchronous I/O?

A: The $\overline{W/RDY}$ signal is most commonly used in Block Transfer Mode with a DMA, and this use is described in the *Z80 DMA Technical Manual*. However, $\overline{W/RDY}$ may be directly connected to the Z80 CPU $\overline{WAIT}$ line in order to use the block I/O instructions OTDR, OTIR, INDR, and INIR. In this case, the SIO can be used for block transfer reception. To do this, the SIO is configured to interrupt on the first character received only (by settings bits 4 and 3 of register WR1 to 01) and additional characters are sensed using the $\overline{W/RDY}$ line. The block I/O instructions decrement a byte counter to determine when I/O is complete.

Q: Can the $\overline{SYNC}$ pin have any use in asynchronous I/O?

A: It may be used as a general-purpose input. For example, by connecting it to a modem ring indicator, the status of that ring indicator can be monitored by the CPU.

Q: How can the SIO be used to transmit characters containing fewer than 5 bits?

A: First, set bits 6 and 5 in register WR5 to indicate that five or fewer bits per character will be transmitted. The SIO then determines the number of bits to actually transmit from the data byte itself. The data byte should consist of zero or more 1s, three 0s, and the data to be transmitted. Thus, beginning the data byte with 11110001 will cause only the last bit to be transmitted:

Contents of data byte (d = arbitrary value)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	d	1
1	1	1	0	0	0	d	d	2
1	1	0	0	0	d	d	d	3
1	0	0	0	d	d	d	d	4
0	0	0	d	d	d	d	d	5

*The rightmost number of bits indicated will be transmitted.

Q: Can a Break sequence be sent for a fixed number of character periods?

A: Yes. Break is continuously transmitted as logic 0 by setting bit 4 of register WR5. You can then send characters to the transmitter as long as the Break level is desired to persist. A Break signal, rather than the characters sent, will actually be transmitted, but each bit of each character sent will be clocked as if it were transmitted. The All Sent bit, bit 0 of register RR1, is set to 1 when the last bit of a character is clocked for transmission, and this may be used to determine when to reset bit 4 of register WR5 and stop the Break signal.

Q: If a Break sequence is initiated by setting bit 4 of register WR5, will any character in the process of being transmitted be completed?

A: No. Break is effective immediately when bit 4 of WR5 is set. The "all sent" bit in register RR1 should be monitored to determine when it is safe to initiate a Break sequence.

SECTION 6

A Longer Example.

In this section, we give a longer example of asynchronous interrupt-driven full-duplex I/O using the SIO. The code for this example is contained in Appendix A, and the basic routines are flow charted in Figures 7-12.

The example includes code for initialization of the SIO, initialization of a receive buffer interrupt routine, and a transfer routine which causes a buffer of up to 80 characters of information to be transmitted on Channel A and a buffer of up to 80 characters of information to be received from Channel A. The transfer routine stops when either all data is received or an error occurs. Completion of an operation on a buffer for both receive and transmit is indicated by a carriage return character. Additional routines (not included in this example) would be needed to call the initialization code and initiate the transfer routine. Therefore, we do not present a complete example; that would only be possible when all details of a particular communication environment and operating system were known.

The code begins by defining the value of the SIO control and data channels, followed by location definitions for the interrupt vector. There is then a series of constant definitions of the various fields in each register of the SIO. This is followed by a table-driven SIO initialization routine called "SIO_init," shown in Figure 7, which uses the table beginning at the location "SIOItable." The SIO_Init routine initializes the SIO with exactly the same

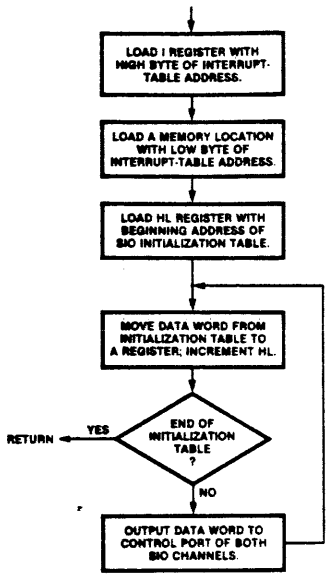


Figure 7. Interrupt-Driven Initialization Routine

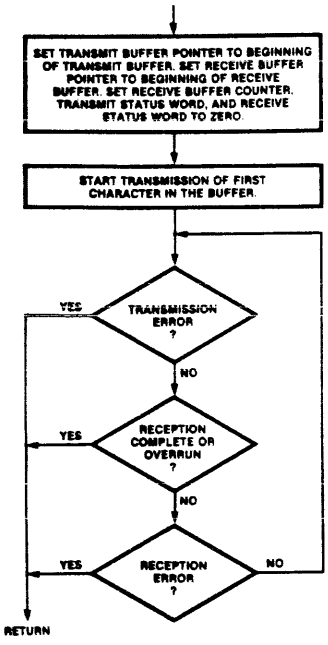


Figure 8. Interrupt-Driven Transmit Routine

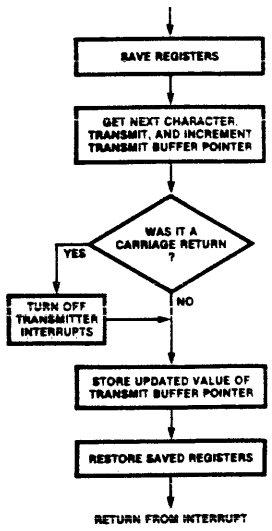


Figure 9. Transmitter Buffer Empty Interrupt Routine

26-0003-0346 26-0003-0347 26-0003-0348

A Longer Example (Continued)

parameters as the interrupt-driven example in the previous section. The table-driven version is presented simply as an alternative means of coding this material.

A short routine for filling the receive buffer with "FF" (hex) characters and buffer definitions follows the SIO_Init routine. This in turn is followed by the transfer routine, Figure 8, which begins transmitting on Channel A; transmission and reception is thereafter directed by the interrupt routines. After the transfer routine begins output, it checks for various error conditions and loops until there is either completion or an error.

Then the four interrupt routines follow: TxBEmpty, Figure 9, is called on a transmit buffer interrupt; it begins transmission of the next character in the buffer. A carriage return stops transmission. RecvChar, Figure 10, is called on a normal receive interrupt; it places the received character in the buffer if the buffer is not full and updates receive counters. The routines SpRecvChar, Figure 11, and ExtStatus, Figure 12, are error interrupts; they update information to indicate the nature of the error.

The code of this example can be used in a situation where data is being sent to a device which echoes the data sent. In such a case, the transmit and receive buffers could be compared upon completion for line or transmission errors.

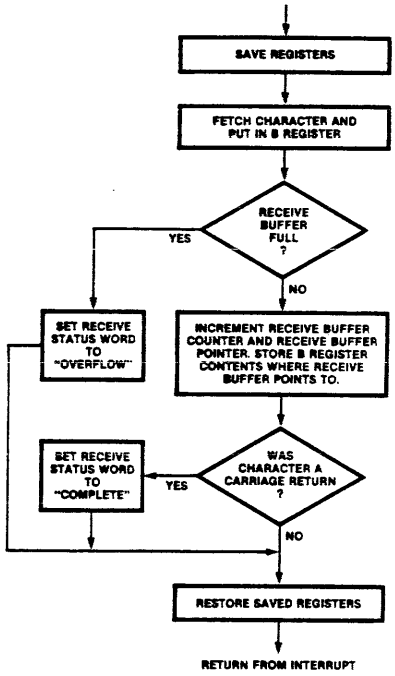


Figure 10. Receive Character Interrupt Routine

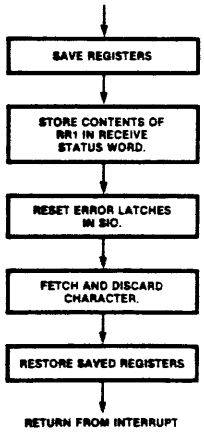


Figure 11. Special Receive Condition Interrupt Routine

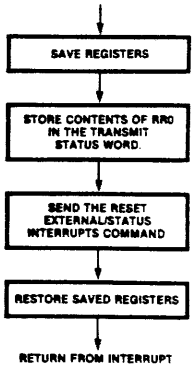


Figure 12. External/Status Interrupt Routine

Appendix A

Interrupt-Driven Code Example

SIO Port Identifiers and System Address Bus Addresses

```
SIO:        EQU     40H

SIOAData:   EQU     SIO+1
SIOACtrl:   EQU     SIO+2
SIOBData:   EQU     SIO+3
SIOBCtrl:   EQU     SIO+4
```

Table of Interrupt Vectors

The table (Int_Tab) starts at the lowest priority vector, which should be dddd000d.

```
            ORG     0D0H        ;starts at address with low
                                ; byte = 11010000
Int_Tab:    DEFW    TxBEmpty    ;interrupt types for Channel B
            DEFW    ExtStat
            DEFW    RxChar
            DEFW    SpRxCond

            DEFW    TxBEmpty    ;interrupt types for Channel A
            DEFW    ExtStat
            DEFW    RxChar
            DEFW    SpRxCond
```

Command Identifiers and Values

Includes all control bytes for asynchronous and synchronous I/O.

WR0 Commands

```
R0:         EQU     00H         ;SIO register pointers
R1:         EQU     01H
R2:         EQU     02H
R3:         EQU     03H
R4:         EQU     04H
R5:         EQU     05H
R6:         EQU     06H
R7:         EQU     07H

NC:         EQU     00H         ;Null Code
SA:         EQU     08H         ;Send Abort (SDLC)
RESI:       EQU     10H         ;Reset Ext/Stat Int
CHRST:      EQU     18H         ;Channel Reset
EIONRC:     EQU     20H         ;Enable Int On Next Rx Char
RTIP:       EQU     28H         ;Reset Tx Int Pending
ER:         EQU     30H         ;Error Reset
RFI:        EQU     38H         ;Return From Int
RRCC:       EQU     40H         ;Reset Rx CRC Checker
RTCG:       EQU     80H         ;Reset Tx CRC Generator
RTUEL:      EQU     0C0H        ;Reset Tx Under/EOM Latch
```

WR1 Commands

```
WAIT:       EQU     00H         ;Wait function
DRCVRI:     EQU     00H         ;Disable Receive interrupts
EXTIE:      EQU     01H         ;External interrupt enable
XMTRIE:     EQU     02H         ;Transmit interrupt enable
SAVECT:     EQU     04H         ;Status affects vector
FIRSTC:     EQU     08H         ;Rx interrupt on first character
PAVECT:     EQU     10H         ;Rx interrupt on all characters
                                ; (parity affects vector)
PDAVCT:     EQU     18H         ;Rx interrupt on all characters
                                ; (parity doesn't affect vector)
WRONRT:     EQU     20H         ;Wait/Ready on receive
RDY:        EQU     40H         ;Ready function
WRDYEN:     EQU     80H         ;Wait/Ready enable
```

WR2 Commands

```
IV:         EQU     00H
```

WR3 Commands

```
B5:         EQU     00H         ;Receive 5 bits/character
RENABL:     EQU     01H         ;Receiver enable
ENRCVR:     EQU     01H         ;Receiver enable
SCLINH:     EQU     02H         ;Sync character load inhibit
ADSRCH:     EQU     04H         ;Address search mode
RCRCEN:     EQU     08H         ;Receive CRC enable
HUNT:       EQU     10H         ;Enter hunt mode
AUTOEN:     EQU     20H         ;Auto enables
B7:         EQU     40H         ;Receive 7 bits/character
B6:         EQU     80H         ;Receive 6 bits/character
B8:         EQU     0C0H        ;Receive 8 bits/character
```

WR4 Commands

```
SYNC:       EQU     00H         ;Sync modes enable
NOPRTY:     EQU     00H         ;Disable parity
ODD:        EQU     00H         ;Odd parity
MONO:       EQU     00H         ;8 bit sync character
C1:         EQU     00H         ;X1 clock mode
PARITY:     EQU     01H         ;Enable parity
EVEN:       EQU     02H         ;Even parity
S1:         EQU     04H         ;1 stop bit/character
S1HALF:     EQU     08H         ;1 and a half stop bits/character
S2:         EQU     0CH         ;2 stop bits/character
BISYNC:     EQU     10H         ;16 bit sync character
SDLC:       EQU     20H         ;SDLC mode
ESYNC:      EQU     30H         ;External sync mode
C16:        EQU     40H         ;X16 clock mode
C32:        EQU     80H         ;X32 clock mode
C64:        EQU     0C0H        ;X64 clock mode
```

WR5 Commands

```
T5:         EQU     00H         ;Transmit 5 bits/character
XCRCEN:     EQU     01H         ;Transmit CRC enable
RTS:        EQU     02H         ;Request to send
SELCRC:     EQU     04H         ;Select CRC-16 polynomial
XENABL:     EQU     08H         ;Transmitter enable
BREAK:      EQU     10H         ;Send break
T7:         EQU     20H         ;Transmit 7 bits/character
T6:         EQU     40H         ;Transmit 6 bits/character
T8:         EQU     60H         ;Transmit 8 bits/character
DTR:        EQU     80H         ;Data terminal ready
```

Initialization

```
SIO_Init:   LD      HL, Int_Tab
            LD      A,H
            LD      I,A
            LD      A,L
            LD      (I_Loc),A
            LD      HL, SIOItable

Init_Loop:  LD      A,(HL)      ;loop for initialization
            INC     HL
            CP      0
            RET     Z
            OUT     (SIOACtrl),A
            OUT     (SIOBCtrl),A
            JR      Init_Loop

SIOItable:  DEFB    CR          ;table for initialization
            DEFB    R4 + RESI
            DEFB    C64 + ODD + PARITY + S2
            DEFB    R3 + RESI
            DEFB    B8 +  AUTOEN + ENRCVR
            DEFB    R5 + RESI
            DEFB    DTR + RTS + T8 + XENABL
            DEFB    R2 + RESI

I_Loc:      DEFS    1           ;location of int table
            DEFB    R1 + RESI   ;address
            DEFB    EXTIE + XMTRIE + SAVECT + PAVECT
            DEFB    0
```

Receiver Buffer Initialization

```
Buf_Init:   LD      A,BufLength   ;fill receiver buffer
            LD      B,A           ; with FF characters
            LD      HL,RBuffer    ; to detect errors
            LD      A,0FFH

Buf_1:      LD      (HL),A        ;a loop for Buf_Init
            INC     HL
            DJNZ    Buf_1
            RET

BufLength:  EQU     80            ;buffer length
XBuffer:    DEFS    BufLength     ;Tx buffer starting location
RBuffer:    DEFS    BufLength     ;Rx buffer starting location

XBufPtr:    DEFS    2             ;Tx pointer
RBufPtr:    DEFS    2             ;Rx pointer
RBufCtr:    DEFS    1             ;Rx counter
```

Transmit Routine (see Figure 8)

Initiates transmission of a buffer-full of data and terminates when an error is detected or a complete buffer has been received.

```
RxStat:     DEFS    1             ;Receive Status Word
TxStat:     DEFS    1             ;Transmit Status Word

Complete:   EQU     1
CR:         EQU     0DH
Break:      EQU     80H
EOM:        EQU     80H
Overflow:   EQU     0FFH

Transfer:   LD      HL,XBuffer    ;setup to begin Tx
            INC     HL
            LD      (XBufPtr),HL
            LD      HL,RBuffer
            LD      (RBufPtr),HL
            XOR     A             ;A = 0
            LD      (RBufCtr),A
            LD      (TxStat),A
            LD      (RxStat),A

            LD      A,SIOADataa   ;start Tx task
            LD      C,A
            LD      HL,(XBuffer)  ;first character
            LD      A,(HL)
            OUT     (C),A

Tloop:      LD      A,(TxStat)    ;await Tx completion or error
            CP      0
            RET     NZ
            LD      A,(RxStat)
            CP      Overflow
            RET     Z
            CP      Complete
            RET     Z
            JR      NZ,Tloop
            RET
```

Transmitter Buffer Empty Routine (see Figure 9)

```
TxBEmpty    PUSH    AF
            PUSH    BC
            PUSH    HL

            LD      HL,(XBufPtr)
            LD      A,SIOADataa
            LD      C,A
            LD      A,(HL)
            OUTI
            CP      CR
            JR      NZ, TxBExit   ;last character?

            LD      A,RTIP        ;Reset Tx Int Pending
            INC     C
            OUT     (C),A         ;to control port

TxBExit:    LD      (XBufPtr),HL  ;save pointer
            POP     HL
            POP     BC
            POP     AF
            EI
            RETI
```

Receive Character Routine (see Figure 10)

```
RxChar:     PUSH    AF
            PUSH    BC

            LD      A,SIOADataa
            LD      C,A
            IN      A,(C)         ;get character
            LD      B,A
            LD      A,(RBufCtr)
            CP      BufLength
            JR      Z,Over

            INC     A             ;bump counter
            LD      (RBufCtr),A
            LD      A,B
            LD      HL,(RBufPtr)  ;bump pointer
            LD      (HL),A
            INC     HL
            LD      (RBufPtr),HL
            CP      CR
            JR      NZ,RxExit

            LD      A,Complete
            LD      (RxStat),A
            JR      RxExit

Over:       LD      A,Overflow    ;indicate error
            LD      (RxStat),A

RxExit:     POP     BC
            POP     AF
            EI
            RETI
```

Special Receive Condition Routine (see Figure 11)

```
SpRxCond:   PUSH    AF
            PUSH    BC

            LD      A,SIOADataa
            LD      C,A
            LD      A,R1          ;get RR1
            INC     C
            OUT     (C),A
            IN      A,(C)
            LD      (RxStat),A    ;save status
            LD      A,ER          ;Reset Errors
            DEC     C
            OUT     (C),A
            DEC     C
            IN      A,(C)         ;get character

            POP     BC
            POP     AF
            EI
            RETI
```

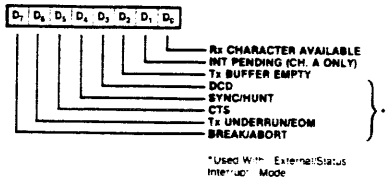
External/Status Routine (see Figure 12)

```
ExtStatus:  PUSH    AF
            PUSH    BC

            LD      A,SIOACtrl
            LD      C,A
            IN      A,(C)         ;get RR0
            LD      (TxStat),A
            LD      A,RESI        ;Reset Ext Stat Int
            OUT     (C),A

            POP     BC
            POP     AF
            EI
            RETI

END
```

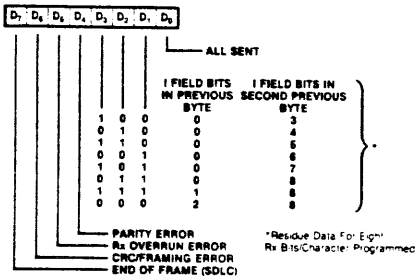
Appendix B

Read Register Bit Functions

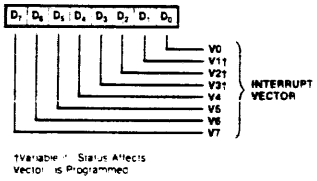
READ REGISTER 0



READ REGISTER 1†

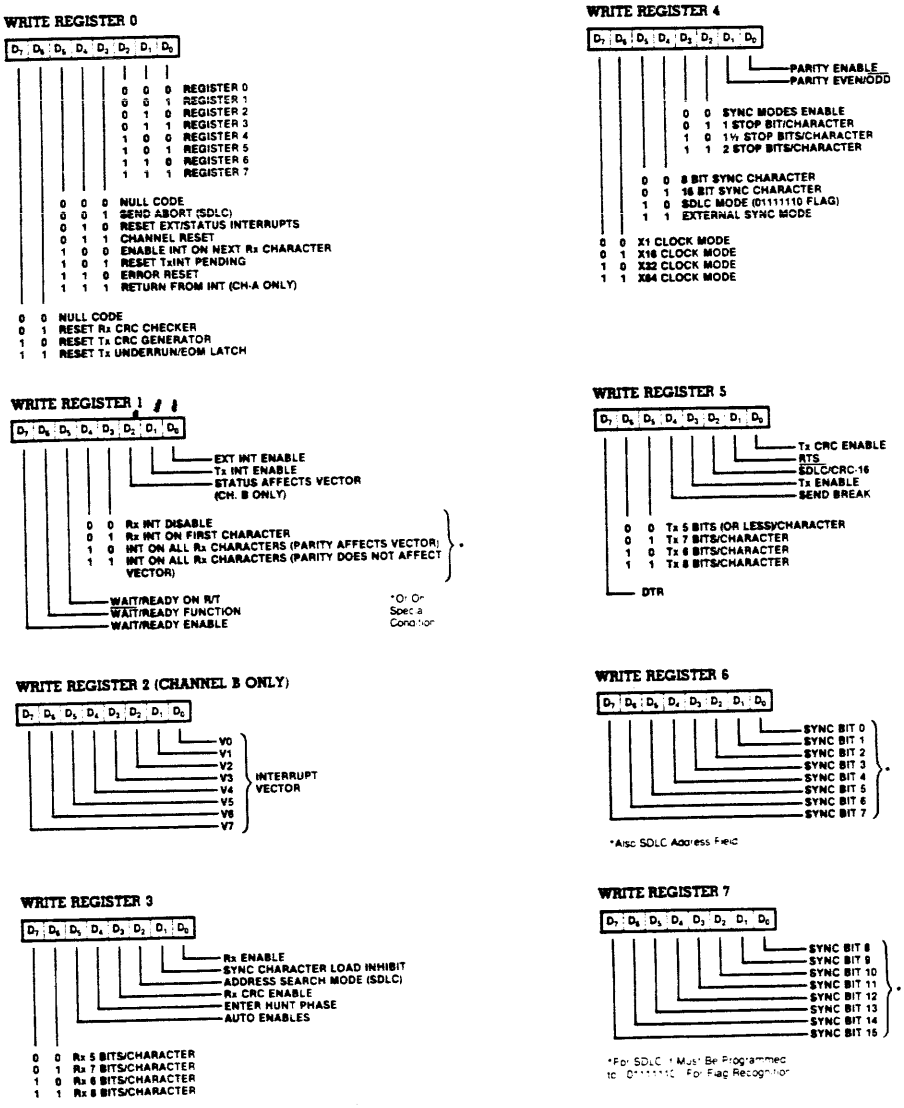


READ REGISTER 2



Appendix C

Write Register Bit Functions



Z8430
Z80® CTC Counter/ Timer Circuit

Zilog

Product Specification

June 1982

Features

- Four independently programmable counter/timer channels, each with a readable downcounter and a selectable 16 or 256 prescaler. Downcounters are reloaded automatically at zero count.
- Three channels have Zero Count/Timeout outputs capable of driving Darlington transistors.
- Selectable positive or negative trigger initiates timer operation.
- Standard Z-80 Family daisy-chain interrupt structure provides fully vectored, prioritized interrupts without external logic. The CTC may also be used as an interrupt controller.
- Interfaces directly to the Z-80 CPU or—for baud rate generation—to the Z-80 SIO.

General Description

The Z-80 CTC four-channel counter/timer can be programmed by system software for a broad range of counting and timing applications. The four independently programmable channels of the Z-80 CTC satisfy common microcomputer system requirements for event counting, interrupt and interval timing, and general clock rate generation.

System design is simplified because the CTC connects directly to both the Z-80 CPU and the Z-80 SIO with no additional logic. In larger systems, address decoders and buffers may be required.

Programming the CTC is straightforward: each channel is programmed with two bytes; a third is necessary when interrupts are enabled. Once started, the CTC counts down, reloads its time constant automatically, and resumes counting. Software timing loops are completely eliminated. Interrupt processing is simplified because only one vector need be specified; the CTC internally generates a unique vector for each channel.

The Z-80 CTC requires a single +5 V power supply and the standard Z-80 single-phase system clock. It is fabricated with n-channel silicon-gate depletion-load technology, and packaged in a 28-pin plastic or ceramic DIP.

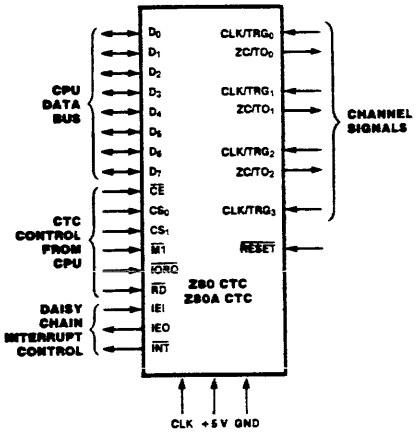


Figure 1. Pin Functions

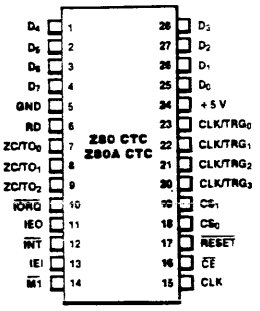


Figure 2. Pin Assignments

Functional Description

The Z-80 CTC has four independent counter/timer channels. Each channel is individually programmed with two words: a control word and a time-constant word. The control word selects the operating mode (counter or timer), enables or disables the channel interrupt, and selects certain other operating parameters. If the timing mode is selected, the control word also sets a prescaler, which divides the system clock by either 16 or 256. The time-constant word is a value from 1 to 256.

During operation, the individual counter channel counts down from the preset time constant value. In counter mode operation the counter decrements on each of the CLK/TRG input pulses until zero count is reached. Each decrement is synchronized by the system clock. For counts greater than 256, more than one counter can be cascaded. At zero count, the down-counter is automatically reset with the time constant value.

The timer mode determines time intervals as small as 4 μs (Z-80A) or 6.4 μs (Z-80) without additional logic or software timing loops. Time intervals are generated by dividing the system clock with a prescaler that decrements a preset down-counter.

Thus, the time interval is an in egral multiple of the clock period, the prescaler value (16 or 256) and the time constant that is preset in the down-counter. A timer is triggered automatically when its time constant value is programmed, or by an external CLK/TRG input.

Three channels have two outputs that occur at zero count. The first output is a zero-count/timeout pulse at the ZC/TO output. The fourth channel (Channel 3) does not have a ZC/TO output; interrupt request is the only output available from Channel 3.

The second output is Interrupt Request ($\overline{INT}$), which occurs if the channel has its interrupt enabled during programming. When the Z-80 CPU acknowledges Interrupt Request, the Z-80 CTC places an interrupt vector on the data bus.

The four channels of the Z-80 CTC are fully prioritized and fit into four contiguous slots in a standard Z-80 daisy-chain interrupt structure. Channel 0 is the highest priority and Channel 3 the lowest. Interrupts can be individually enabled (or disabled) for each of the four channels.

Architecture

The CTC has four major elements, as shown in Figure 3.

- CPU bus I/O
- Channel control logic
- Interrupt logic
- Counter/timer circuits

CPU Bus I/O. The CPU bus I/O circuit decodes the address inputs, and interfaces the CPU data and control signals to the CTC for distribution on the internal bus.

Internal Control Logic. The CTC internal control logic controls overall chip operating functions such as the chip enable, reset, and read/write logic.

Interrupt Logic. The interrupt control logic ensures that the CTC interrupts interface properly with the Z-80 CPU interrupt system. The logic controls the interrupt priority of the CTC as a function of the IEI signal. If IEI is High, the CTC has priority. During interrupt

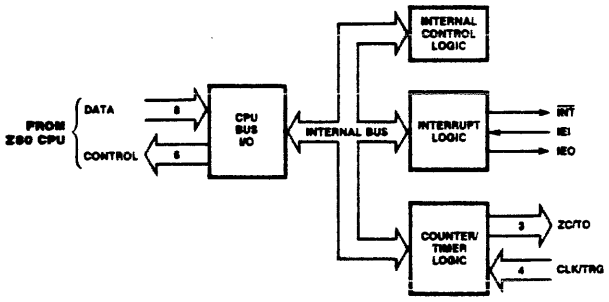


Figure 3. Functional Block Diagram

Architecture (Continued)

processing, the interrupt logic holds IEO Low, which inhibits the interrupt operation on lower priority devices. If the IEI input goes Low, priority is relinquished and the interrupt logic drives IEO Low.

If a channel is programmed to request an interrupt, the interrupt logic drives IEO Low at the zero count, and generates an $\overline{INT}$ signal to the Z-80 CPU. When the Z-80 CPU responds with interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$), then the interrupt logic arbitrates the CTC internal priorities, and the interrupt control logic places a unique interrupt vector on the data bus.

If an interrupt is pending, the interrupt logic holds IEO Low. When the Z-80 CPU issues a Return From Interrupt (RETI) instruction, each peripheral device decodes the first byte (ED_{16}). If the device has a pending interrupt, it raises IEO (High) for one $\overline{M1}$ cycle. This ensures that all lower priority devices can decode the entire RETI instruction and reset properly.

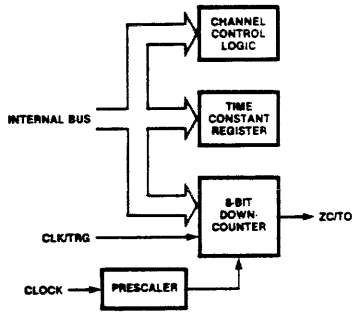


Figure 4. Counter/Timer Block Diagram

Counter/Timer Circuits. The CTC has four independent counter/timer circuits, each containing the logic shown in Figure 4.

Channel Control Logic. The channel control logic receives the 8-bit channel control word when the counter/timer channel is programmed. The channel control logic decodes the control word and sets the following operating conditions:

- Interrupt enable (or disable)
- Operating mode (timer or counter)
- Timer mode prescaler factor (16 or 256)
- Active slope for CLK/TRG input
- Timer mode trigger (automatic or CLK/TRG input)
- Time constant data word to follow
- Software reset

Time Constant Register. When the counter/timer channel is programmed, the time constant register receives and stores an 8-bit time constant value, which can be anywhere from 1 to 256 (0 = 256). This constant is automatically loaded into the down-counter when the counter/timer channel is initialized, and subsequently after each zero count.

Prescaler. The prescaler, which is used only in timer mode, divides the system clock frequency by a factor of either 16 or 256. The prescaler output clocks the down-counter during timer operation. The effect of the prescaler on the down-counter is a multiplication of the system clock period by 16 or 256. The prescaler factor is programmed by bit 5 of the channel control word.

Down-Counter. Prior to each count cycle, the down-counter is loaded with the time constant register contents. The counter is then decremented one of two ways, depending on operating mode:

- By the prescaler output (timer mode)
- By the trigger pulses into the CLK/TRG input (counter mode)

Without disturbing the down-count, the Z-80 CPU can read the count remaining at any time by performing an I/O read operation at the port address assigned to the CTC channel. When the down-counter reaches the zero count, the ZC/TO output generates a positive-going pulse. When the interrupt is enabled, zero count also triggers an interrupt request signal ($\overline{INT}$) from the interrupt logic.

Programming Each Z-80 CTC channel must be programmed prior to operation. Programming consists of writing two words to the I/O port that corresponds to the desired channel. The first word is a control word that selects the operating mode and other parameters; the second word is a time constant, which is a binary data word with a value from 1 to 256. A time constant word must be preceded by a channel control word.

After initialization, channels may be reprogrammed at any time. If updated control and time constant words are written to a channel during the count operation, the count continues to zero before the new time constant is loaded into the counter.

If the interrupt on any Z-80 CTC channel is enabled, the programming procedure should also include an interrupt vector. Only one vector is required for all four channels, because the interrupt logic automatically modifies the vector for the channel requesting service.

A control word is identified by a 1 in bit 0. A 1 in bit 2 indicates a time constant word is to follow. Interrupt vectors are always addressed to Channel 0, and identified by a 0 in bit 0.

Addressing. During programming, channels are addressed with the channel select pins CS_1 and CS_2. A 2-bit binary code selects the appropriate channel as shown in the following table.

Channel	CS_1	CS_0
0	0	0
1	0	1
2	1	0
3	1	1

Reset. The CTC has both hardware and software resets. The hardware reset terminates all down-counts and disables all CTC interrupts by resetting the interrupt bits in the control registers. In addition, the ZC/TO and Interrupt outputs go inactive, IEO reflects IEI, and D_0–D_7 go to the high-impedance state. All channels must be completely reprogrammed after a hardware reset.

The software reset is controlled by bit 1 in the channel control word. When a channel receives a software reset, it stops counting. When a software reset is used, the other bits in the control word also change the contents of the channel control register. After a software reset a new time constant word must be written to the same channel.

If the channel control word has both bits D_1 and D_2 set to 1, the addressed channel stops operating, pending a new time constant word. The channel is ready to resume after the new constant is programmed. In timer mode, if $D_3 = 0$, operation is triggered automatically when the time constant word is loaded.

Channel Control Word Programming. The channel control word is shown in Figure 5. It sets the modes and parameters described below.

Interrupt Enable. D_7 enables the interrupt, so that an interrupt output ($\overline{INT}$) is generated at zero count. Interrupts may be programmed in either mode and may be enabled or disabled at any time.

Operating Mode. D_6 selects either timer or counter mode.

Prescaler Factor. (Timer Mode Only). D_5 selects factor—either 16 or 256.

Trigger Slope. D_4 selects the active edge or slope of the CLK/TRG input pulses. Note that reprogramming the CLK/TRG slope during operation is equivalent to issuing an active edge. If the trigger slope is changed by a control word update while a channel is pending operation in timer mode, the result is the same as a CLK/TRG pulse and the timer starts. Similarly, if the channel is in counter mode, the counter decrements.

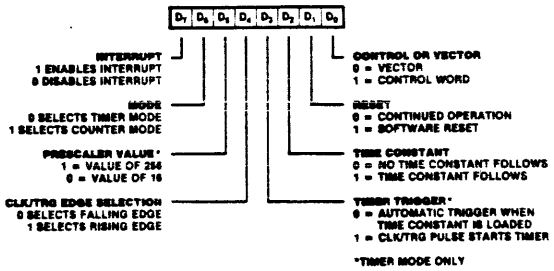


Figure 5. Channel Control Word

Programming (Continued)

Trigger Mode (Timer Mode Only). D_3 selects the trigger mode for timer operation. When D_3 is reset to 0, the timer is triggered automatically. The time constant word is programmed during an I/O write operation, which takes one machine cycle. At the end of the write operation there is a setup delay of one clock period. The timer starts automatically (decrements) on the rising edge of the second clock pulse (T_2) of the machine cycle following the write operation. Once started, the timer runs continuously. At zero count the timer reloads automatically and continues counting without interruption or delay, until stopped by a reset.

When D_3 is set to 1, the timer is triggered externally through the CLK/TRG input. The time constant word is programmed during an I/O write operation, which takes one machine cycle. The timer is ready for operation on the rising edge of the second clock pulse (T_2) of the following machine cycle. Note that the first timer decrement follows the active edge of the CLK/TRG pulse by a delay time of one clock cycle if a minimum setup time to the rising edge of clock is met. If this minimum is not met, the delay is extended by another clock period. Consequently, for immediate triggering, the CLK/TRG input must precede T_2 by one clock cycle plus its minimum setup time. If the minimum time is not met, the timer will start on the third clock cycle (T_3).

Once started the timer operates continuously, without interruption or delay, until stopped by a reset.

Time Constant to Follow. A 1 in D_2 indicates that the next word addressed to the selected channel is a time constant data word for the time constant register. The time constant word may be written at any time.

A 0 in D_2 indicates no time constant word is to follow. This is ordinarily used when the channel is already in operation and the new channel control word is an update. A channel will not operate without a time constant value. The only way to write a time constant value is to write a control word with D_2 set.

Software Reset. Setting D_1 to 1 causes a software reset, which is described in the Reset section.

Control Word. Setting D_0 to 1 identifies the word as a control word.

Time Constant Programming. Before a channel can start counting it must receive a time constant word from the CPU. During programming or reprogramming, a channel control word in which bit 2 is set must precede the time constant word to indicate that the next word is a time constant. The time constant word can be any value from 1 to 256 (Figure 6). Note that 00_{16} is interpreted as 256.

In timer mode, the time interval is controlled by three factors:

- The system clock period (ϕ)
- The prescaler factor (P), which multiplies the interval by either 16 or 256
- The time constant (T), which is programmed into the time constant register

Consequently, the time interval is the product of $\phi \times P \times T$. The minimum timer resolution is $16 \times \phi$ (4 μs with a 4 MHz clock). The maximum timer interval is $256 \times \phi \times 256$ (16.4 ms with a 4 MHz clock). For longer intervals timers may be cascaded.

Interrupt Vector Programming. If the Z-80 CTC has one or more interrupts enabled, it can supply interrupt vectors to the Z-80 CPU. To do so, the Z-80 CTC must be pre-programmed with the most-significant five bits of the interrupt vector. Programming consists of writing a vector word to the I/O port corresponding to the Z-80 CTC Channel 0. Note that D_0 of the vector word is always zero, to distinguish the vector from a channel control word. D_1 and D_2 are not used in programming the vector word. These bits are supplied by the interrupt logic to identify the channel requesting interrupt service with a unique interrupt vector (Figure 7). Channel 0 has the highest priority.

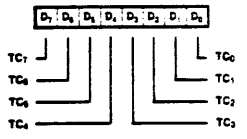


Figure 6. Time Constant Word

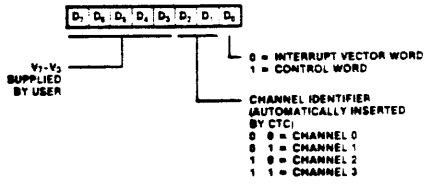


Figure 7. Interrupt Vector Word

Pin Description

$\overline{CE}$. *Chip Enable* (input, active Low). When enabled the CTC accepts control words, interrupt vectors, or time constant data words from the data bus during an I/O write cycle; or transmits the contents of the down-counter to the CPU during an I/O read cycle. In most applications this signal is decoded from the eight least significant bits of the address bus for any of the four I/O port addresses that are mapped to the four counter-timer channels.

CLK. *System Clock* (input). Standard single-phase Z-80 system clock.

CLK/TRG_0-CLK/TRG_3. *External Clock/Timer Trigger* (input, user-selectable active High or Low). Four pins corresponding to the four Z-80 CTC channels. In counter mode, every active edge on this pin decrements the down-counter. In timer mode, an active edge starts the timer.

CS_0-CS_1. *Channel Select* (inputs active High). Two-bit binary address code selects one of the four CTC channels for an I/O write or read (usually connected to A_0 and A_1).

D_0-D_7. *System Data Bus* (bidirectional, 3-state). Transfers all data and commands between the Z-80 CPU and the Z-80 CTC.

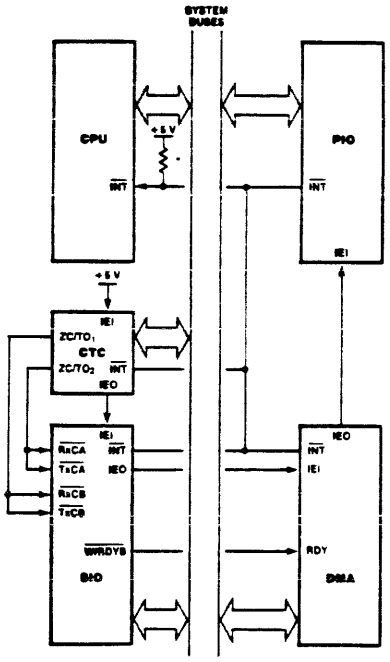


Figure 8. A Typical Z-80 Environment

$\overline{IEI}$. *Interrupt Enable In* (input, active High). A High indicates that no other interrupting devices of higher priority in the daisy chain are being serviced by the Z-80 CPU.

IEO. *Interrupt Enable Out* (output, active High). High only if IEI is High and the Z-80 CPU is not servicing an interrupt from any Z-80 CTC channel. IEO blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced.

$\overline{INT}$. *Interrupt Request* (output, open drain, active Low). Low when any Z-80 CTC channel that has been programmed to enable interrupts has a zero-count condition in its down-counter.

$\overline{IORQ}$. *Input/Output Request* (input from CPU, active Low). Used with $\overline{CE}$ and $\overline{RD}$ to transfer data and channel control words between the Z-80 CPU and the Z-80 CTC. During a write cycle, $\overline{IORQ}$ and $\overline{CE}$ are active and $\overline{RD}$ inactive. The Z-80 CTC does not receive a specific write signal; rather, it internally generates its own from the inverse of an active $\overline{RD}$ signal. In a read cycle, $\overline{IORQ}$, $\overline{CE}$ and $\overline{RD}$ are active; the contents of the down-counter are read by the Z-80 CPU. If $\overline{IORQ}$ and $\overline{M1}$ are both true, the CPU is acknowledging an interrupt request, and the highest priority interrupting channel places its interrupt vector on the Z-80 data bus.

$\overline{M1}$. *Machine Cycle One* (input from CPU, active Low). When $\overline{M1}$ and $\overline{IORQ}$ are active, the Z-80 CPU is acknowledging an interrupt. The Z-80 CTC then places an interrupt vector on the data bus if it has highest priority, and if a channel has requested an interrupt ($\overline{INT}$).

$\overline{RD}$. *Read Cycle Status* (input, active Low). Used in conjunction with $\overline{IORQ}$ and $\overline{CE}$ to transfer data and channel control words between the Z-80 CPU and the Z-80 CTC.

$\overline{RESET}$. *Reset* (input active Low). Terminates all down-counts and disables all interrupts by resetting the interrupt bits in all control registers; the ZC/TO and the Interrupt outputs go inactive; IEO reflects IEI; D_0-D_7 go to the high-impedance state.

ZC/TO_0-ZC/TO_2. *Zero Count/Timeout* (output, active High). Three ZC/TO pins corresponding to Z-80 CTC channels 2 through 0 (Channel 3 has no ZC/TO pin). In both counter and timer modes the output is an active High pulse when the down-counter decrements to zero.

Timing

Read Cycle Timing. Figure 9 shows read cycle timing. This cycle reads the contents of a down-counter without disturbing the count. During clock cycle T_2, the Z-80 CPU initiates a read cycle by driving the following inputs Low: $\overline{RD}$, $\overline{IORQ}$, and $\overline{CE}$. A 2-bit binary code at inputs CS_1 and CS_0 selects the channel to be read. $\overline{M1}$ must be High to distinguish this cycle from an interrupt acknowledge. No additional wait states are allowed.

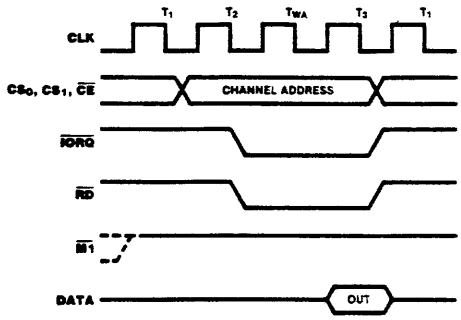


Figure 9. Read Cycle Timing

Write Cycle Timing. Figure 10 shows write cycle timing for loading control, time constant or vector words.

The CTC does not have a write signal input, so it generates one internally when the read ($\overline{RD}$) input is High during T_1. During T_2 $\overline{IORQ}$ and $\overline{CE}$ inputs are Low. $\overline{M1}$ must be High to distinguish a write cycle from an interrupt acknowledge. A 2-bit binary code at inputs CS_1 and CS_0 selects the channel to be addressed, and the word being written is placed on the Z-80 data bus. The data word is latched into the appropriate register with the rising edge of clock cycle T_3.

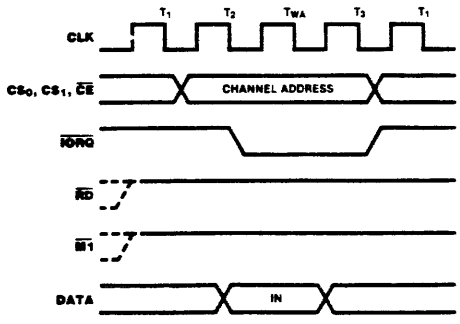


Figure 10. Write Cycle Timing

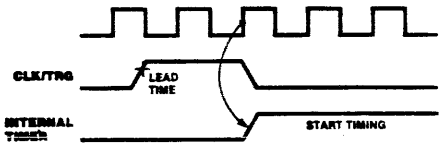


Figure 11. Timer Mode Timing

Timer Operation. In the timer mode, a CLK/TRG pulse input starts the timer (Figure 11) on the second succeeding rising edge of CLK. The trigger pulse is asynchronous, and it must have a minimum width. A minimum lead time (210 ns) is required between the active edge of the CLK/TRG and the next rising edge of CLK to enable the prescaler on the following clock edge. If the CLK/TRG edge occurs closer than this, the initiation of the timer function is delayed one clock cycle. This corresponds to the startup timing discussed in the programming section. The timer can also be started automatically if so programmed by the channel control word.

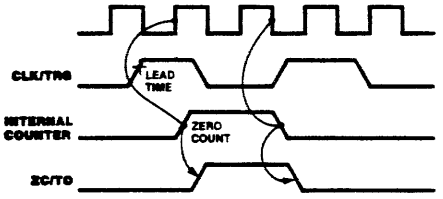


Figure 12. Counter Mode Timing

Counter Operation. In the counter mode, the CLK/TRG pulse input decrements the down-counter. The trigger is asynchronous, but the count is synchronized with CLK. For the decrement to occur on the next rising edge of CLK, the trigger edge must precede CLK by a minimum lead time as shown in Figure 12. If the lead time is less than specified, the count is delayed by one clock cycle. The trigger pulse must have a minimum width, and the trigger period must be at least twice the clock period.

The ZC/TO output occurs immediately after zero count, and follows the rising CLK edge.

Interrupt Operation

The Z-80 CTC follows the Z-80 sys em interrupt protocol for nested priority interrupts and return from interrupt, wherein the interrupt priority of a peripheral is determined by its location in a daisy chain. Two lines—IEI and IEO—in the CTC connect it to the system daisy chain. The device closest to the +5 V supply has the highest priority (Figure 13). For additional information on the Z-80 interrupt structure, refer to the *Z-80 CPU Product Specification* and the *Z-80 CPU Technical Manual.*

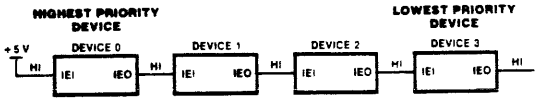


Figure 13. Daisy-Chain Interrupt Priorities

Within the Z-80 CTC, interrupt priority is predetermined by channel number: Channel 0 has the highest priority, and Channel 3 the lowest. If a device or channel is being serviced with an interrupt routine, it cannot be interrupted by a device or channel with lower priority until service is complete. Higher priority devices or channels may interrupt the servicing of lower priority devices or channels.

A Z-80 CTC channel may be programmed to request an interrupt every time its down-counter reaches zero. Note that the CPU must be programmed for interrupt mode 2. Some time after the interrupt request, the CPU sends an interrupt acknowledge. The CTC interrupt control logic determines the highest priority channel that is requesting an interrupt. Then, if the CTC IEI input is High (indicating that it has priority within the system daisy chain) it places an 8-bit interrupt vector on the system data bus. The high-order five bits of this vector ming process; the next two bits are provided by the CTC interrupt control logic as a binary code that identifies the highest priority channel requesting an interrupt; the low-order bit is always zero.

Interrupt Acknowledge Timing. Figure 14 shows interrupt acknowledge timing. After an interrupt request, the Z-80 CPU sends an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). All channels are inhibited from changing their interrupt request status when $\overline{M1}$ is active—about two clock cycles earlier than $\overline{IORQ}$. $\overline{RD}$ is High to distinguish this cycle from an instruction fetch.

The CTC interrupt logic determines the highest priority channel requesting an interrupt. If the CTC interrupt enable input (IEI) is High, the highest priority interrupting channel within the CTC places its interrupt vector on the data bus when $\overline{IORQ}$ goes Low. Two wait states (T_{WA}) are automatically inserted at this time to allow the daisy chain to stabilize. Additional wait states may be added.

Return from Interrupt Timing. At the end of an interrupt service routine the RETI (Return From Interrupt) instruction initializes the daisy chain enable lines for proper control of nested priority interrupt handling. The CTC decodes the 2-byte RETI code internally and determines whether it is intended for a channel being serviced. Figure 15 shows RETI timing.

If several Z-80 peripherals are in the daisy chain, IEI settles active (High) on the chip currently being serviced when the opcode ED_{16} is decoded. If the following opcode is $4D_{16}$, the peripheral being serviced is released and its IEO becomes active. Additional wait states are allowed.

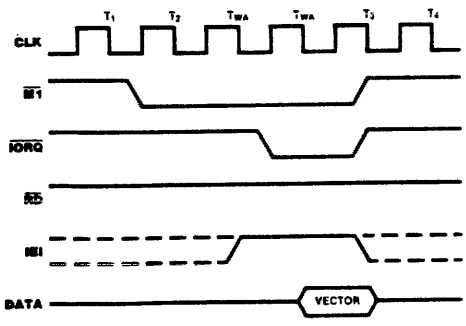


Figure 14. Interrupt Acknowledge Timing

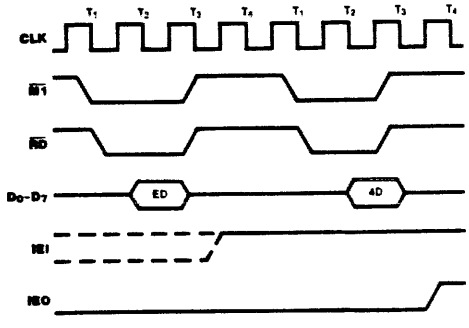


Figure 15. Return From Interrupt Timing

2041-0166, 0167, 0168

Z8470 Z80® DART Dual Asynchronous Receiver/Transmitter

Zilog

Product Specification

June 1982

Features

- Two independent full-duplex channels with separate modem controls. Modem status can be monitored.
- In x1 clock mode, data rates are 0 to 500K bits/second with a 2.5 MHz clock, or 0 to 800K bits/second with a 4.0 MHz clock.
- Receiver data registers are quadruply buffered; the transmitter is doubly buffered.
- Programmable options include 1, 1½ or 2 stop bits; even, odd or no parity; and x1, x16, x32 and x64 clock modes.
- Break generation and detection as well as parity-, overrun- and framing-error detection are available.
- Interrupt features include a programmable interrupt vector, a "status affects vector" mode for fast interrupt processing, and the standard Z-80 peripheral daisy-chain interrupt structure that provides automatic interrupt vectoring with no external logic.
- On-chip logic for ring indication and carrier-detect status.

Description

The Z-80 DART (Dual-Channel Asynchronous Receiver/Transmitter) is a dual-channel multi-function peripheral component that satisfies a wide variety of asynchronous serial data communications requirements in microcomputer systems. The Z-80 DART is used as a serial-to-parallel, parallel-to-serial converter/controller in asynchronous applications. In addition, the device also provides modem controls for both channels. In applications where modem controls are not needed, these lines can be used for general-purpose I/O.

Zilog also offers the Z-80 SIO, a more versatile device that provides synchronous (Bisync, HDLC and SDLC) as well as asynchronous operation.

The Z-80 DART is fabricated with n-channel silicon-gate depletion-load technology, and is packaged in a 40-pin plastic or ceramic DIP.

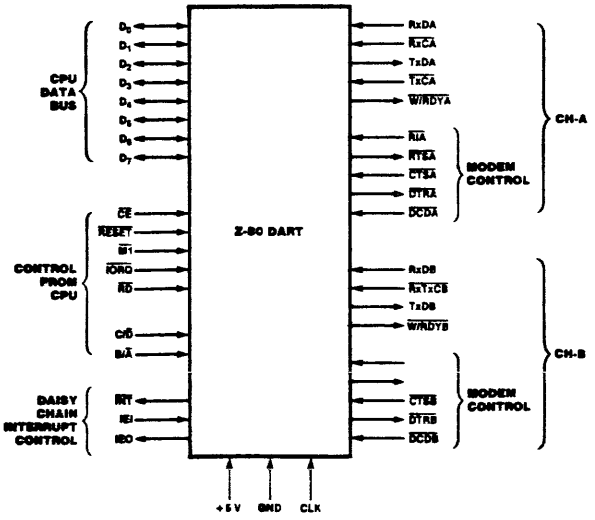


Figure 1. Z80 DART Pin Functions

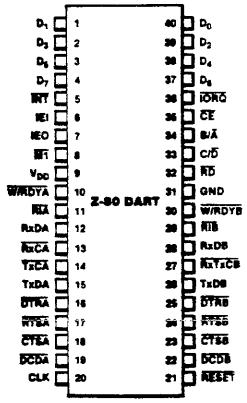


Figure 2. Pin Assignments

Pin Description

B/$\overline{A}$. *Channel A Or B Select* (input, High selects Channel B). This input defines which channel is accessed during a data transfer between the CPU and the Z-80 DART.

C/$\overline{D}$. *Control Or Data Select* (input, High selects Control). This input specifies the type of information (control or data) transferred on the data bus between the CPU and the Z-80 DART.

$\overline{CE}$. *Chip Enable* (input, active Low). A Low at this input enables the Z-80 DART to accept command or data input from the CPU during a write cycle, or to transmit data to the CPU during a read cycle.

CLK. *System Clock* (input). The Z-80 DART uses the standard Z-80 single-phase system clock to synchronize internal signals.

$\overline{CTSA}$, $\overline{CTSB}$. *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime signals.

D_0-D_7. *System Data Bus* (bidirectional, 3-state) transfers data and commands between the CPU and the Z-80 DART.

$\overline{DCDA}$, $\overline{DCDB}$. *Data Carrier Detect* (inputs, active Low). These pins function as receiver enables if the Z-80 DART is programmed for Auto Enables; otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered.

$\overline{DTRA}$, $\overline{DTRB}$. *Data Terminal Ready* (outputs, active Low). These outputs follow the state programmed into the DTR bit. They can also be programmed as general-purpose outputs.

IEI. *Interrupt Enable In* (input, active High) is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z-80 DART. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$. *Interrupt Request* (output, open drain, active Low). When the Z-80 DART is requesting an interrupt, it pulls $\overline{INT}$ Low.

$\overline{M1}$. *Machine Cycle One* (input from Z-80 CPU, active Low). When $\overline{M1}$ and $\overline{RD}$ are both active, the Z-80 CPU is fetching an instruction from memory; when $\overline{M1}$ is active while $\overline{IORQ}$ is active, the Z-80 DART accepts $\overline{M1}$ and $\overline{IORQ}$ as an interrupt acknowledge if the Z-80 DART is the highest priority device that has interrupted the Z-80 CPU.

$\overline{IORQ}$. *Input/Output Request* (input from CPU, active Low). $\overline{IORQ}$ is used in conjunction with B/$\overline{A}$, C/$\overline{D}$, $\overline{CE}$ and $\overline{RD}$ to transfer commands and data between the CPU and the Z-80 DART. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are all active, the channel selected by B/$\overline{A}$ transfers data to the CPU (a read operation). When $\overline{CE}$ and $\overline{IORQ}$ are active, but $\overline{RD}$ is inactive, the channel selected by B/$\overline{A}$ is written to by the CPU with either data or control information as specified by C/$\overline{D}$.

$\overline{RxCA}$, $\overline{RxCB}$. *Receiver Clocks* (inputs). Receive data is sampled on the rising edge of $\overline{RxC}$. The Receive Clocks may be 1, 16, 32 or 64 times the data rate.

$\overline{RD}$. *Read Cycle Status.* (input from CPU, active Low). If $\overline{RD}$ is active, a memory or I/O read operation is in progress.

RxDA, RxDB. *Receive Data* (inputs, active High).

$\overline{RESET}$. *Reset* (input, active Low). Disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts.

$\overline{RIA}$, $\overline{RIB}$. *Ring Indicator* (inputs, Active Low). These inputs are similar to $\overline{CTS}$ and $\overline{DCD}$. The Z-80 DART detects both logic level transitions and interrupts the CPU. When not used in switched-line applications, these inputs can be used as general-purpose inputs.

$\overline{RTSA}$, $\overline{RTSB}$. *Request to Send* (outputs, active Low). When the RTS bit is set, the $\overline{RTS}$ output goes Low. When the RTS bit is reset, the output goes High after the transmitter empties.

$\overline{TxCA}$, $\overline{TxCB}$. *Transmitter Clocks* (inputs). TxD changes on the falling edge of $\overline{TxC}$. The Transmitter Clocks may be 1, 16, 32 or 64 times the data rate; however, the clock multiplier for the transmitter and the receiver must be the same. The Transmit Clock inputs are Schmitt-trigger buffered. Both the Receiver and Transmitter Clocks may be driven by the Z-80 CTC Counter Time Circuit for programmable baud rate generation.

TxDA, TxDB. *Transmit Data* (outputs, active High).

$\overline{W/RDYA}$, $\overline{W/RDYB}$. *Wait/Ready* (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z-80 DART data rate. The reset state is open drain.

Functional Description

The functional capabilities of the Z-80 DART can be described from two different points of view: as a data communications device, it transmits and receives serial data, and meets the requirements of asynchronous data communications protocols; as a Z-80 family peripheral, it interacts with the Z-80 CPU and other Z-80 peripheral circuits, and shares the data, address and control buses, as well as being a part of the Z-80 interrupt structure. As a peripheral to other microprocessors, the Z-80 DART offers valuable features such as non-vectored interrupts, polling and simple handshake capability.

The first part of the following functional description introduces Z-80 DART data communications capabilities; the second part describes the interaction between the CPU and the Z-80 DART.

The Z-80 DART offers RS-232 serial communications support by providing device signals for external modem control. In addition to dual-channel Request To Send, Clear To Send, and Data Carrier Detect ports, the Z-80 DART also features a dual channel Ring Indicator (RIA, RIB) input to facilitate local/remote or station-to-station communication capability.

Communications Capabilities. The Z-80 DART provides two independent full-duplex channels for use as an asynchronous receiver/transmitter. The following is a short description of receiver/transmitter capabilities. For more details, refer to the Asynchronous Mode section of the *Z-80 SIO Technical Manual*. The Z-80 DART offers transmission and reception of five to eight bits per character, plus optional even or odd parity. The transmitter can supply one, one and a half or two stop bits per character and can provide a break output at any time. The receiver break detection logic interrupts the CPU both at the start and end of a received break. Reception is protected from spikes by a transient spike rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the Receive Data input. If the Low does not persist—as in the case of a transient—the character assembly process is not started.

Framing errors and overrun errors are detected and buffered together with the character on which they occurred. Vectored interrupts allow fast servicing of interrupting conditions using dedicated routines. Furthermore, a built-in checking process avoids interpreting a framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit is begun.

The Z-80 DART does not require symmetric Transmit and Receive Clock signals—a feature that allows it to be used with a Z-80 CTC or any other clock source. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32 or 1/64 of the clock rate supplied to the Receive and Transmit Clock inputs. When using Channel B, the bit rates for transmit and receive operations must be the same because $\overline{RxC}$ and $\overline{TxC}$ are bonded together ($\overline{RxTxCB}$).

I/O Interface Capabilities. The Z-80 DART offers the choice of Polling, Interrupt (vectored or non-vectored) and Block Transfer modes to transfer data, status and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

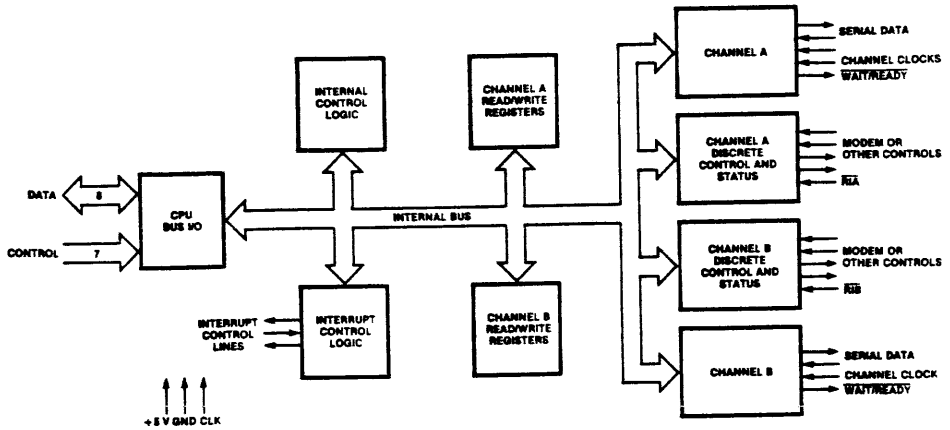


Figure 3. Block Diagram

Functional Description (Continued)

POLLING. There are no interrupts in the Polled mode. Status registers RR0 and RR1 are updated at appropriate times for each function being performed. All the interrupt modes of the Z-80 DART must be disabled to operate the device in a polled environment.

While in its Polling sequence, the CPU examines the status contained in RR0 for each channel; the RR0 status bits serve as an acknowledge to the Poll inquiry. The two RR0 status bits D_0 and D_2 indicate that a data transfer is needed. The status also indicates Error or other special status conditions (see "Z-80 DART Programming"). The Special Receive Condition status contained in RR1 does not have to be read in a Polling sequence because the status bits in RR1 are accompanied by a Receive Character Available status in RR0.

INTERRUPTS. The Z-80 DART offers an elaborate interrupt scheme that provides fast interrupt response in real-time applications. As a member of the Z-80 family, the Z-80 DART can be daisy-chained along with other Z-80 peripherals for peripheral interrupt-priority resolution. In addition, the internal interrupts of the Z-80 DART are nested to prioritize the various interrupts generated by Channels A and B. Channel B registers WR2 and RR2 contain the interrupt vector that points to an interrupt service routine in the memory. To eliminate the necessity of writing a status analysis routine, the Z-80 DART can modify the interrupt vector in RR2 so it points directly to one of eight interrupt service routines. This is done under program control by setting a program bit (WR1, D_2) in Channel B called "Status Affects Vector." When this bit is set, the interrupt vector in RR2 is modified according to the assigned priority of the various interrupting conditions.

Transmit interrupts, Receive interrupts and External/Status interrupts are the main sources of interrupts. Each interrupt source is enabled under program control with Channel A having a higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted by the transmit buffer *becoming* empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on the first received character
- Interrupt on all received characters
- Interrupt on a Special Receive condition

Interrupt On First Character is typically used with the Block Transfer mode. Interrupt On All Receive Characters can optionally modify the interrupt vector in the event of a parity error. The Special Receive Condition interrupt can occur on a character basis. The Special Receive condition can cause an interrupt only if the Interrupt On First Receive Character or Interrupt On All Receive Characters mode is selected. In Interrupt On First Receive Character, an interrupt can occur from Special Receive conditions (except Parity Error) after the first receive character interrupt (example: Receive Overrun interrupt).

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{CTS}$, $\overline{DCD}$ and $\overline{RI}$ pins; however, an External/Status interrupt is also caused by the detection of a Break sequence in the data stream. The interrupt caused by the Break sequence has a special feature that allows the Z-80 DART to interrupt when the Break sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Break condition.

CPU/DMA BLOCK TRANSFER. The Z-80 DART provides a Block Transfer mode to accommodate CPU block transfer functions and DMA block transfers (Z-80 DMA or other designs). The Block Transfer mode uses the $\overline{W/RDY}$ output in conjunction with the Wait/Ready bits of Write Register 1. The $\overline{W/RDY}$ output can be defined under software control as a Wait line in the CPU Block Transfer mode or as a Ready line in the DMA Block Transfer mode.

To a DMA controller, the Z-80 DART Ready output indicates that the Z-80 DART is ready to transfer data to or from memory. To the CPU, the Wait output indicates that the Z-80 DART is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle.

Internal Architecture

The device internal structure includes a Z-80 CPU interface, internal control and interrupt logic, and two full-duplex channels. Each channel contains read and write registers, and discrete control and status logic that provides the interface to modems or other external devices.

The read and write register group includes five 8-bit control registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B that may be read through Read Register 2 in Channel B. The registers for both channels are designated as follows:

WR0-WR5 — Write Registers 0 through 5
RR0-RR2 — Read Registers 0 through 2

The bit assignment and functional grouping of each register is configured to simplify and organize the programming process.

The logic for both channels provides formats, bit synchronization and validation for data transferred to and from the channel interface. The modem control inputs Clear to Send ($\overline{CTS}$), Data Carrier Detect ($\overline{DCD}$) and Ring Indicator ($\overline{RI}$) are monitored by the control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit and External/Status interrupts are prioritized in that order within each channel.

Data Path. The transmit and receive data path illustrated for Channel A in Figure 4 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service a Receive Character Available interrupt in a high-speed data transfer.

The transmitter has an 8-bit transmit data register that is loaded from the internal data bus, and a 9-bit transmit shift register that is loaded from the transmit data register.

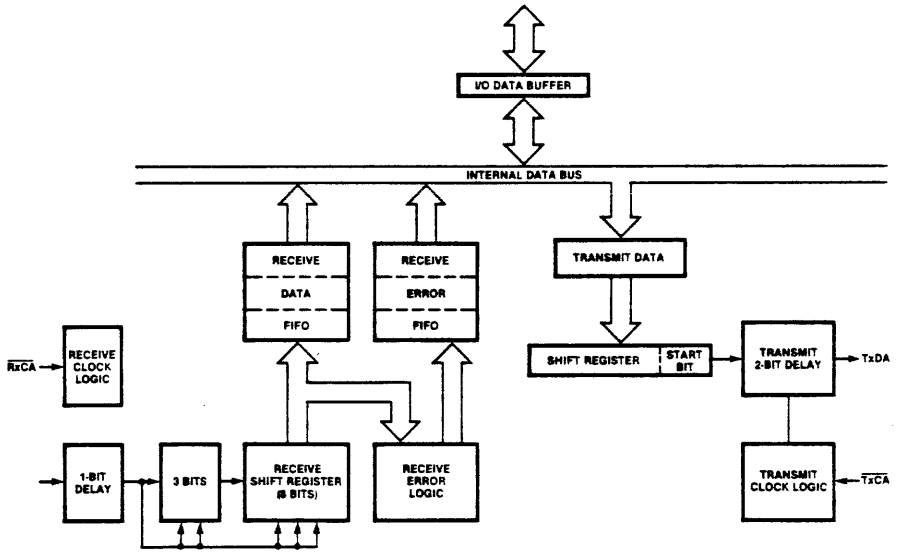


Figure 4. Data Path

Read, Write and Interrupt Timing

Read Cycle. The timing signals generated by a Z-80 CPU input instruction to read a Data or Status byte from the Z-80 DART are illustrated in Figure 5a.

Write Cycle. Figure 5b illustrates the timing and data signals generated by a Z-80 CPU output instruction to write a Data or Control byte into the Z-80 DART.

Interrupt Acknowledge Cycle. After receiving an Interrupt Request signal ($\overline{INT}$ pulled Low), the Z-80 CPU sends an Interrupt Acknowledge signal ($\overline{M1}$ and $\overline{IORQ}$ both Low). The daisy-chained interrupt circuits determine the highest priority interrupt requestor. The IEI of the highest priority peripheral is terminated High. For any peripheral that has no interrupt pending or under service, IEO = IEI. Any peripheral that does have an interrupt pending or under service forces its IEO Low.

To insure stable conditions in the daisy chain, all interrupt status signals are prevented from changing while $\overline{M1}$ is Low. When $\overline{IORQ}$ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-under-service latch.

Refer to the *Z-80 SIO Technical Manual* for additional details on the interrupt daisy chain and interrupt nesting.

Return From Interrupt Cycle. Normally, the Z-80 CPU issues an RETI (Return From Interrupt) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-under-service latch to terminate the interrupt that has just been processed.

When used with other CPUs, the Z-80 DART allows the user to return from the interrupt cycle with a special command called "Return From Interrupt" in Write Register 0 of Channel A. This command is interpreted by the Z-80 DART in exactly the same way it would interpret an RETI command on the data bus.

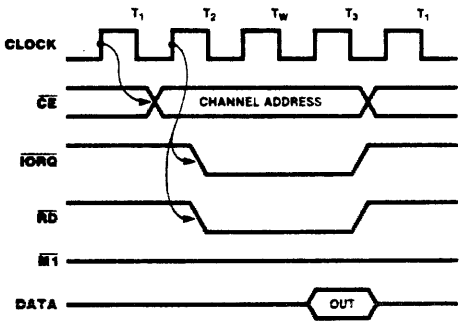


Figure 5a. Read Cycle

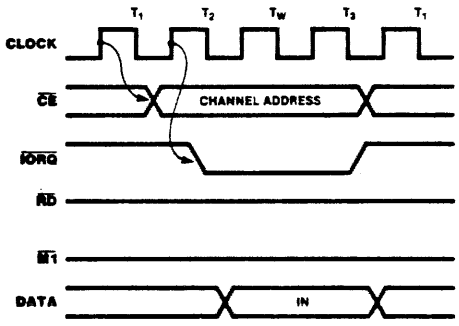


Figure 5b. Write Cycle

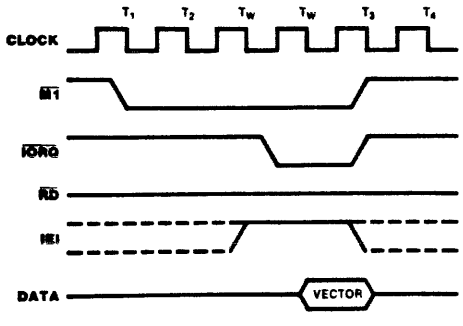


Figure 5c. Interrupt Acknowledge Cycle

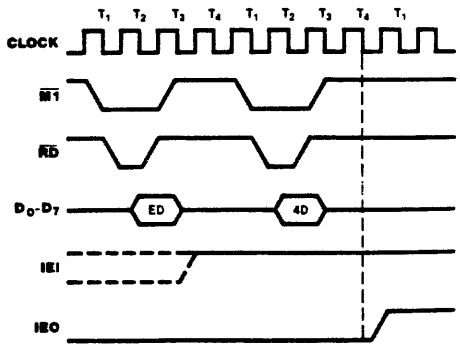


Figure 5d. Return from Interrupt Cycle

Z-80 DART Programming

To program the Z-80 DART, the system program first issues a series of commands that initialize the basic mode and then other commands that qualify conditions within the selected mode. For example, the character length, clock rate, number of stop bits, even or odd parity are first set, then the Interrupt mode and, finally, receiver or transmitter enable.

Both channels contain command registers that must be programmed via the system program prior to operation. The Channel Select input (B/$\overline{A}$) and the Control/Data input (C/$\overline{D}$) are the command structure addressing controls, and are normally controlled by the CPU address bus.

Write Registers. The Z-80 DART contains six registers (WR0-WR5) in each channel that are programmed separately by the system program to configure the functional personality of the channels (Figure 4). With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z-80 DART.

WR0 is a special case in that all the basic commands (CMD_0-CMD_2) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits D_0-D_2 to point to WR0. This means that a register cannot be pointed to in the same operation as a channel reset.

Write Register Functions	
WR0	Register pointers, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and control
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls

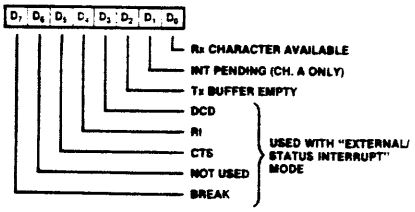
Read Registers. The Z-80 DART contains three registers (RR0-RR2) that can be read to obtain the status information for each channel (except for RR2, which applies to Channel B only). The status information includes error conditions, interrupt vector and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

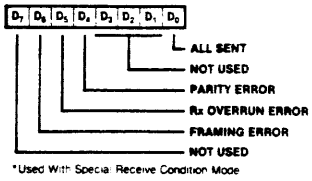
The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

Read Register Functions	
RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

READ REGISTER 0

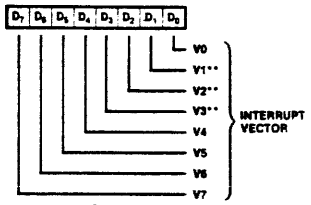


READ REGISTER 1*



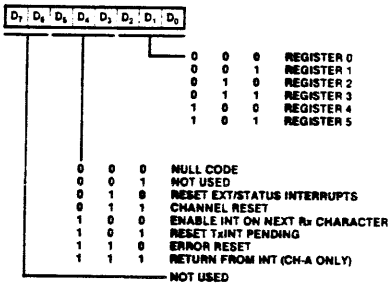
*Used With Special Receive Condition Mode

READ REGISTER 2

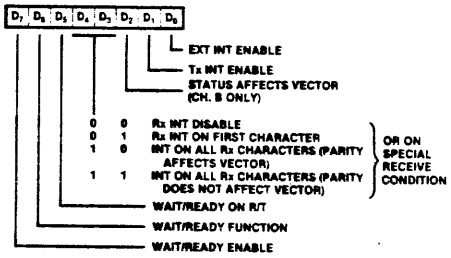


**Variable if Status Affects Vector is Programmed

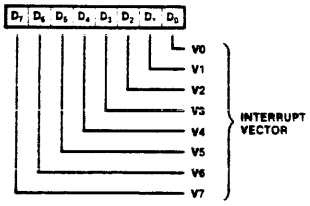
WRITE REGISTER 0



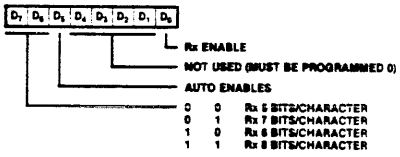
WRITE REGISTER 1



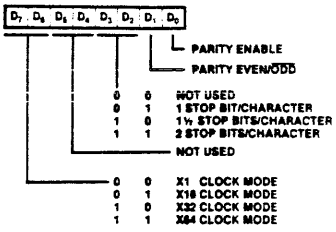
WRITE REGISTER 2 (CHANNEL B ONLY)



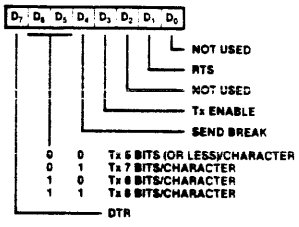
WRITE REGISTER 3



WRITE REGISTER 4



WRITE REGISTER 5



REVISIONS
SEE SHEET 7

RDOUT 13 — 7D LS Ø8 — 11 — 11 — LS Ø2 3D — 13 — 15 — 2B LS 279 — 13
QCYC 12
RPLYIN

DRVRPLY — 12 — 4A 88 81 — 13 BRECL — ARI
11 — 8C LS Ø8 — 10 DRVRPLY*

RDIN 12 — LS 32 7B — 11 — 14
RDOUT 13 — DINORDOUT

+5V — 4.7K — 9B — 5 — BBI BPOKH — 9 — 7A LS 14 — 8 — BUF POK

+5V — 4 — 2 D P Q 5 — PWRFAIL — LS74 2ØA C — 3 — 1 — LEV7RES*

22C S139 A B Ø
MOTAI7 13 — 3 — 9 — STAT*
MOTAI6 14 — 2 — 10 — Z8ØIO*
1 — 11 — EXCEP*
Ø — 12 — PROMENA*
15

27AB LS393 Y2 Qc QB
1ØMHZCLK* 1 — 5 — E1 E3 PROMOTK
4 — E2
2

MOTFC2
MOTA16 4
MOTAØ2Ø 8 — 16C LS 26Ø — 6 — 1 — 7D S11 — 12
MOTAØ8 9
MOTAØ7 10
MOTA21 11
MOTA19 1
MOTA18 2 — 16C LS 26Ø — 5
MOTA17 3
MOTAØ6 12
MOTAØ5 13
MOTA15 1
MOTA12 2 — 18C LS 26Ø — 5
MOTAØ3 3
MOTA13 12
MOTA14 13
MOTAØ4 4
MOTA11 8 — 18C LS 26Ø — 6
MOTA1Ø 9
MOTAØ9 11
1Ø
1 — 19C AS 21 — BOTTOM8 — 6
2
4
5

F138 24E
MOTA3 3
MOTA2 2
MOTA1 1
+5V 6
CONDAS* 5
11 — 13 — F 21 19C — 8 — VECTIACK*
12 — 12
13 — 1Ø
14 — 9
4 — INTACK*
9 — Z8IACK*
1Ø — BEVNTCLR*
7 — LEV7CLR*

CONDA23
CONDA22
22C S139 A B Ø
3 — 3 — 7
2 — 6
1 — 5
Ø — 4 — N/C
1

2 — 17C S32 — 3
1 — CONDA23
3 — 22A LS 04 — 4 — PGREGENA
2 — 17C S32 — 11
13 — CONDA22
12

2 — 1 — 23C LS 2Ø — 6 — REGDTK
5
4
5 — 22A LS 04 — 6 — SEGREGENA
PGREGENA*
SEGREGENA*

BUSPARITY 3
PWRFAIL 13
PARITYERR 2 — 24C LS 26Ø — 5
HALTLINE 12

MOTA23 52
MOTA22 51
9 — 25D S08 — 8 — 21 — VPA*
1Ø

MOTAS 11 — 8D LS 1Ø — 8
1Ø
9

MOTFC2 — 9 — 12D LS 1Ø — 8 — INTACK* — 2C LS 04 — 6 — INTACK
MOTFC1 — 1Ø — 11
MOTFCØ

11 — D Q — 1Ø — LS174 5B — 9 — 1ØMHZCLK*
13 — D Q — 12 — LS174 5B — 9 — Z1NT*

ONBDCYC 1Ø
Z8ØDTACK 4
PROMDTK 11 — LS 26Ø 24C — 6
REGDTK 9
BUSDTK 8

+5V — 13 — BR*
12 — BGACK*
1Ø — DTACK*

M68ØØØ
18 — RESET*
17 — MOTHLT*
9 — R/W
6 — AS* 13 — 4E LS 04 — 12

MOTRW*
13 — 8C LS Ø4 — 12 — MOTWR*
MOTAS
9 — 4A 88 81 — 8 — SRUN — AF1

DRVDATA 1Ø — LS 32 — 8 — 8E
DRVDIN 9
ONBDCYC 9 — 8 — 9E S02 — 1Ø

QCYC 5 — 7B LS 32 — 6
HAVE BUS 4

IDALxx

NMI* — 4 — 7 — 3 — 6 — BEVNT INT* 2 — 5 — LS 148 5D — 6 — 23 — IPL2*
1 — 13 — 7 — 24 — IPL1*
6 — 12 — 9 — 25 — IPLØ*
1Ø — Ø — 10 — 5

MOTA21 5Ø
MOTA2Ø 48
MOTA19 47
MOTA18 46
MOTA17 45
MOTA16 44
MOTA15 43
MOTA14 42
MOTA13 41
MOTA12 4Ø
MOTA11 39
MOTA1Ø 38
MOTAØ9 37
MOTAØ8 36
MOTAØ7 35
MOTAØ6 34
MOTAØ5 33
MOTAØ4 32
MOTAØ3 31
MOTAØ2 3Ø
MOTAØ1 29

MOTAxx

Vcc = 14,49
GND = 16,53

54 MOTD15 — 2 — 19 — 18 — BUFD15 — 14 — 11 — 15 — 15
55 MOTD14 — 3 — 17 — BUFD14 — 8 — 9 — 14
56 MOTD13 — 4 — F245 17F — 16 — BUFD13 — 7 — LS373 15H — 6 — 13
57 MOTD12 — 5 — 15 — BUFD12 — 17 — 16 — 12
58 MOTD11 — 6 — 14 — BUFD11 — 18 — 19 — 11
59 MOTD1Ø — 9 — 11 — BUFD1Ø — 3 — 2 — 1Ø
6Ø MOTDØ9 — 8 — 12 — BUFDØ9 — 13 — 12 — 9
61 MOTDØ8 — 7 — 13 — BUFDØ8 — 4 — 5 — 8
1 — 1

62 MOTDØ7 — 7 — 19 — 13 — BUFDØ7 — 4 — 11 — 5 — 7
63 MOTDØ6 — 6 — 14 — BUFDØ6 — 18 — 19 — 6
64 MOTDØ5 — 4 — F245 1ØF — 16 — BUFDØ5 — 17 — LS373 12H — 16 — 5
1 MOTDØ4 — 5 — 15 — BUFDØ4 — 13 — 12 — 4
2 MOTDØ3 — 2 — 18 — BUFDØ3 — 14 — 15 — 3
3 MOTDØ2 — 9 — 11 — BUFDØ2 — 7 — 6 — 2
4 MOTDØ1 — 3 — 17 — BUFDØ1 — 3 — 2 — 1
5 MOTDØØ — 8 — 12 — BUFDØØ — 8 — 9 — Ø
1 — 1

MOTDxx
MOTWR*

BUFDxx
ADRTIME

MOTA23 5 — 17C S32 — 6
MOTA22 4

+5V — 9B DIP — 4.7K — 2 — 7 — API — 1 — 8A 86 4Ø — 2 — BUFHLINE — 11 — LEV7RES* — LS74 2ØA C — 13 — 9 — HALTLINE
8A

+5V — 1Ø — 12 — D P Q — LS74 8B — Q — 8 — 13 — BEVNTL — 12 — 8A 86 4Ø — 13 — 11 — BRI
BEVNTCLR* 12 — 24B LS 32 — 11
AS* 13

14 — D Q — 15 BEVNTINT* — LS174 5B — 9 — 1ØMHZCLK*

BPI — BIRQ7* 9 — 8A 86 4Ø — 14 — 5
ABI — 6* — 14 — 12 — 5A
AAI — 5* — 13 — 13
AL2 — 4* — 3 — 4
5A
1ØMHZCLK* 9

5C LS175 — CLR — 1 — +5V
D

+5V — 59 60 61 62 — E58 E63 — R1 1ØKΩ — 7A LS 14 — 3 — 4 — BAI — BDCOKH
C1 4.7 MFD

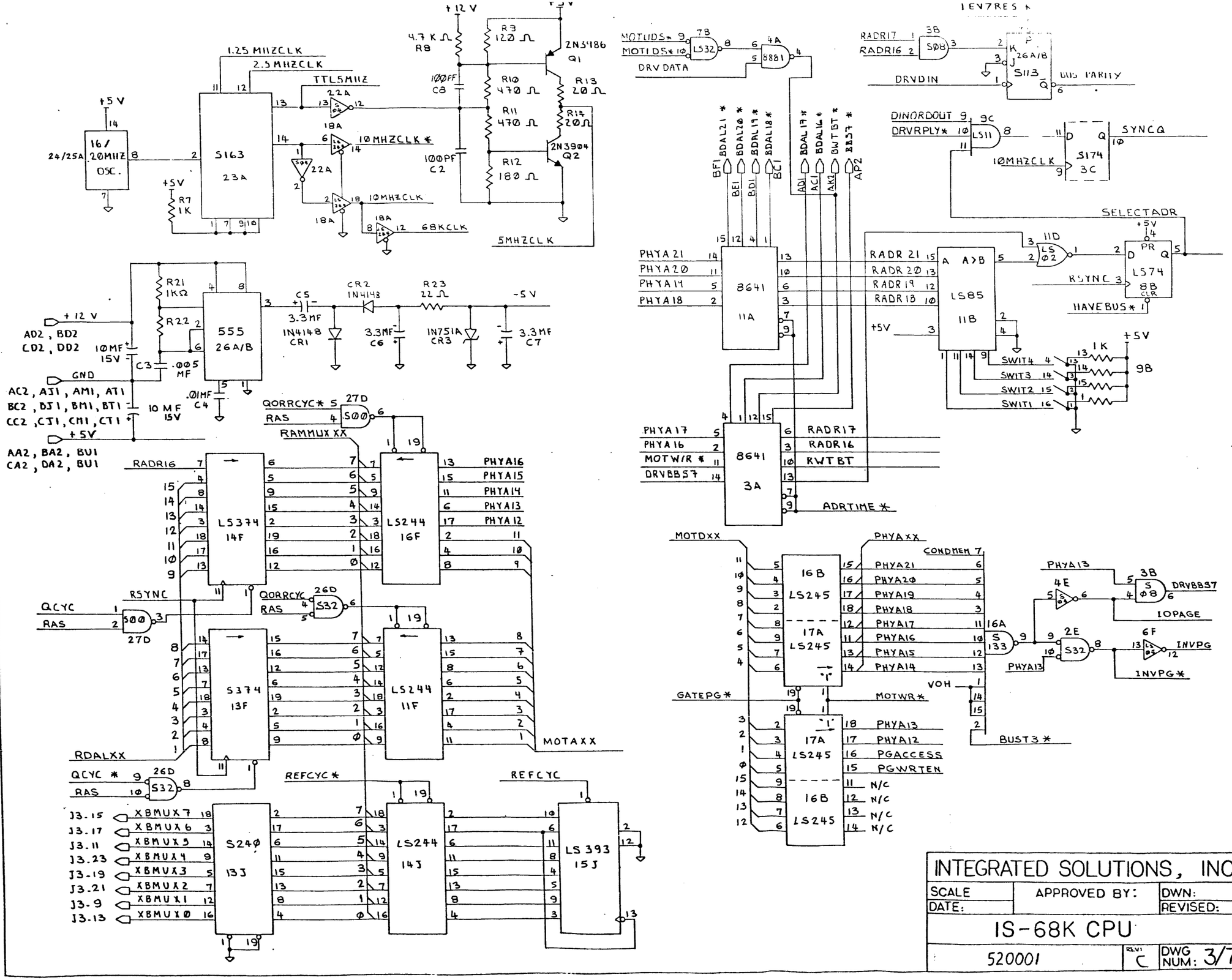
+5V — R18 33Ø — DS1 LED
+5V — 9B DIP — 4.7K — 7A LS 14 — 13 — 12 — 11 — 7A LS 14 — 1Ø
6A 88 81 — 11 — 12 — 13 — 6 — MOTHLT*
6A 88 81 — 2 — 3 — 1 — 7 — RESET*

AT2 — BINITL — 1Ø — 6A 88 81 — 9 — 8 — 7A LS 14 — 2 — 1

INTEGRATED SOLUTIONS, INC.		
SCALE	APPROVED BY:	DWN:
DATE		REVISED:
IS-68K CPU		
520001	REV: C	DWG NUM: 1/7

RRPLY
2B LS 279
INTACK *
6B LSØØ
BUST2
HAVEBUS
VOH
1ØMHZCLK *
4D LS Ø8
OFFBDREQ
1ØMHZCLK
S74 6D
MOTAS
NOACCESS*
INVPG *
S11 7D
4B S174 CLR
INTACK
4A 8881
+5V
R2 18ØΩ
R3 39ØΩ
AN2
BIAKO *
BUFDxx
xBUFDI5 J4-4
xBUFDI4 J4-6
xBUFDI3 J4-9
xBUFDI2 J4-11
xBUFDI1 J4-7
xBUFDIØ J4-1
xBUFDØ9 J4-2
xBUFDØ8 J4-12
xBUFDØ7 J4-19
xBUFDØ6 J4-22
xBUFDØ5 J4-24
xBUFDØ4 J4-21
xBUFDØ3 J4-28
xBUFDØ2 J4-16
xBUFDØ1 J4-26
xBUFDØØ J4-17
J4-3,5,8,1Ø,13,15,18,2Ø,23,25,27,29,31,33 GND
BUST4
SØ8 3B
MOTW/R*
DRVDIN
7B S 32
6C SØ2
DRVCLR*
ADRTIME
ADRTIME *
4E
MOTR/W*
3B SØ8
S174 3C
S174 2D
2C
1B LSØ4
BUST3*
BDAL15* BV2
BDAL14* BU2
BDAL13* BT2
BDAL12* BS2
BDAL11* BR2
BDAL1Ø* BP2
BDALØ9* BN2
BDALØ8* BM2
RDALxx
BUFDxx
8641 14A
15A
14H LS244
8641 13A
13H LS244
8641 12A
AU2 BDALØØ*
AV2 BDALØ1*
BE2 BDALØ2*
BF2 BDALØ3*
BDALØ4* BH2
BDALØ5* BJ2
BDALØ6* BK2
BDALØ7* BL2
A32 BSYNC *
AH2 BDIN *
AE2 BDOUT *
AF2 BRPLY *
8641 2A
RSYNC
RDIN
RDOUT
DRVRPLY
RRPLY
HAVEBUS
1C SØ8
GATEBUS
LSØ4
2E LS 32
6E LSØØ
4D LS Ø8
2D S174 CLR
1ØMHZ CLK*
VOH
BERR*
BUSDTK
4C LS 51
7C LS 51
HAVEBUS
DRVDIN
QCYC
RDOUT
ADRTIME
DRVDATA
S32 2E
SELECTADR
RDIN
MOTWR*
BUST2
6E LSØØ
LS 279 2B
DRVCLR*
PARINHI
PARINLO
PARITYENA
MUXRW *
9E SØ2
8D LS 1Ø
ONBDCYC
S113 9A
PARITY ERR
N/C
LEV7CLR*
AS*
24B LS 32
25B LSØ8
RESET*
LEV7RES*
11D SØ2
MUXWR*
QCYC
CAS
RPLYIN
PHYA15
PHYA14
PHYA13
PHYA12
MOTA11
MOTAIØ
MOTAØ9
MOTAØ8
ADRTIME *
15F LS244
MOTAØ7
MOTAØ6
MOTAØ5
MOTAØ4
MOTAØ3
MOTAØ2
MOTAØ1
MOTLDS*
12F LS244
IDALXX

INTEGRATED SOLUTIONS, INC.		
SCALE:	APPROVED BY:	DWN:
DATE:		REVISED:
IS-68K CPU		
520001	REV: C	DWG NUM: 2/7



INTEGRATED SOLUTIONS, INC.		
SCALE	APPROVED BY:	DWN:
DATE:		REVISED:
IS-68K CPU		
520001	REV: C	DWG NUM: 3/7

22D
LS174
MOTD09 MOTD08 MOTD07 MOTD06 MOTD05 MOTD04 WREXCEP*
D1 D2 D3 D4 D5 D6
Q1 Q2 Q3 Q4 Q5 Q6
N/C N/C
+5V
20D LS244
MOTD07 MOTD06 MOTD05 MOTD04 RDEXCEP*
MOTFC2
SEGREGENA
LS257 21D
S 3A 4A 2A 1A 1B 2B 3B 4B
3Y 4Y 2Y 1Y
PGREGENA* 3F LS 32 GATEPG*
SEGREGENA* 3F LS 32 GATESEG*
MOTR/W* MOTLDS* 3D LS Ø2 WLDS
SEGREGENA 21A SØØ
MOTA15 MOTA14 MOTA13 MOTA12 SEGREGENA*
20D VZLS 244
MOTA21 MOTA2Ø MOTA19 MOTA18 MOTA17 MOTA16 PROTBITØ PROTBIT1
2149H-2 20/21C WE CS 1Ø
2ØC 21C
PGREGENA
MOTDXX
LS245 17B
GATESEG* MOTWR*
PGWR*
1ØC LS 26Ø LOCALOB
ES3 ES2 ES1 ES0 ES49
MOTA15 MOTA14 MOTA13 MOTA12
2ØB CS
PHYA18 PHYA19 PHYA2Ø PHYA21
PHYA17 PHYA16 PHYA15 PHYA14
18B CS
QORRCYC* SØ8 RAS*
RAS CAS REFCYC* MUXA17*
25E S24Ø
XBRAS* J3-7
XCAS* J3-3
XBREF J3-5
XBD17 J3-1
WRH* WRL* PARINH PARINLO
9F S24Ø
XBWRH J3-34
XBWRL J3-33
XBPINH* J4-32
XBPINL* J4-34
+5V
PHYA13 PHYA12
LS175 19A
18A
22A
PGACCESS
MOTRW*
PGWRTEN
PGREGENA
PGWR* 25D
UPDATE*
HALFCYC ONBDCYC BUST3 BUST4
4C LS51
2C
LS367 21B
MOTA21 MOTA2Ø MOTA19 MOTA18 MOTA17 MOTA16
PGREGENA*

INTEGRATED SOLUTIONS,INC.		
SCALE	APPROVED BY:	DWN:
DATE		REVISED:
IS-68K CPU		
520001	REV: C	DWG NUM: 4/7

PHYA21 PHYA20 PHYA19 PHYA18 VOH IOPAGE
S85 IØA A>B
SWIT1 SWIT2 SWIT3 SWIT4
AGTB
CONDAS* MOTAS S174 2D IØMHZCLK* VOH
VECTIACK* 6B
CONDMEM 6C
INTACK* 6E
CONDA23 CONDA22 11D LS ØØ
GATEBUS 3D LS Ø2
ONBDCYC* 9E
INVP6* 7D S11
PROTVIOL*
VECTIACK* FC2 FC1 FCØ PROTBIT1 PROTBITØ MOTWR*
256x4 PROM 9D
PROTVIOL NOACCESS*
1.25MHZCLK
1/2 LS393 27A/B
VOH S113 26A/B
REFCYC* IØ
.15MHZCLK
SYNCQ DRVRPLY* IØ LS 20 23C
REFREQ
RAS* 26B/C
AGTB CONDMEM 27B/C
IØMHZCLK*
7E LS 393
1.25MHZCLK
E4 E5 E6 E7
AS* OFFBDREQ
TIMEOUT
BERR*
BN1 BSACK* 8A
5F LS 393
.15MHZCLK
6F LS Ø4
+E
5E S113
IØMHZCLK*
+5V
S174 5B
HAVEBUS*
HAVEBUS
3D LS Ø2
LSØ4 8C
2B LS 279
S174 3C
LS174 5B
RSYNC
2E S32
2B LS 279
IØMHZCLK
RAS
27E DDU-6-5Ø
IØ 4Ø 5Ø
N/C CAS
27C/D F113
ONBDCYC*
ONBDCYC
27D FØØ
27B/C S02
QORRCYC*
REFCYC*
REFCYC
26D S32
QCYC*
QCYC
68KCLK
S113 26C/D
R4 18Ø Ω
R5 39Ø Ω
QDMGD* AS2

RADR17 RADR1Ø RADR21 RADR2Ø RADR19 RADR18
+5V CLR
LS 174 11C
LADR2Ø LADR19 LADR18 LADR21
RSYNC
RDOUT RWTBT
9C LS11
PHYA17 RDIN MOTRW* MOTUDS* MOTLDS*
LS158 12C
MUXA17* REFCYC*
MUXWR* 8C
MUXRW*
WRH* WRL*
12D LS IØ
26D S32 F ØØ 27D
RAS
26E S37
22 Ω RP6 RAS1* RAS1Ø*
RAS*
S174 3C
IØMHZCLK
HALFCYC
1C S Ø8
LSØ4
8E S 32
RESET*
3E LS74
+5V
IØMHZCLK*
ONBDCYC* PROTVIOL*
CPUBDRAM
CAS
22 Ω RP6 CASØ* CAS1*

XBAD21 XBAD2Ø XBAD19 XBAD18
J3-25 J3-31 J3-29 J3-27
S157 1ØD
PHYA21 PHYA2Ø PHYA19 PHYA18
56 E57 E54 E55 E53
1ØC LS 26
QCYC LOCALOB ONBDCYC
7C LS 51
8C LS Ø4

J3-2,4,6,8,1Ø,12,14,
16,2Ø,22,24,26,28,
3Ø,32,34 GND

INTEGRATED SOLUTIONS,INC.

SCALE:	APPROVED BY:	DWN:
DATE:		REVISED:

IS-68K CPU

52ØØØ1	REV C	DWG NUM: 5/7

Page missing from original document

ECN	REV	DESCRIPTION	DATE	APPRD
261	B	MISC. CHANGES	4-3-84	
538	C	ADDED NOTE FOR R1 "ID AND "RP13" #26	12-1-84	

2.5 MHZCLK E30 E28 E29 1.25 MHZCLK

MOTDXX

Z8Ø CTC#Ø 2Ø:23E Vcc=24 GND=5

MOTA2 MOTA1 CTCØ* Z8ØM1* Z8ØRD* Z8ØIORQ* RESET* 5MHZCLK

+5V 2.2 K R6 ZINT*

Z8Ø SIOØ #Ø Vcc=9 GND=31 2Ø-24F

MOTA2 33 MOTA1 34 SIOØ* 35 Z8ØM1* 8 Z8ØRD* 32 Z8ØIORQ* 36 RESET* 21 5MHZCLK 2Ø

+5V E44 E43 E45 E46 E47 E48 -5V

25/26F 26LS3Ø 27PF C2 C3 C4 N/C 26/27F

TXDØ J2-Ø1
DTRØ/TXDØRET J2-Ø8
RTSØ J2-Ø3
N/C
RTSØRET J2-Ø5

26LS32 19F E8 E9 E10 E11 E16 E17 E18

CTSØ/CLKØ J2-Ø4
CLKØRET J2-Ø9

RXDØ J2-Ø2
RXDØRET J2-1Ø

E19 E2Ø E21 E49

DCDØ J2-Ø7
CHØRET/DCDØ RET J2-Ø6

25/26F 27H 26LS3Ø N/C

J2-13 J2-18 J2-11 J2-15

+5V E37 E38 E39 E4Ø E41 E42 -5V

E12 E13 E14 E15 E22 E24 E23 E25 E27 E26 18F

CTS1/CLK1 J2-14
CLK1RET J2-19
RXD1 J2-12
RXD1RET J2-2Ø
DCD1 J2-17
CH1RET J2-16

PROMENA* MOTWR* MOTAXX

+5V E32 E31 E33 E35 E34 E36 2716/32 16Ø/128 VCC=28 GND=14 16/17 D/E 1716/32 16Ø/128 VCC=28 GND=14 18/19 D/E

MOTDXX

TIMEOUT INUPG PROTVIOL PARITYENA LS157 1ØE S174 11E 12E RESET* RDEXCEP*

EXCEP* MOTWR* 25C LS32 WREXCEP* LS04 WLDS LS00 6E BERR* 6F