

```
cordic.scfix.cpp
//*****
// Test of SystemC Fixpoint CORDIC
//
// Licensing:
// This code is distributed under GNU LGPL license.
//
// Modified:
// 2012.07.05
//
// Author:
// Based on SCLive 3.0 and www.asic-world.com example codes
//
// Modifications by Young W. Lim
//
//*****

#define SC_INCLUDE_FX

#include <systemc.h>
#include <iostream>
#include <iomanip>

#include "cordic.hpp"

int sc_main(int argc, char * argv[]) {

    double pi = 3.141592653589793;
    double K = 1.646760258121;
    int nIter = 10;

    double x, y, z;

    //-----
    // printf ("\nGrinding on [K, 0, 0]\n");
    // Circular (X0C, 0L, 0L);
    //-----
    x = 1 / K;
    y = 0.0;
    z = 0.0;
    cout << "-----\n"
         << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

    cordic(&x, &y, &z, nIter);

    cout << "xo=" << x << " yo=" << y << " zo=" << z << "\n";

    //-----
    // printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
    // Circular (X0C, 0L, HalfPi / 3L);
    //-----
    x = 1 / K;
    y = 0.0;
    z = pi / 6.0;
    cout << "-----\n"
         << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

    cordic(&x, &y, &z, nIter);

    cout << "xo=" << x << " yo=" << y << " zo=" << z << "\n";

    //-----
```

```

// printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
// Circular (X0C, 0L, HalfPi / 2L);
//-----
x = 1 / K;
y = 0.0;
z = pi / 4.0;
cout << "-----\n"
      << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

cordic(&x, &y, &z, nIter);

cout << "xo=" << x << " yo=" << y << " zo=" << z << "\n";

```

```

//-----
// printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
// Circular (X0C, 0L, 2L * (HalfPi / 3L));
//-----
x = 1 / K;
y = 0.0;
z = pi / 3.0;
cout << "-----\n"
      << "xi=" << x << " yi=" << y << " zi=" << z << "\n";

cordic(&x, &y, &z, nIter);

cout << "xo=" << x << " yo=" << y << " zo=" << z << "\n";

```

```

return (0);

```

```

}
:::::::::::::
cordic.cpp
:::::::::::::
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <cmath>
# include <ctime>

using namespace std;

# include "cordic.hpp"

#define SC_INCLUDE_FX

#include <systemc.h>

#define FIXPT
#define WL 32
#define IWL 15

sc_fxtype_params param1(32, 15);
sc_fxtype_context c1(param1);

//*****80

void cordic ( double *x, double *y, double *z, int n )

```

```

//*****80
// CORDIC returns the sine and cosine using the CORDIC method.
//
// Licensing:
//
// This code is distributed under the GNU LGPL license.
//
// Modified:
//
// 2012.04.17
//
// Author:
//
// Based on MATLAB code in a Wikipedia article.
//
// Modifications by John Burkardt
//
// Further modified by Young W. Lim
//
// Parameters:
//
// Input:
// *x: x coord of an init vector
// *y: y coord of an init vector
// *z: angle (-90 <= angle <= +90)
// n: number of iteration
// A value of 10 is low. Good accuracy is achieved
// with 20 or more iterations.
//
// Output:
// *xo: x coord of a final vector
// *yo: y coord of a final vector
// *zo: angle residue
//
// Local Parameters:
//
// Local, real ANGLES(60) = arctan ( (1/2)^(0:59) );
//
// Local, real KPROD(33), KPROD(j) = product ( 0 <= i <= j ) K(i),
// K(i) = 1 / sqrt ( 1 + (1/2)^(2i) ).
//
//
//
{
#define ANGLES_LENGTH 60
#define KPROD_LENGTH 33

#ifdef FIXPT
// sc_fixed<WL, IWL> angle;
// sc_fixed<WL, IWL> angles[ANGLES_LENGTH] = {

sc_fix angle;
sc_fix angles[ANGLES_LENGTH] = {
#else
double angle;
double angles[ANGLES_LENGTH] = {
#endif
7.8539816339744830962E-01,
4.6364760900080611621E-01,
2.4497866312686415417E-01,
1.2435499454676143503E-01,
6.2418809995957348474E-02,
3.1239833430268276254E-02,
1.5623728620476830803E-02,
7.8123410601011112965E-03,
3.9062301319669718276E-03,
1.9531225164788186851E-03,
9.7656218955931943040E-04,
4.8828121119489827547E-04,
2.4414062014936176402E-04,

```

```
1.2207031189367020424E-04,  
6.1035156174208775022E-05,  
3.0517578115526096862E-05,  
1.5258789061315762107E-05,  
7.6293945311019702634E-06,  
3.8146972656064962829E-06,  
1.9073486328101870354E-06,  
9.5367431640596087942E-07,  
4.7683715820308885993E-07,  
2.3841857910155798249E-07,  
1.1920928955078068531E-07,  
5.9604644775390554414E-08,  
2.9802322387695303677E-08,  
1.4901161193847655147E-08,  
7.4505805969238279871E-09,  
3.7252902984619140453E-09,  
1.8626451492309570291E-09,  
9.3132257461547851536E-10,  
4.6566128730773925778E-10,  
2.3283064365386962890E-10,  
1.1641532182693481445E-10,  
5.8207660913467407226E-11,  
2.9103830456733703613E-11,  
1.4551915228366851807E-11,  
7.2759576141834259033E-12,  
3.6379788070917129517E-12,  
1.8189894035458564758E-12,  
9.0949470177292823792E-13,  
4.5474735088646411896E-13,  
2.2737367544323205948E-13,  
1.1368683772161602974E-13,  
5.6843418860808014870E-14,  
2.8421709430404007435E-14,  
1.4210854715202003717E-14,  
7.1054273576010018587E-15,  
3.5527136788005009294E-15,  
1.7763568394002504647E-15,  
8.8817841970012523234E-16,  
4.4408920985006261617E-16,  
2.2204460492503130808E-16,  
1.1102230246251565404E-16,  
5.5511151231257827021E-17,  
2.7755575615628913511E-17,  
1.3877787807814456755E-17,  
6.9388939039072283776E-18,  
3.4694469519536141888E-18,  
1.7347234759768070944E-18 };
```

```
int j;
```

```
#ifdef FIXPT  
// sc_fixed <WL, IWL> factor;  
// sc_fixed <WL, IWL> kprod[KPROD_LENGTH] = {  
  
sc_fix factor;  
sc_fix kprod[KPROD_LENGTH] = {  
#else  
double factor;  
double kprod[KPROD_LENGTH] = {  
#endif  
0.70710678118654752440,  
0.63245553203367586640,  
0.61357199107789634961,  
0.60883391251775242102,  
0.60764825625616820093,  
0.60735177014129595905,  
0.60727764409352599905,  
0.60725911229889273006,  
0.60725447933256232972,  
0.60725332108987516334,
```

```
0.60725303152913433540,  
0.60725295913894481363,  
0.60725294104139716351,  
0.60725293651701023413,  
0.60725293538591350073,  
0.60725293510313931731,  
0.60725293503244577146,  
0.60725293501477238499,  
0.60725293501035403837,  
0.60725293500924945172,  
0.60725293500897330506,  
0.60725293500890426839,  
0.60725293500888700922,  
0.60725293500888269443,  
0.60725293500888161574,  
0.60725293500888134606,  
0.60725293500888127864,  
0.60725293500888126179,  
0.60725293500888125757,  
0.60725293500888125652,  
0.60725293500888125626,  
0.60725293500888125619,  
0.60725293500888125617 };
```

```
#ifndef FIXPT
```

```
// sc_fixed<WL, IWL> pi = 3.141592653589793;  
// sc_fixed<WL, IWL> poweroftwo;  
// sc_fixed<WL, IWL> sigma;  
// sc_fixed<WL, IWL> sign_factor;  
// sc_fixed<WL, IWL> theta;  
//  
// sc_fixed<WL, IWL> xn, yn;
```

```
sc_fix pi = 3.141592653589793;  
sc_fix poweroftwo;  
sc_fix sigma;  
sc_fix sign_factor;  
sc_fix theta;
```

```
sc_fix xn, yn;
```

```
#else
```

```
double pi = 3.141592653589793;  
double poweroftwo;  
double sigma;  
double sign_factor;  
double theta;
```

```
double xn, yn;
```

```
#endif
```

```
//  
// Initialize loop variables:  
//
```

```
theta = *z;
```

```
xn = *x;
```

```
yn = *y;
```

```
poweroftwo = 1.0;
```

```
angle = angles[0];
```

```
//  
// Iterations  
//
```

```
for ( j = 1; j <= n; j++ )  
{  
    if ( theta < 0.0 )  
    {  
        sigma = -1.0;
```

```

}
else
{
    sigma = 1.0;
}

factor = sigma * poweroftwo;

*x =      xn - factor * yn;
*y = factor * xn +      yn;

xn = *x;
yn = *y;

//
// Update the remaining angle.
//
theta = theta - sigma * angle;

poweroftwo = poweroftwo / 2.0;

//
// Update the angle from table, or eventually by just dividing by two.
//
if ( ANGLES_LENGTH < j + 1 )
{
    angle = angle / 2.0;
}
else
{
    angle = angles[j];
}

*z = theta;
}
//
// Adjust length of output vector to be [cos(beta), sin(beta)]
//
// KPROD is essentially constant after a certain point, so if N is
// large, just take the last available value.
//
// if ( 0 < n )
// {
//     *c = *c * kprod [ i4_min ( n, KPROD_LENGTH ) - 1 ];
//     *s = *s * kprod [ i4_min ( n, KPROD_LENGTH ) - 1 ];
// }
//
// Adjust for possible sign change because angle was originally
// not in quadrant 1 or 4.
//
// *c = sign_factor * *c;
// *s = sign_factor * *s;

return;
# undef ANGLES_LENGTH
# undef KPROD_LENGTH
}

```