```
::::::::::::::
Core.make
::::::::::::::
#-----------------------------------------------------------------
# copy include files    ${INC} into the directory ${INCD}
# copy library files    ${LIB} into the directory ${LIBD}
# copy executable files ${EXE} into the directory ${EXED}
# include files in ${INCS} directories to compile this module
#-----------------------------------------------------------------
INCD = /home/young/MyWork/inc
LIBD = /home/young/MyWork/lib
EXED = /home/young/MyWork/exe


BurkDir = /home/young/MyWork/2.cordic_cpp/Burkadt
GHDLDir = /home/young/MyWork/7.cordic_accuracy/IF.GHDL

VPATH = ${BurkDir}:${GHDLDir}        \

INCS =  -I${BurkDir} -I${GHDLDir}    \

.SUFFIXES : .o .cpp .c

.cpp.o :
        g++ -c -Wall -g ${INCS} $<

.c.o :
        g++ -c -Wall -g ${INCS} $<


#-----------------------------------------------------------------
# Classes
#-----------------------------------------------------------------
SRC = Core.hpp                        \
      Core.cpp                        \
      Core.1.fptr1.cordic_org.cpp     \
      Core.1.fptr2.cordic_burk.cpp    \
      Core.1.fptr3.cordic_vhdl.cpp    \
      Core.2.wrap1.cordic_stat.cpp    \
      Core.2.wrap2.cordic_break.cpp   \

OBJ = Core.o                          \
      Core.1.fptr1.cordic_org.o       \
      Core.1.fptr2.cordic_burk.o      \
      Core.1.fptr3.cordic_vhdl.o      \
      Core.2.wrap1.cordic_stat.o      \
      Core.2.wrap2.cordic_break.o     \


INC = Core.hpp                        \
```

```
LIB = libcordic-core.a                \

EXE = Core_tb                         \



#---------------------------------------------------------------
Core.o : cordic-burk cordic-ghdl ${SRC}
        g++ -c -Wall -g ${INCS} Core.cpp

cordic-burk :
        cd ${BurkDir}; make all;
        # cordic-burk library to ${LIBD}

cordic-ghdl :
        cd ${GHDLDir}; make all;
        # cordic-ghdl library to ${LIBD}
        # cordic_vtb executable to ${EXED}


#---------------------------------------------------------------
all : ${OBJ} cordic-burk cordic-ghdl
#       ar -rcs libcordic-core.a cordic_core.o
        ar -cvq libcordic-core.a ${OBJ}
        \cp -f ${LIB} ${LIBD}
        \cp -f ${INC} ${INCD}
        \rm -f ${OBJ}

print : Core.make Core_tb.cpp ${SRC}
        /bin/more $? > Core.print

tar : Core.make Core_tb.cpp ${SRC}
        tar cvf Core.tar $?

clean :
        \rm -f *.o *~ *#







::::::::::::::
Core_tb.cpp
::::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
```

```
#include <iomanip>
#include <fstream>

using namespace std;

#include "Core.hpp"
#include "Core_tb.hpp"


//-----------------------------------------------------------------------
//   Purpose:
//
//     Test various cordic implementations
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.03.27
//
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
//
// Core_tb.cpp
// Core_tb.wrap1.cpp
// Core_tb.wrap2.cpp
//
//-----------------------------------------------------------------------



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

  int nIters = 10;
  double x, y, z;


  //.............................................
  Core C;
```

```
//.............................................

char path[32] = "";
int nBreak =0;

C.setPath(path);
C.setLevel(nIters);
// C.setThreshold(threshold);
C.setNBreak(nBreak);

C.setUseTh(0);
C.setUseThDisp(0);
C.setUseATAN(0);

C.setMode(1);

C.dispVars();


//.............................................

double pi = C.getPi();
double K  = C.getK();


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

printf ("\nGrinding on [K, 0, 0]\n");
cout << "-----------------------------------------------\n";
printf("xi= %f yi= %f zi= %f \n", x, y, z);

C.cordic(&x, &y, &z);

printf("xo= %f yo= %f zo= %f \n", x, y, z);


//---------------------------------------------------------------------------
// 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;
```

```
  printf ("\nGrinding on [K, 0, pi/6] -> [0.86602540, 0.50000000, 0]\n");
  cout << "-------------------------------------------\n";
  printf("xi= %f yi= %f zi= %f \n", x, y, z);

  C.cordic(&x, &y, &z);

  printf("xo= %f yo= %f zo= %f \n", x, y, z);




  //-------------------------------------------------------------------------
  // 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;

  printf ("\nGrinding on [K, 0, pi/4] -> [0.70710678, 0.70710678, 0]\n");
  cout << "-------------------------------------------\n";
  printf("xi= %f yi= %f zi= %f \n", x, y, z);

  C.cordic(&x, &y, &z);

  printf("xo= %f yo= %f zo= %f \n", x, y, z);



  //-------------------------------------------------------------------------
  // 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;

  printf ("\nGrinding on [K, 0, pi/3] -> [0.50000000, 0.86602540, 0]\n");
  cout << "-------------------------------------------\n";
  printf("xi= %f yi= %f zi= %f \n", x, y, z);

  C.cordic(&x, &y, &z);

  printf("xo= %f yo= %f zo= %f \n", x, y, z);


  return 0;

}
::::::::::::::
```

Core.hpp
::::::::::::::
```
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <ctime>
#include <string.h>


using namespace std;


const int ANGLES_LENGTH =60;
const int KPROD_LENGTH =33;

//-----------------------------------------------------------------------
//   Purpose:
//
//     Class Core Interface Files
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.14
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
// Mode                setMode(),              getMode()
// UseTh               setUseTh(),             getUseTh()
// UseThDisp           setUseThDisp()          getUseThDisp()
// UseATAN             setUseATAN()            getUseATAN()
//
// Level               setLevel(),             getLevel()
// Path                setPath(),              getPath()
// Threshold           setThreshold(),         getThreshold()
// nBreak;             setNBreak(),            getNBreak()
// nBreakInit;         setNBreakInit(),        getNBreakInit()
//
```

```
// Pi                   setPi(),               getPi()
// K                    setK(),                getK()
// Angles               setAngles()
// KProd                setKProd()
//
//                      dispVars()
//                      initAcc()
//
//                      cordic_fptr()
//                      setFuncPtr()
//                      initScale()
//
//                      cordic()
//                      cordic_stat()
//                      cordic_break()
//
// zz;
// xx, sum_xx, sum_xx2;   //    SCE,        SSE,        SRE;
// yy, sum_yy, sum_yy2;   //   sSCE,       sSSE,       sSRE;
// sum_xx_n, sum_xx2_n;   //   mSCE,       mSSE,       mSRE;
// sum_yy_n, sum_yy2_n;   //  rmSCE,      rmSSE,      rmSRE;
// max_err, max_errn;     // minSCE,     minSSE,     minSRE;
// cnt_xx, cnt_yy;        // maxSCE,     maxSSE,     maxSRE;
//-----------------------------------------------------------------------------



class Core;
//-------------------------------------------------------------
// used via a pointer to a function  (friend functions)
//-------------------------------------------------------------
void    cordic_org  ( double *x, double *y, double *z, Core *C );
void    cordic_burk ( double *x, double *y, double *z, Core *C  );
void    cordic_vhdl ( double *x, double *y, double *z, Core *C  );



class Core
{

public:

  Core();
  ~Core();


  void    setMode(int m);
  void    setUseTh(int flag);
  void    setUseThDisp(int flag);
  void    setUseATAN(int flag);
```

```
int     getMode();
int     getUseTh();
int     getUseThDisp();
int     getUseATAN();


// ---------------------------------------------------------------
// level      : Number of Iteration = Height of binary angle tree
// path       : path string in the binary angle tree
// threshold  : threshold for breaking the cordic algorithm's loop
// nBreak     : number of such breaking events
// nBreakInit : initialize the nBreak counter
// ---------------------------------------------------------------
void    setLevel(int l);
void    setPath(char *p);
void    setThreshold(double th);
void    setNBreak(int nB);
void    setNBreakInit(int nBInit);

int     getLevel();
void    getPath(char *p);
double  getThreshold();
int     getNBreak();
int     getNBreakInit();


//-----------------------------------------------
void    setPi();
void    setK();
void    setAngles();
void    setKprod();

double  getPi();
double  getK();


//-----------------------------------------------
double *getAngles();
double *getKprod();

//-----------------------------------------------
void    dispVars();

void    initAcc () ;



//---------------------------------------------------------------
// used via a pointer to a function
```

```
  //-------------------------------------------------------------
  friend void    cordic_org  ( double *x, double *y, double *z, Core *C );
  friend void    cordic_burk ( double *x, double *y, double *z, Core *C );
  friend void    cordic_vhdl ( double *x, double *y, double *z, Core *C );

  //-------------------------------------------------------------
  // mode = 1: cordic_fptr = & cordic_org;
  // mode = 2: cordic_fptr = & cordic_burk;
  // mode = 3: cordic_fptr = & cordic_vhdl;
  //-------------------------------------------------------------
  void    (* cordic_fptr) (double *x, double *y, double *z, Core *C );

  void    setFuncPtr();

  void    initScale(double *x, double *y);


  //-------------------------------------------------------------
  // Wrapper Function
  //-------------------------------------------------------------
  void    cordic (double *x, double *y, double *z);
  void    cordic_stat (double *x, double *y, double *z, int& cnt, int& xx, int& yy, int& zz);
  void    cordic_break ( double *x, double *y, double *z, int& init);




public:
    double zz;

    // xx = (*x - cosz); sum_xx += xx; sum_xx2 += (xx*xx);
    // yy = (*y - sinz); sum_yy += yy; sum_yy2 += (yy*yy);

    double xx, sum_xx, sum_xx2;
    double yy, sum_yy, sum_yy2;

    double sum_xx_n, sum_xx2_n;
    double sum_yy_n, sum_yy2_n;

    double max_err, max_errn;
    int    cnt_xx, cnt_yy;


    double    SCE,        SSE,        SRE;
    double   sSCE,       sSSE,       sSRE;
    double   mSCE,       mSSE,       mSRE;
    double  rmSCE,      rmSSE,      rmSRE;
    double minSCE,     minSSE,     minSRE;
    double maxSCE,     maxSSE,     maxSRE;
```

```
private:
  int           mode;

  int           useTh;
  int           useThDisp;
  int           useATAN;

  int           level;
  char          path[256];

  double        threshold;
  int           nBreak;
  int           nBreakInit;

  double        pi;
  double        K;
  double        angles[ANGLES_LENGTH];
  double        kprod[KPROD_LENGTH];

};



::::::::::::::
Core.cpp
::::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>


#include "Core.hpp"


using namespace std;

//****************************************************************************
//   Purpose:
//
//     Class Core Implementation Files
//
//  Discussion:
//
//
//  Licensing:
//
```

```
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.14
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//------------------------------------------------------------------------
// Mode                 setMode(),              getMode()
// UseTh                setUseTh(),             getUseTh()
// UseThDisp            setUseThDisp()          getUseThDisp()
// UseATAN              setUseATAN()            getUseATAN()
//
// Level                setLevel(),             getLevel()
// Path                 setPath(),              getPath()
// Threshold            setThreshold(),         getThreshold()
// nBreak;              setNBreak(),            getNBreak()
// nBreakInit;          setNBreakInit(),        getNBreakInit()
//
// Pi                   setPi(),                getPi()
// K                    setK(),                 getK()
// Angles               setAngles()
// KProd                setKProd()
//
//                      dispVars()
//                      initAcc()
//
//                      cordic_fptr()
//                      setFuncPtr()
//                      initScale()
//
//                      cordic()
//                      cordic_stat()
//                      cordic_break()
//
// zz;
// xx, sum_xx, sum_xx2;   //    SCE,        SSE,        SRE;
// yy, sum_yy, sum_yy2;   //   sSCE,       sSSE,       sSRE;
// sum_xx_n, sum_xx2_n;   //   mSCE,       mSSE,       mSRE;
// sum_yy_n, sum_yy2_n;   //  rmSCE,      rmSSE,      rmSRE;
// max_err, max_errn;     // minSCE,     minSSE,     minSRE;
// cnt_xx, cnt_yy;        // maxSCE,     maxSSE,     maxSRE;
//------------------------------------------------------------------------
```

```
Core::Core()
{
  setPi();
  setK();
  setAngles();
  setKprod();

  mode          = 1;

  useTh         = 1;
  useThDisp     = 1;
  useATAN       = 0;

  level         = 10;

  nBreak        = 0;
  nBreakInit    = 0;
  threshold     = 0.0001;

  strcpy(path, "");



}


Core::~Core()
{

}


//---------------------------------------------------------------------------
//  Accessor & Changer
//---------------------------------------------------------------------------
void    Core::setMode      (int m)    { mode      = m;    }
void    Core::setUseTh     (int flag) { useTh     = flag; }
void    Core::setUseThDisp (int flag) { useThDisp = flag; }
void    Core::setUseATAN   (int flag) { useATAN   = flag; }


int     Core::getMode()      { return(mode);      }
int     Core::getUseTh()     { return(useTh);     }
int     Core::getUseThDisp() { return(useThDisp); }
int     Core::getUseATAN()   { return(useATAN);   }

//---------------------------------------------------------------------------
void    Core::setLevel      (int l)      { level      = l;      }
void    Core::setNBreak     (int nB)     { nBreak     = nB;     }
void    Core::setNBreakInit (int nBInit) { nBreakInit = nBInit; }
```

```
void    Core::setThreshold  (double th)  { threshold  = th;     }
void    Core::setPath       (char *p)    { strcpy(path, p);     }


int     Core::getLevel()       { return(level);       }
int     Core::getNBreak()      { return(nBreak);      }
int     Core::getNBreakInit()  { return(nBreakInit);  }
double  Core::getThreshold()   { return(threshold);   }
void    Core::getPath(char *p) { strcpy(p, path);     }

//-----------------------------------------------------------------------
double *Core::getAngles()      { return angles;       }
double *Core::getKprod()       { return kprod;        }


void    Core::dispVars() {
  printf("...............................................................\n");
  printf(". CORDIC Parameter Settings \n");
  printf("...............................................................\n");
  printf(". mode = %d \n", mode);
  printf(". (1: cordic_org, 2: cordic_burk, 3: cordic_vhdl)\n\n");

  printf(". useTh = %d \n", useTh);
  printf(". useThDisp = %d \n", useThDisp);
  printf(". useATAN = %d \n\n", useATAN);

  printf(". level = %d \n", level);
  printf(". path = %s \n\n", path);

  printf(". threshold = %f \n", threshold);
  printf(". nBreak = %d \n", nBreak);
  printf(". nBreakInit = %d \n", nBreakInit);
  printf("...............................................................\n");

if (0) {
  for (int i=0; i < ANGLES_LENGTH; ++i) {
    printf("angles[%d]=%f \n", i, angles[i]);
  }

  for (int i=0; i < KPROD_LENGTH; ++i) {
    printf("kprod[%d]=%f \n", i, kprod[i]);
  }
}


}


  double        pi;
  double        K;
```

```
  double        angles[ANGLES_LENGTH];
  double        kprod[KPROD_LENGTH];



//---------------------------------------------------------------------------
//  Initialize variables for statistics
//---------------------------------------------------------------------------
void Core::initAcc ()
{
    max_err =0.0,  max_errn =0.0;

    sum_xx =0.0,   sum_xx2 =0.0;
    sum_yy =0.0,   sum_yy2 =0.0;

    sum_xx_n =0.0, sum_xx2_n =0.0;
    sum_yy_n =0.0, sum_yy2_n =0.0;

    cnt_xx =0.0,   cnt_yy =0.0;
}



//---------------------------------------------------------------------------
//  Initialize the constants: pi, K
//---------------------------------------------------------------------------
void Core::setPi()
{
  pi = 3.141592653589793;
}


void Core::setK()
{
  K = 1.646760258121;
}

double Core::getPi() { return pi; }
double Core::getK()  { return K;  }


//---------------------------------------------------------------------------
//  Initialize the array Angles[ANGLES_LENGTH]
//---------------------------------------------------------------------------
void Core::setAngles()
{
  double angles_in[ANGLES_LENGTH] = {
    7.85398163397448309615E-01,
    4.63647609000806116214E-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 };

    for (int i=0; i<ANGLES_LENGTH; ++i) {
      angles[i] = angles_in[i];
    }

}


//---------------------------------------------------------------------------
//  Initialize the array kprod[ANGLES_LENGTH]
//---------------------------------------------------------------------------
void Core::setKprod()
{

  double kprod_in[KPROD_LENGTH] = {
    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,
```

```
    0.6072529350088126179,
    0.6072529350088125757,
    0.6072529350088125652,
    0.6072529350088125626,
    0.6072529350088125619,
    0.6072529350088125617 };

    for (int i=0; i<KPROD_LENGTH; ++i) {
      kprod[i] = kprod_in[i];
    }

}

//-----------------------------------------------------------------------
//  cordic ftpr function
//-----------------------------------------------------------------------
void Core::setFuncPtr() {

   switch (mode) {
        case 1 : cordic_fptr = cordic_org;   break;
        case 2 : cordic_fptr = cordic_burk;  break;
        case 3 : cordic_fptr = cordic_vhdl;  break;
        default: cordic_fptr = cordic_org;   break;
   }

}


//-----------------------------------------------------------------------
//  Adjust Initial Scaling Factor (starting with (1,0) or (K, 0))
//-----------------------------------------------------------------------
void Core::initScale(double *x, double *y) {

   switch (mode) {
        case 1 : (*x) = (*x) * K;  break;
        case 2 :   break;
        case 3 :   break;
        default: (*x) = (*x) * K;  break;
   }

}


//-----------------------------------------------------------------------
//  cordic wrapper function
//-----------------------------------------------------------------------
void Core::cordic(double *x, double *y, double *z )
{

   setFuncPtr();
```

```
   initScale(x, y);

   (* cordic_fptr)(x, y, z, this);

   return;
}
```

```
::::::::::::::
Core.1.fptr1.cordic_org.cpp
::::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>

#include "Core.hpp"


using namespace std;

//****************************************************************************
//  Purpose:
//
//     stand alone cordic_org() implementation file
//     friend function of class Core
//
//     [Core.1.fptr1.cordic_org.cpp]
//
//****************************************************************************
//  CORDIC returns the sine and cosine using the CORDIC method.
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.12
//
//  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)
//
//      level : 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)
//                ANGLES(j) = arctan ( (1/2)^(0:59) );
//                ANGLES_LENGTH
//
//    Local, real KPROD(33)
//                KPROD(j) = product ( 0 <= i <= j ) K(i)
//                    K(i) = 1 / sqrt ( 1 + 2^{-2i} )
//                KPROD_LENGTH
//
//-----------------------------------------------------------------------
//
//   C->useATAN  : using arctang function
//   C->useTh    : using thresholding
//      C->nBreakInit
//      C->nBreak
//      C->useThDisp
//
//-----------------------------------------------------------------------


//-----------------------------------------------------------------------
void cordic_org ( double *x, double *y, double *z, Core *C )
{
  double angle;
  double factor;

  double sigma;
```

```
  double poweroftwo;
  double theta;

  double xn, yn;

  int j;



  //---------------------------------------------------------------------
  //  Initialize loop variables:
  //---------------------------------------------------------------------
  xn = *x;
  yn = *y;
  theta = *z;

  poweroftwo = 1.0;

  if (C->useATAN)              // if useATAN, then use arctangent
    angle = atan( 1. );
  else                         // otherwise, use angles array values
    angle = (C->angles)[0];



  //---------------------------------------------------------------------
  for ( j = 1; j <= C->level; j++ )
  //---------------------------------------------------------------------
  {


      if ( theta < 0.0 ) sigma = -1.0;  // if theta is pos, subtract
      else               sigma = +1.0;  // otherwise, add

      //.....................................................................
      // path[i] : the path to the leaf angle in the binary angle tree
      //.....................................................................
      if ( theta < 0.0 ) (C->path)[j-1] = '0'; // left child  : '0' (subtracting)
      else               (C->path)[j-1] = '1'; // right child : '1' (adding)
      (C->path)[j] = '\0';                     // null terminated string


      //.....................................................................
      //   x' = cos(a)*x -sin(a)*y  ==> x' = cos(a) {x        -y*tan(a)}
      //   y' = sin(a)*x +cos(a)*y  ==> y' = cos(a) {x*tan(a)  y       }
      //.....................................................................
      //   Generally, cos(t) = a/r = a/sqrt(a^2+b^2) = 1/sqrt(1+(b/a)^2)
      //   cos(t) = 1/sqrt(1+tan^2(t))
      //.....................................................................
      //   x' = 1/sqrt{1+tan^2(a)} {x        -y*tan(a)}
```

```
//   y' = 1/sqrt{1+tan^2(a)} {x*tan(a)  y       }
//.....................................................................
//   tan(t) = 1/2^i
//   Ki = 1 / sqrt(1 + 2^{-2i}) ==> K = Prod {K_i}
//.....................................................................
//   x' = Ki {x         -y*2^{-2i}}  ==> x" = {x        -y*2^{-2i}}
//   y' = Ki {x*2^{-2i}  y        }  ==> y" = {x*2^{-2i}  y       }
//.....................................................................
//   x" = {x         -y*2^{-2i}}
//   y" = {x*2^{-2i}  y       }
//   a" = a - atan{1/2^i}
//.....................................................................
factor = sigma * poweroftwo;             // +2^(j-1) / -2^(j-1)

*x =          xn - factor * yn;          // x" = {x         -y*2^{-2i}}
*y = factor * xn +          yn;          // y" = {x*2^{-2i}  y        }

xn = *x;
yn = *y;

//.....................................................................
//  Update the remaining angle.
//.....................................................................
theta = theta - sigma * angle;           // a" = a - atan{1/2^i}

*z = theta;

//.....................................................................
// Thresholding:
// If residual angle is less than a given threshold, then break
//.....................................................................
if (C->useTh) {
  static int cntBreak = 0;
  if (C->nBreakInit == 0)         // nBreakInit==0 : init Break counter
    cntBreak = 0;
  if (fabs(*z) < C->threshold) {  // residue less than Th
    C->nBreak = ++cntBreak;       // inc Break counter

    if (C->useThDisp) {           // if useThDisp, display its statistics
      cout << "cntBreak= " << cntBreak;
      cout << "  z= " << right << setw(15) << *z;
      cout << "  < "  << right << setw(7)  << C->threshold;
      cout << "  j= " << right << setw(4)  << j << endl;
    }
    break;
  }
}


//.....................................................................
```

```
      //  Update the angle from table, or eventually by just dividing by two.
      //.................................................................
      poweroftwo = poweroftwo / 2.0;   // 1/2^j

      if (C->useATAN)                  // if useATAN, then use arctangent
        angle = atan( 1. / (1 << j));  // atan(1/2^i)
      else                             // otherwise, use angles array values
        if ( ANGLES_LENGTH < j+1 )  angle = angle / 2.0;
        else                        angle = (C->angles)[j];


  //-----------------------------------------------------------------------
  }  /* end of j */
  //-----------------------------------------------------------------------



  //-----------------------------------------------------------------------
  //  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 ( j > KPROD_LENGTH ) {
    *x = *x * (C->kprod) [ KPROD_LENGTH - 1 ];     // K = 1.647 limit val
    *y = *y * (C->kprod) [ KPROD_LENGTH - 1 ];     // K = 1.647 limit val
  }
  else {
    *x = *x * (C->kprod) [ j - 1 ];                // K = Prod(Ki)
    *y = *y * (C->kprod) [ j - 1 ];                // K = Prod(Ki)
  }

  //
  //  Adjust for possible sign change because angle was originally
  //  not in quadrant 1 or 4.
  //
  //   *c = sign_factor * *c;
  //   *s = sign_factor * *s;

  return;

}
```

```
::::::::::::::
Core.1.fptr2.cordic_burk.cpp
::::::::::::::
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <iomanip>
#include <fstream>


#include "Core.hpp"

using namespace std;

#include "cordic_burkardt.hpp"


//****************************************************************************
//   Purpose:
//
//     stand alone cordic_burk() implementation file
//     friend function of Core class
//
//     [Core.1.fptr2.cordic_burk.cpp]
//
//  Discussion:
//
//
//  CORDIC returns the sine and cosine using the CORDIC method.
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.15
//
//  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)
//
//      level : 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) ).
//
//-----------------------------------------------------------------------
void cordic_burk( double *x, double *y, double *z, Core *C )
{

  using namespace burkardt;

  // using cossin_cordic routine in the file "cordic_burkardt.cpp"
  // void cossin_cordic ( double beta, int n, double *c, double *s );
  //
  // setLevel() is requried
  //
  // See http://people.sc.fsu.edu/~jburkardt/cpp_src/cordic/cordic.html
  //

  cossin_cordic(*z, C->level, x, y);

  return;

}
```

::::::::::::::
Core.1.fptr3.cordic_vhdl.cpp

```
::::::::::::::
#include "Core.hpp"


using namespace std;

extern "C" {
  void  cordic_ghdl( double *x, double *y, double *z) ;
}


//-----------------------------------------------------------------------
//   Purpose:
//
//     stand alone cordic_vhdl() implementation file
//     friend function of Core class
//
//     [Core.1.fptr3.cordic_vhdl.cpp]
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.03.15
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//-----------------------------------------------------------------------
void cordic_vhdl ( double *x, double *y, double *z, Core *C) {


   cordic_ghdl ( x, y, z );


}
```

```
::::::::::::::
Core.2.wrap1.cordic_stat.cpp
::::::::::::::
#include "Core.hpp"

//-----------------------------------------------------------------------
//   Purpose:
//
//     Class Core Implementation Files
//     Core::cordic_stat()
//
//     [Core.2.wrap1.cordic_stat.cpp]
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.15
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//
//    ref var    cnt=0 initialize statistics accumulators
//    ref var    xx = (*x - cosz)
//    ref var    yy = (*y - sinz)
//    ref var    zz = (*z)

//-----------------------------------------------------------------------
void Core::cordic_stat (double *x, double *y, double *z, int& cnt, int& xx, int& yy, int& zz)
{

    double cosz, sinz;


    if (cnt == 0) {
        setNBreak(nBreak=0);
        setNBreakInit(nBreakInit=0);
        initAcc();
        cnt++;
```

```
        sSCE   =   sSSE =   sSRE = 0.0;
        minSCE = minSSE = minSRE = +1.0e+10;
        maxSCE = maxSSE = maxSRE = -1.0e+10;
    }

    cosz = cos(*z);
    sinz = sin(*z);

    setNBreakInit(nBreakInit++);
    //.....................................................
    cordic(x, y, z);
    //.....................................................

    xx = (*x - cosz);
    yy = (*y - sinz);
    zz = (*z);

      SCE  = xx * xx;       SSE  = yy * yy;       SRE  = zz * zz;
     sSCE += SCE;          sSSE += SSE;          sSRE += SRE;
     mSCE  = sSCE/cnt;     mSSE  = sSSE/cnt;     mSRE  = sSRE/cnt;
    rmSCE  = sqrt(mSCE);  rmSSE  = sqrt(mSSE);  rmSRE  = sqrt(mSRE);

    minSCE = (minSCE > SCE) ? SCE : minSCE;
    minSSE = (minSSE > SSE) ? SSE : minSSE;
    minSRE = (minSRE > SRE) ? SRE : minSRE;

    maxSCE = (maxSCE < SCE) ? SCE : maxSCE;
    maxSSE = (maxSSE < SSE) ? SSE : maxSSE;
    maxSRE = (maxSRE < SRE) ? SRE : maxSRE;

}
```

```
::::::::::::::
Core.2.wrap2.cordic_break.cpp
::::::::::::::
#include "Core.hpp"


using namespace std;


//---------------------------------------------------------------------------
//   Purpose:
```

```
//
//     Class Core Implementation Files
//     Core::cordic_break()
//
//     [Core.2.wrap2.cordic_break.cpp]
//
//  Discussion:
//
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    2014.04.15
//
//  Author:
//
//    Young Won Lim
//
//  Parameters:
//    ref var init=0 initializes statistics accumulators
//
//-----------------------------------------------------------------------
void Core::cordic_break ( double *x, double *y, double *z, int& init)
{

    double cosz, sinz;


    if (init == 0) {
        setNBreak(nBreak=0);
        setNBreakInit(nBreakInit=0);
        initAcc();
        init++;
    }

    cosz = cos(*z);
    sinz = sin(*z);

    setNBreakInit(nBreakInit++);
    //.....................................................
    cordic(x, y, z);
    //.....................................................

    xx = (*x - cosz);
    yy = (*y - sinz);

    sum_xx += xx; sum_xx2 += (xx*xx);
```

```
    sum_yy += yy; sum_yy2 += (yy*yy);

    if (max_err < fabs(xx)) max_err = fabs(xx);
    if (max_err < fabs(yy)) max_err = fabs(yy);

    if (fabs(cosz) > 1.0e-10) {
        if (max_errn < fabs(xx/cosz))
            max_errn = fabs(xx/cosz);
        sum_xx_n += xx/cosz;
        sum_xx2_n += (xx*xx)/(cosz*cosz);
        cnt_xx++;
    }
    if (fabs(sinz) > 1.0e-10) {
        if (max_errn < fabs(yy/sinz))
            max_errn = fabs(yy/sinz);
        sum_yy_n += yy/sinz;
        sum_yy2_n += (yy*yy)/(sinz*sinz);
        cnt_yy++;
    }

}
```