GPS Signal Simulator

Bradley University Dept. of Electrical and Computer Engineering

May 12, 2006

Anthony Hoehne
Benjamin Herreid

Advisor:
Dr. In Soo Ahn

Sponsored By:
Tracking and Imaging Systems, Inc.

Table Of Contents

Abstract

The Global Positioning System (GPS) is a satellite based navigation system that provides guaranteed coverage anywhere in the world. This project involves the development of a GPS signal simulator capable of driving a GPS receiver. This process is the reverse of what a GPS receiver does, as the position is given and the GPS satellite signals are computed. The main purpose of the simulator is for testing during the development of GPS receivers. Compared to live testing, a simulator provides convenience, repeatability, and in many scenarios a much lower cost. There are several existing companies that offer similar devices, but the goal of this project was to design a low-cost alternative that would not sacrifice a great amount of accuracy. To operate the simulator, the user supplies the receiver trajectory, antenna characteristics, GPS almanac file, and the initial date, time, and location. These selections will be made using a GUI-based Windows application on a PC. The PC communicates with an FPGA board which, in combination with an up-conversion mixer and a D/A converter, generates an analog GPS signal such that a receiver will track the specified locations and times. The theory behind the project has been developed, and significant portions of this project were built and tested.

Acknowledgements

The following people provided assistance and shared their knowledge to aid in the completion of this project:

The Bradley University Electrical and Computer Engineering Department Faculty, Staff, and Students

Dr. In Soo Ahn
Dr. Donald Schertz
Dr. Prasad Shastry
Dr. Aleksander Malinowski
Mr. Nick Schmidt
Aparna Sankara Subramaniam

Tracking and Imaging Systems, Inc.

Dr. James Sennott
Mr. Dave Seffner

I. Introduction:

i) Purpose:

The purpose of this project was to design a low-cost GPS signal simulator to be used for testing commercial GPS receivers. This simulator was intended to be a simplified version of those already on the market, meaning it would be much cheaper without sacrificing too many features. It was intended that virtually any scenario could be modeled by the simulator, even those not physically possible in reality, and that every test would be completely repeatable as many times as desired.

ii) Theory:

The principle behind the design of a GPS signal simulator is the exact opposite of the design of a receiver. In this case, a known position and trajectory of the receiver is specified by the user. This information is then used to calculate and generate the signals that a receiver would "see" as if it were actually receiving them from GPS satellites in orbit. As long as the right data is used to generate the signals and the signals are generated with the correct codes and frequencies, the receiver being tested will never know that it is not seeing real signals out of the sky on its antenna.

iii) Top Level System Layout:

The GPS signal simulator that was designed consists of 3 major subsystems as shown in Figure 1 below.

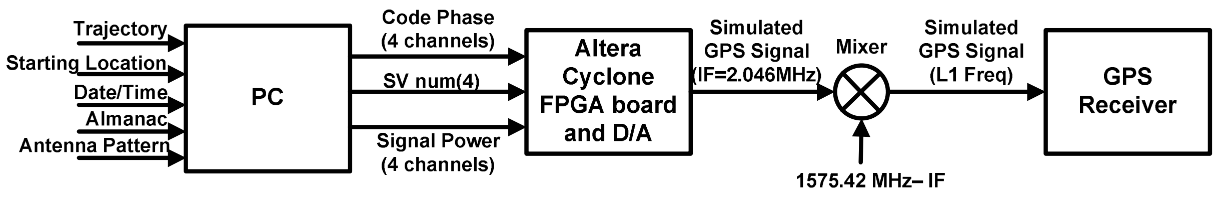


Figure 1: Top level system block diagram

The three major subsystems are the PC and included software, FPGA board, and the analog, or RF subsystem, which includes everything from the D/A converter up to the receiver. The PC subsystem uses a chain of programs that are controlled by a graphical user interface. These programs take in data such as the receiver's starting location and trajectory, as well as a number of parameters and special files that are needed to calculate the desired information. The end result of the PC subsystem is a set of data that can be transmitted to the FPGA board. This data includes line-of-sight code phase, satellite ID number, and signal amplitude level for each of four satellites that are in view of the receiver.

The FPGA subsystem communicates with the PC and uses the received data to index lookup tables which then provide values for the C/A code and carrier signals for each sample. The C/A code is a unique sequence of 1023 bits, or "chips," that is generated by each satellite. This sequence repeats every 1ms on each satellite, and it is used in GPS receivers to lock on to the GPS signal and obtain a position measurement in some cases. Since the sampling rate is much higher than the communication update rate, the FPGA performs a great deal of linear interpolation on the received data in order to calculate the current values at each sample point. The calculations for each satellite in view are done in parallel so that the entire operation can happen in one clock cycle for each sample.

The RF subsystem is the section of the design that actually generates the signals needed to drive the receiver. The D/A converter takes sample data from the FPGA at 8.184 MHz and generates an approximation of the analog signal. This signal, after being converted from differential to single-ended, is then sent to a mixer to mix the signal up to the GPS L1 frequency of 1575.42 MHz. The resulting signal power is too high for a GPS receiver to recognize, so a series of attenuators is used to lower the signal power. The signal is then routed to the receiver for testing.

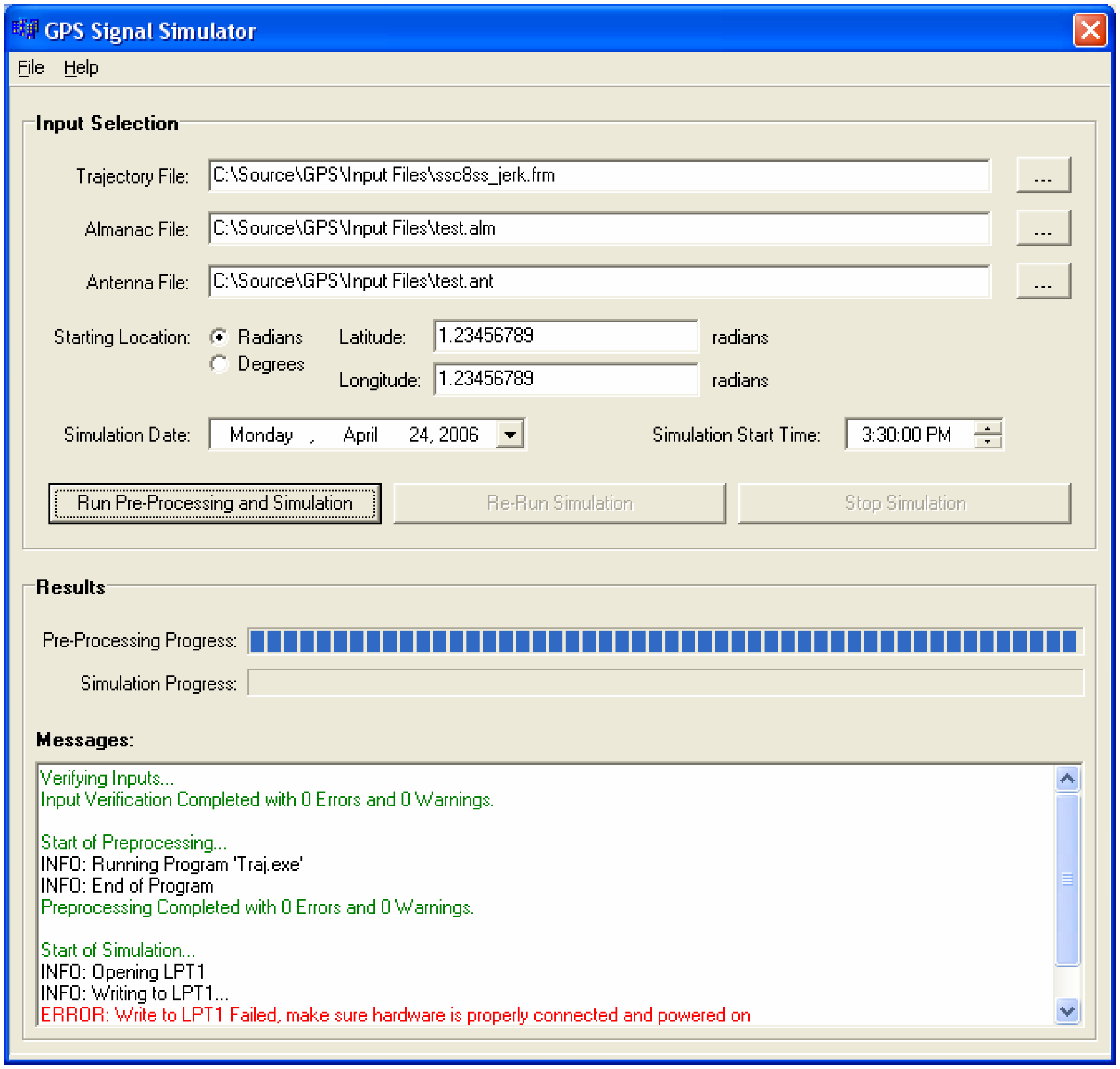
II. <u>Design Process:</u>

i) <u>PC Subsystem:</u>

The first major area of focus was the PC subsystem. This is the software portion of the project which runs on a PC with Windows XP and a parallel port. The main function of the subsystem is to allow the user to provide the simulation inputs. The subsystem then performs a number of calculations and formats the data to be sent to the FPGA.

A screenshot of the GUI application can be seen in Figure 2. This screen allows the user to select and enter all of the necessary inputs. These include a trajectory file, almanac file, antenna file, starting location, and starting date/time. The trajectory file describes the movement of the receiver antenna from the starting location, specified in durations of jerk along the axes of yaw, pitch, roll, and thrust. The initial acceleration, velocity, and attitude are assumed to be zero. The almanac file is in the industry standard RINEX2 format, containing ephemeris data for each satellite. The antenna file models the receiver antenna, containing values for the antenna gain through the ranges of azimuth and elevation angles. This file can also be used to model antenna blockage from certain directions.

Once the user has made these selections, he or she can begin the pre-processing and simulation process. During pre-processing, a series of calculations are performed and the data to be sent to the FPGA is the end result. The simulation process then becomes simply transmitting this data to the FPGA sequentially.



Input Selection

Trajectory File: C:\Source\GPS\Input Files\ssc8ss_jerk.frm

Almanac File: C:\Source\GPS\Input Files\test.alm

Antenna File: C:\Source\GPS\Input Files\test.ant

Starting Location: [✓ Radians] Degrees

Latitude: 1.23456789 radians

Longitude: 1.23456789 radians

Simulation Date: Monday , April 24, 2006

Simulation Start Time: 3:30:00 PM

Results

Pre-Processing Progress:

Simulation Progress:

Messages:

```
Verifying Inputs...
Input Verification Completed with 0 Errors and 0 Warnings.

Start of Preprocessing...
INFO: Running Program 'Traj.exe'
INFO: End of Program
Preprocessing Completed with 0 Errors and 0 Warnings.

Start of Simulation...
INFO: Opening LPT1
INFO: Writing to LPT1...
ERROR: Write to LPT1 Failed, make sure hardware is properly connected and powered on
```

Figure 2: Screenshot of GUI Application

A diagram of the data path for pre-processing can be seen in Figure 3. This figure shows the programs and functions that operate on the user supplied data to generate the data for the FPGA. Most of these programs were supplied by Tracking and Imaging Systems, Inc. Several attempts were made to shorten the data generation process, such as generating the Position file from the Traj program directly in ECEF and obtaining a line-of-sight distance output from the EnRAP v9.0b program. Neither of these methods worked as desired, as the data did not match what was generated by the original data path. These are calculations that are possible to perform, but there is some error in the current implementations.

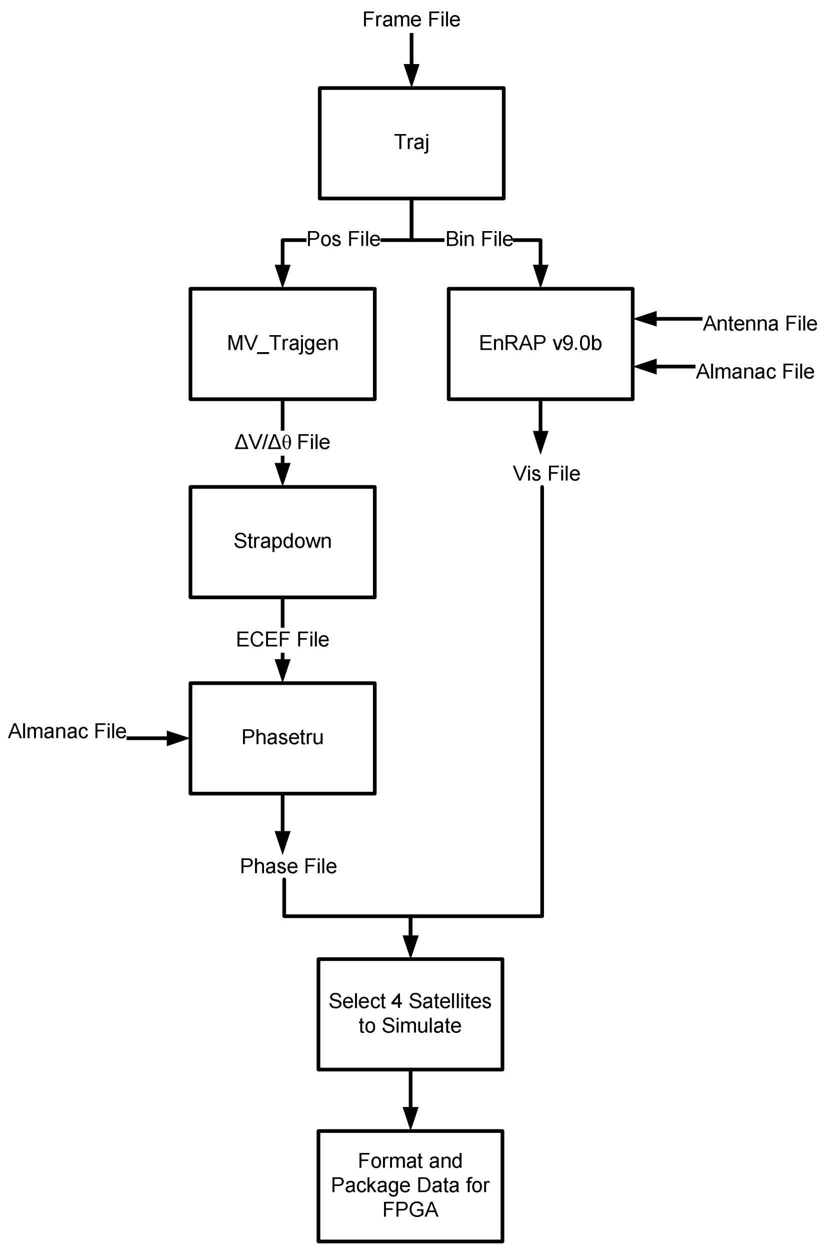


Figure 3: PC Data Path

ii) <u>FPGA Subsystem:</u>

The second major area of focus was the FPGA subsystem. This begins with the communication setup. The communication was never established in lab, but a methodology was established for parallel communication. The data for each satellite required 4 bytes total. These bytes were broken up to carry different types of data in order to utilize space effectively. The required data included the satellite ID number, signal amplitude, fractional code phase, and two flags to indicate overflow and underflow for the linear interpolation algorithm. The satellite number required five bits because there are 32 satellites total. The signal amplitude required seven bits. This was a design choice that was made because it was believed to provide enough dynamic range to the signal amplitude. The code phase required 18 bits, 10 integer to account for all 1023 chips of the code, and 8 fractional to provide enough resolution between data points. The two flags each required one bit each as one would expect.

The transmitted data was set to update at 2 KHz. This allowed a reasonable communication data rate while not requiring excessive interpolation on the FPGA. The communication scheme was to send the four bytes for each satellite in a predefined order so that no further encoding would be necessary to separate the data. Each time data was read in for a new satellite (channel), the data would be stored in a unique register on the FPGA. At the end of the 2 KHz update cycle, the new values would be shifted into the interpolator, and communication would continue for the next set of data. All of this was set to be controlled by handshaking between the FPGA and the PC using a Centronics printer interface, although this was never implemented in practice.

The most critical section of the FPGA subsystem design was the linear interpolation section. Although this was mathematically the simplest part of the entire project, implementing it on the FPGA proved to be somewhat complex. The equation that this section used to calculate the current values was:

$$\frac{\text{(Previous position)} - \text{(Next position)}}{\text{(Previous time)} - \text{(Next time)}} \times \text{(Current time step)} + \text{(Previous position)}$$

Equation 1: Basic linear interpolation used in FPGA calculations

The interpolation section was implemented using schematic entry exclusively (no VHDL written). As in the rest of the design, the interpolation section used fixed-point arithmetic, which added to the difficulty by complicating changes to the design. Once the interpolation section was completed, the wrap-around problem was discovered, and the whole interpolation design had to be reconsidered.

The wrap-around problem arose from the fact that fractional code phases were used instead of whole and fractional numbers. This was done to avoid having to transmit and operate on the extremely large number of the repetitions of the C/A code, which would be a waste of space on the FPGA. Since only fractional numbers were used, the transmitted values would only go from 0 to 1023. This meant that if the signal being generated needed to go into the next whole code cycle, the value used for interpolation would result in a slope in the wrong direction. See Figure 4 for an illustration of this phenomenon.

To avoid this problem, flags needed to be sent from the PC side when the code phase crossed over into a new code cycle. This problem was fixed in the interpolator by using a combination of subtractors, adders, and multiplexers in conjunction with the two flags sent from the PC. With this method, alternate values were always calculated to account for overflow (underflow was never checked for correctness), and the flags and carry bits from the subtractors were used to select whether or not the adjusted values should be used instead of the standard interpolated values. The modified interpolator did work for overflow, and the underflow condition was never completely implemented and verified due to lack of time. However, a problem occurred in the project file, and the updated version was corrupted. Time did not allow for this problem to be fixed appropriately, so the current version of the project still contains the original interpolator without wrap-around protection.

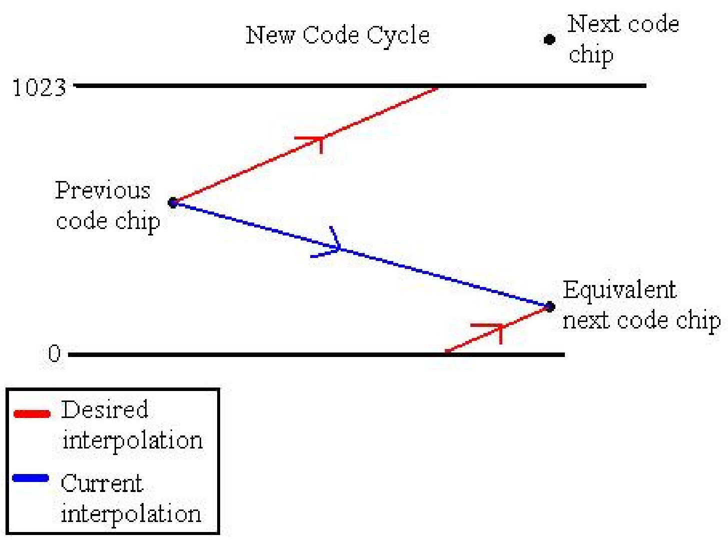


Figure 4: Illustration of interpolation wrap-around problem

At the result end of the interpolator, the interpolated values were used to calculate both the code chip and carrier signal lookup table indices. The code chip index was a simple conversion. The integer result from the interpolator was used directly (after going through a series of multiplexers to get the appropriate value for each channel and satellite number) to index a code chip lookup table that was unique for each satellite. The carrier lookup was slightly more complicated. Here, the fractional part of the interpolated value was used. Since the carrier frequency, IF, was designed to be 2.046 MHz, and the code chipping frequency has to be 1.023 MHz, the result was two complete carrier cycles for each code chip. The decision was made to have a dynamic range of +/- 64 binary values for each channel (7 bits total for amplitude), meaning 128 values were needed to represent the whole range of values. To represent a whole carrier cycle, 256 values were needed. Thus, to get the lookup table index for the carrier table, the fractional interpolated value is divided by 2 (2 carrier cycles per code chip), and this result is then multiplied by 255 (0 to 255 = 256 values) to get a lookup table index between 0 and 255.

iii) <u>RF Subsystem:</u>

The majority of work on the RF subsystem was left unfinished due to time constraints. Some experiments were performed in the RF lab to confirm that the signals coming from the D/A converter could be appropriately upconverted and attenuated to the proper frequencies and signal power levels. There was not a great deal of design involved with this section, and the only major issue was getting the right parts, such as a mixer that could operate with the necessary input and output frequency ranges. Some testing was done on the FPGA output pins to determine the characteristics of the signals at the input to the D/A converter, and this experiment led to one minor addition to the project. As a result of the high sampling rate (8.184 MHz), the signals at the output pins, which are 0V to 3.3V signals, show a large amount of overshoot and ringing. The voltage levels on the high side did not present any problems, but on the falling edges of the signal, the level approached -2V. Since the absolute maximum rating for the D/A converter was -0.3V below ground, the FPGA output was not safe to run into the D/A converter. In order to solve this problem, a germanium diode clipping circuit was proposed to prevent the levels from going too low and damaging the D/A converter. An individual version of this circuit was tested and validated, but 11 of these circuits would be required for full scale operation, one for each of ten bits and one for the sample clock.

III. Results:

i) Overall System Success:

The overall project was only partially successful due to time constraints and the broad scope of the project. Every subsystem section that was implemented was either tested or simulated successfully, but not every subsystem was completed.

ii) PC Success:

The GUI portion of the PC was completed successfully. All of the necessary user controls and inputs were present and functional. The computation behind it was not completed, however. Of the data path shown in Figure 3, only the Traj program was fully integrated into the GUI. The remaining programs were functional when run manually and individually, and valid Visibility and Phase files could be generated. The last two functions, picking satellites and formatting the data, were not completed.

iii) FPGA Success:

On the FPGA subsystem, a communication link was never established with the PC, and although a method was designed for communication, the actual hardware was never implemented. The majority of the calculations on the FPGA were completed for one channel. Once one channel is completed, the only thing left to do would be to copy the schematic once for each channel needed, although a few minor modifications may be needed. The current version of the project file does not include the interpolation with the wrap-around problem fixed, nor does it include the amplitude interpolation. Both of these sections were implemented and simulated successfully, but a problem occurred in the Quartus II software, the design tool for the FPGA, which prevented further compilation of the project, meaning it was necessary to revert to an older version of the project. The only other section not implemented on the FPGA was the addition of the phase due to time calculation. This calculation could be implemented easily using a counter and a few other minor components.

iv) <u>RF Success:</u>

All of the RF subsystem was tested in the lab except for the inclusion of the D/A converter and the conversion to a single-ended signal. Instead, a second signal generator was used to model the expected signals until the D/A converter was running successfully. It was proven experimentally that the specified mixer could handle the necessary signal frequencies and power levels. Also, data was acquired to indicate that some of the attenuation for the signal power levels could be accomplished by adjusting the power levels on the RF signal generator, thereby reducing the amount of attenuation needed after the mixer. In summary, the bulk of the RF subsystem was tested and ready to be added into the overall system, and the only section missing was the D/A converter and conversion to a single-ended signal.

IV. Conclusions:

i) Analysis of Problems:

The primary problem that prevented the completion of this project was time constraints. Essentially, all of the important theory behind the design of the project was established and confirmed in some fashion, but not every subsystem that was proposed could be implemented completely in the available time. All of the steps for the completion of this project are laid out, and provided more time, all of the subsystems could be combined successfully.

ii) Suggested Future Work:

The current status of this project lends itself to the possibility of a continuation project by students next year. Outside of wrapping up minor parts of the FPGA and PC software design, the first major step would be to establish the communication link between the PC and the FPGA. Again, the method for this has already been established, and all that remains is the actual implementation.

Another major step would be to add the navigation data to the generated signals. With the current configuration of this project, the best result that could be expected would be to see a commercial GPS receiver lock on to all four visible satellites individually, but no actual navigation would be possible. For the receiver to obtain a position measurement, the navigation message would have to be present on the generated signals. The addition of this part of the signal is not conceptually difficult, but it requires a large amount of very specific data formatting, meaning that the process would take a great deal of time, and the current FPGA chip may be filled quickly, perhaps not even allowing for the four required channels to be implemented on the chip. In this case, either a larger FPGA or multiple FPGA chips would be required because the navigation message is useless to the receiver if less than four satellites (channels) are available for tracking and navigating.

In addition to the completion of the project in its current form, a number of features could be added to the system to increase the accuracy of the scenario being modeled and to enhance the control of the system for the user. A more realistic scenario could be modeled by including the effects of the ionosphere and troposphere, as well as problems associated with multipath. The addition of these features would not only require further calculations, but some of the assumptions made for the current calculations would no longer be valid, such as the calculation of the carrier signal from the code phase. On the PC side of the system, another interesting feature would be the addition of some type of instrument panel connected with the GUI. This could show items such as an altimeter and speedometer, a scalable map of the earth with the current position shown, as well as the GPS time and the signal power levels being generated. This interface would not affect the generation of the GPS signals in any way, but it would allow the user of the simulator to get additional feedback on the simulation to confirm the correctness of the scenario being modeled.

References

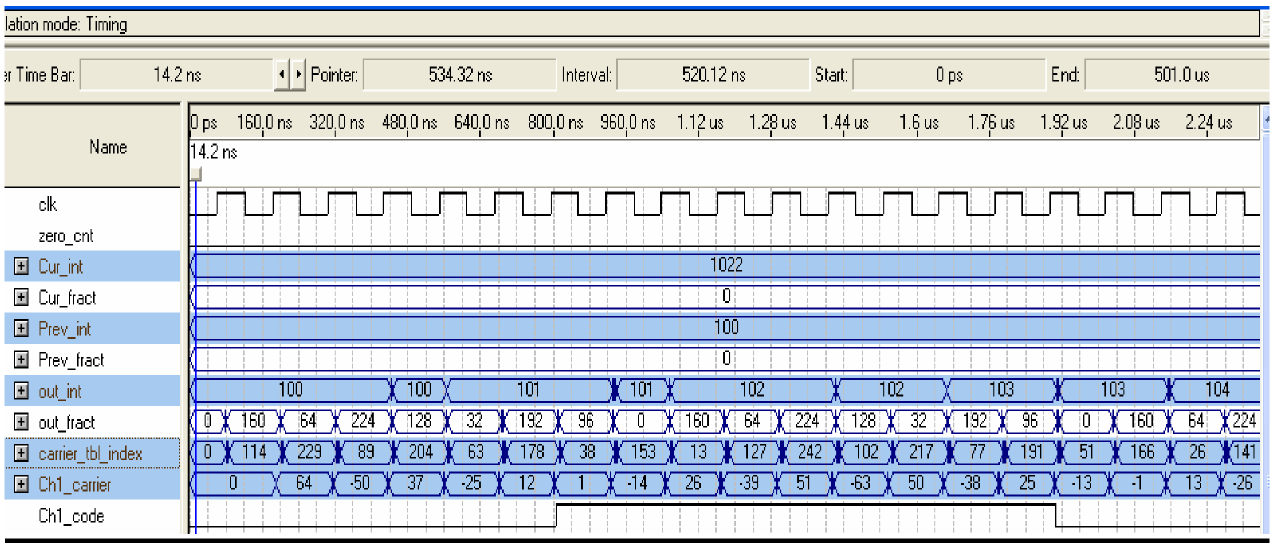
1) United States Patent and Trademark Office. http://www.uspto.gov/main/sitesearch.htm (24 Jan. 2006).

2) Misra, P. and Enge, P. Global Positioning System : Signals, Measurements, and Performance. Lincoln, MA: Ganga-Jamuna Press, 2004.

3) GPS Software Package. Includes: Traj, MV_Trajgen, Strapdown, Phasetru, and EnRAP v9.0b. Tracking and Imaging Systems, Inc. St. Petersburg, FL. 2006.

4) “FPGAs, CPLDs, & Structured ASICs: Altera, the Leader in Programmable Logic” [online]. San Jose, CA. Available : http://altera.com/

Appendix A: Simulations



lation mode: Timing

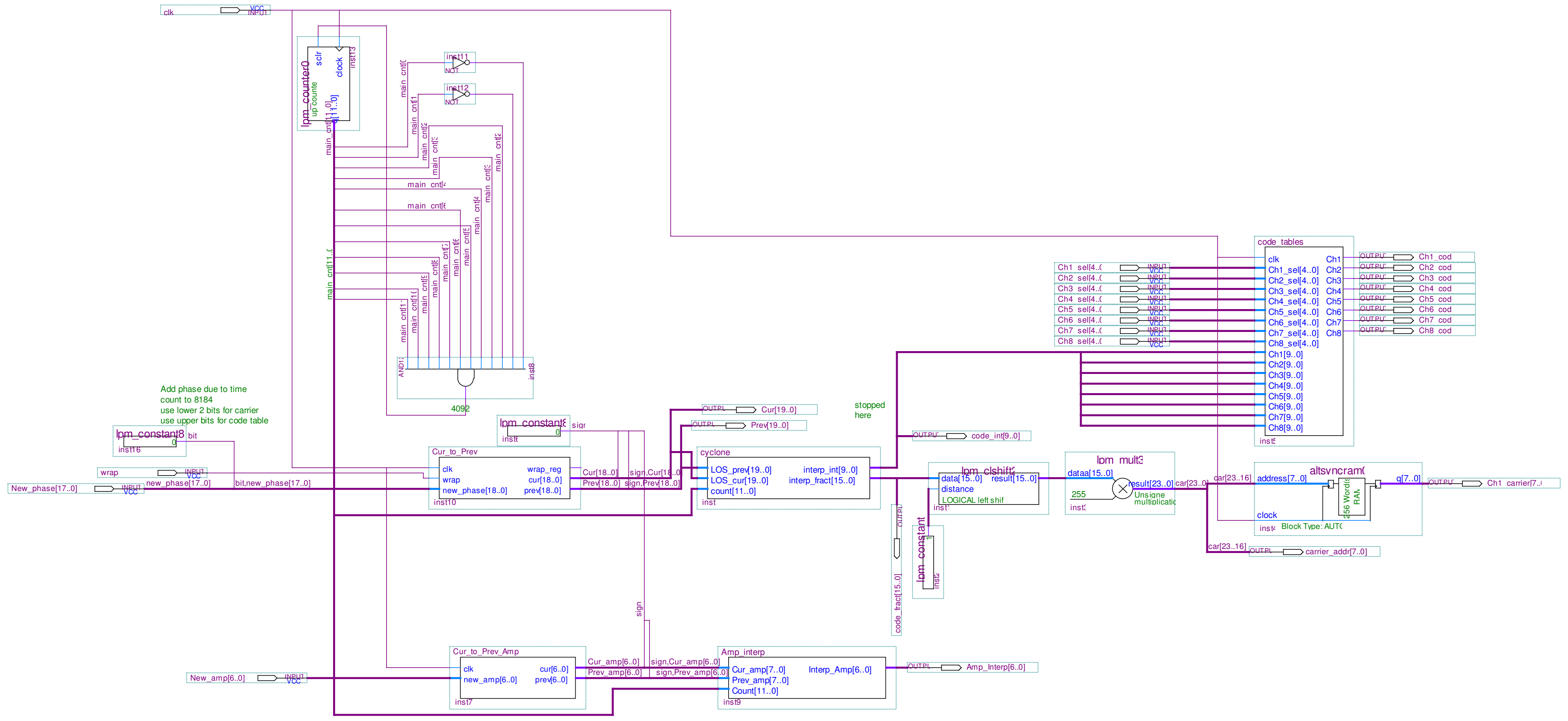
r Time Bar: 14.2 ns Pointer: 534.32 ns Interval: 520.12 ns Start: 0 ps End: 501.0 us

Name
0 ps
160.0 ns
320.0 ns
480.0 ns
640.0 ns
800.0 ns
960.0 ns
1.12 us
1.28 us
1.44 us
1.6 us
1.76 us
1.92 us
2.08 us
2.24 us
14.2 ns
clk
zero_cnt
Cur_int 1022
Cur_fract 0
Prev_int 100
Prev_fract 0
out_int 100 100 101 101 102 102 103 103 104
out_fract 0 160 64 224 128 32 192 96 0 160 64 224 128 32 192 96 0 160 64 224
carrier_tbl_index 0 114 229 89 204 63 178 38 153 13 127 242 102 217 77 191 51 166 26 141
Ch1_carrier 0 64 -50 37 -25 12 1 -14 26 -39 51 -63 50 -38 25 -13 -1 13 -26
Ch1_code

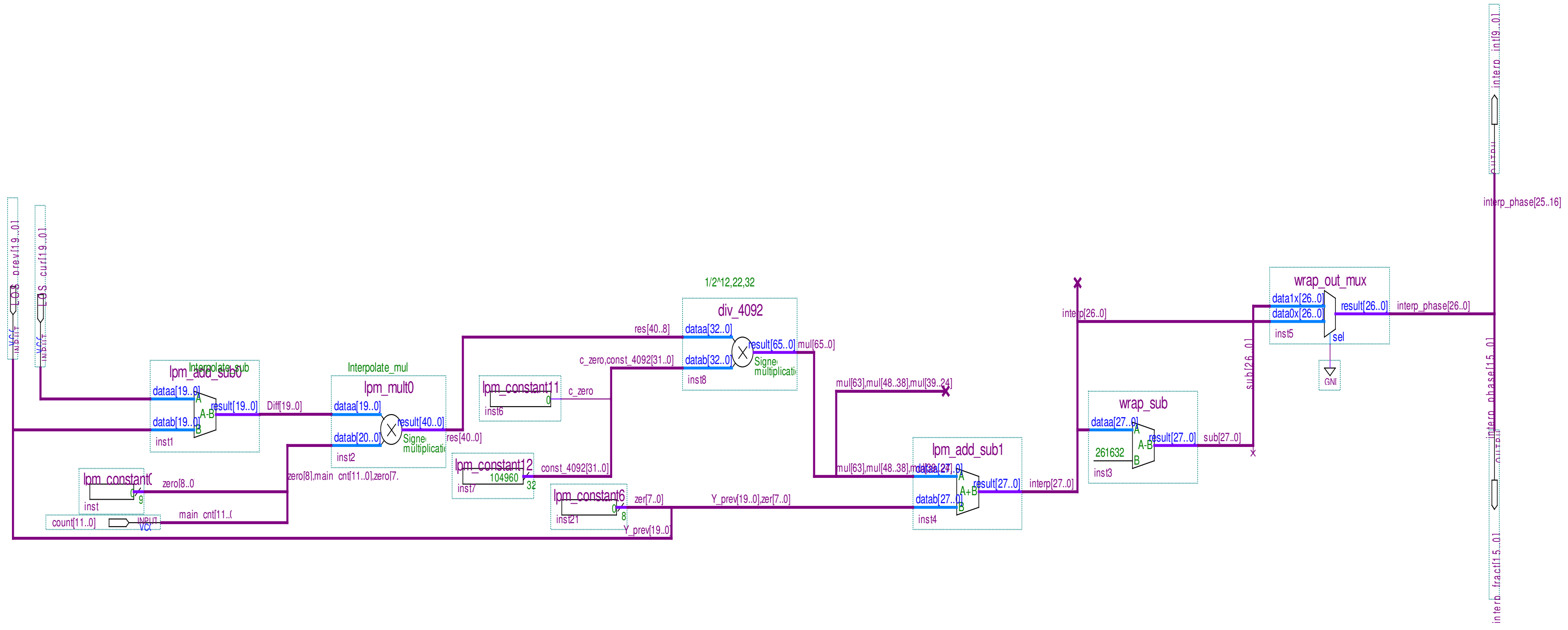
Figure A-1: Showing linear interpolation w/o rollover protection and carrier signal generation.

Figure A-2: Showing generation of C/A code for channel 1, SV #1.

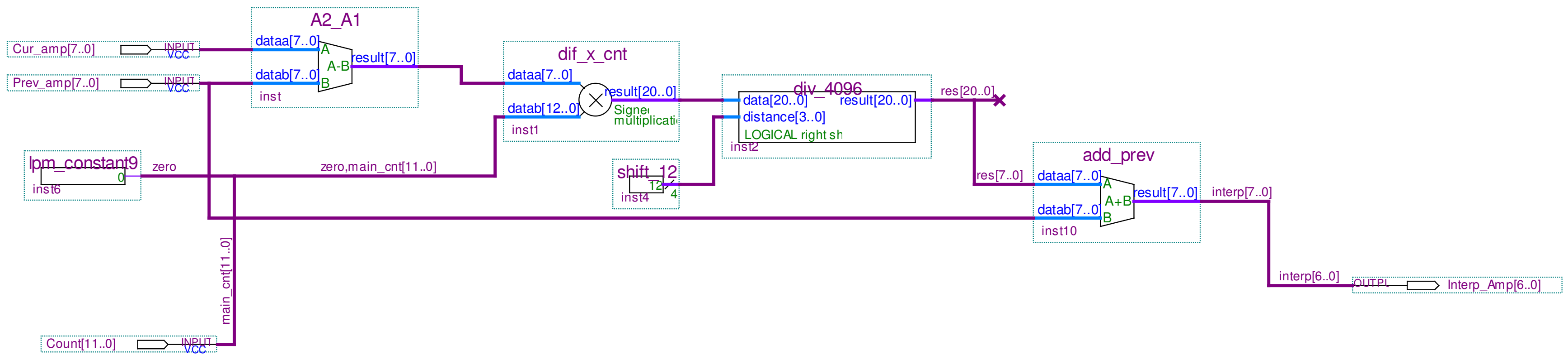
Appendix B: FPGA Design Files



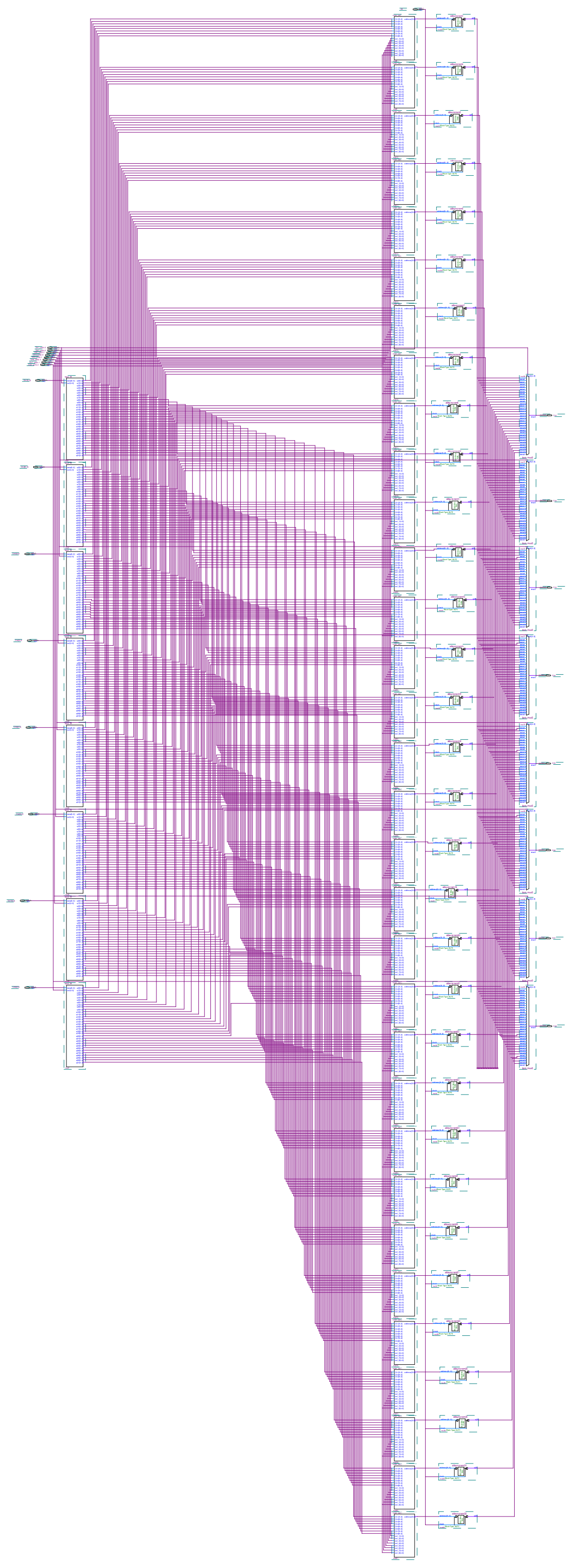
Top Level FPGA Subsystem



Linear Interpolation Block



Amplitude Interpolation



C/A Code Lookup Tables and Indexing

Shifting in next value of code phase due to distance for interpolation

```
--Anthony Hoehne
--3/6/06
--GPS Signal simulator, shift current values of code phase into previous registers for calcs

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

Entity Cur_to_Prev is
Port(
                clk, wrap : IN std_logic;
                wrap_reg  : OUT std_logic;
                --count : IN std_logic_vector(11 downto 0);
                new_phase : IN std_logic_vector(18 downto 0);
                cur, prev : OUT std_logic_vector(18 downto 0)
);
End Cur_to_Prev;

Architecture syn of Cur_to_Prev is

signal temp_cur, temp_prev, cur_add : std_logic_vector(18 downto 0);
signal count : integer range 0 to 4092;
signal wrap_sig : std_logic;

Begin
        Process(clk)
        Begin
                if(rising_edge(clk)) then
                        count <= count + 1;
                        if(count = 0) then
                                cur_add <= new_phase + "0111111111000000000";--register
current value w/ 1022 added(for wrapping)
                                temp_prev <= temp_cur;        --register last code phase
                                temp_cur <= new_phase;        --register current phase unmodified
                                --wrap_sig <= wrap;             --register wrap bit
                                wrap_reg <= wrap;
                                prev <= temp_prev;
                                if(wrap = '1') then
                                        cur <= cur_add;
                                else
                                        cur <= temp_cur;
                                end if;
                        elsif(count = 4092) then
                                count <= 0;
                end if;
            end if;
        end process;

    end syn;
```

Shifting in next signal amplitude value for amplitude interpolation

```
--Anthony Hoehne
--3/6/06
--GPS Signal simulator, shift current values of signal amplitudes into previous registers for calcs

library ieee;
use ieee.std_logic_1164.all;

Entity Cur_to_Prev_Amp is
Port(
                clk : IN std_logic;
                --count : IN std_logic_vector(11 downto 0);
                new_amp : IN std_logic_vector(6 downto 0);
                cur, prev : OUT std_logic_vector(6 downto 0)
);
End Cur_to_Prev_Amp;

Architecture syn of Cur_to_Prev_Amp is

signal temp_cur, temp_prev : std_logic_vector(6 downto 0);
signal count : integer range 0 to 4092;

Begin
        Process(clk)
        Begin
                if(rising_edge(clk)) then
                        count <= count + 1;
                        if(count = 0) then
                                temp_prev <= temp_cur;      --store last code phase
                                temp_cur <= new_amp;        --prepare next code phase for
interpolation
                        elsif(count = 4092) then
                                count <= 0;
                        end if;
                end if;
        end process;
        cur <= temp_cur;
        prev<= temp_prev;
end syn;
```

Appendix C: GUI Code

main_form.h

```
//---------------------------------------------------------------------------

#ifndef main_formH
#define main_formH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>

#include <stdio.h>
//---------------------------------------------------------------------------
class TMainForm : public TForm
{
__published:      // IDE-managed Components
      TButton *RunButton;
        TLabel *Label1;
        TEdit *TrajectoryFileEdit;
        TButton *TrajectoryFileSelectButton;
        TLabel *Label2;
        TEdit *AlmanacFileEdit;
        TLabel *Label3;
        TEdit *AntennaFileEdit;
        TButton *AlmanacFileSelectButton;
        TButton *AntennaFileSelectButton;
        TButton *StopButton;
        TProgressBar *SimulationProgressBar;
        TLabel *Label4;
        TMainMenu *MainMenu1;
        TMenuItem *File1;
        TMenuItem *Exit1;
        TMenuItem *Help1;
        TMenuItem *About1;
        TOpenDialog *TrajectoryFileOpenDialog;
        TOpenDialog *AntennaFileOpenDialog;
        TOpenDialog *AlmanacFileOpenDialog;
      TLabel *Label5;
      TProgressBar *PreprocessingProgressBar;
      TButton *ReRunButton;
      TRichEdit *MessageEdit;
      TLabel *Label6;
      TDateTimePicker *DatePicker;
      TDateTimePicker *TimePicker;
      TLabel *Label7;
      TLabel *Label8;
      TGroupBox *InputsGroupBox;
      TGroupBox *OutputsGroupBox;
      TBevel *Bevel1;
      TLabel *Label9;
      TEdit *RadiansLatEdit;
      TLabel *RadiansLatLabel;
      TEdit *RadiansLonEdit;
      TLabel *RadiansLonLabel;
      TLabel *Label12;
      TLabel *Label13;
      TRadioButton *RadiansRadioButton;
      TRadioButton *DegreesRadioButton;
      TEdit *DegreesLatEdit;
      TEdit *DegreesLonEdit;
      TLabel *DegreesLatLabel;
      TLabel *DegreesLonLabel;
      TEdit *MinutesLonEdit;
      TLabel *MinutesLonLabel;
      TLabel *MinutesLatLabel;
      TEdit *MinutesLatEdit;
      TEdit *SecondsLatEdit;
      TEdit *SecondsLonEdit;
      TLabel *SecondsLonLabel;
      TLabel *SecondsLatLabel;
      TComboBox *DegreesLatDropdown;
      TComboBox *DegreesLonDropdown;
        void __fastcall TrajectoryFileSelectButtonClick(TObject *Sender);
      void __fastcall AlmanacFileSelectButtonClick(TObject *Sender);
      void __fastcall AntennaFileSelectButtonClick(TObject *Sender);
      void __fastcall Exit1Click(TObject *Sender);
      void __fastcall RunButtonClick(TObject *Sender);
      void __fastcall ReRunButtonClick(TObject *Sender);
      void __fastcall RadDegClick(TObject *Sender);
private:    // User declarations
      void Message(TMsgDlgType type, AnsiString msg);
     void ReadMessagesFromFile(void);
      bool GetFileLine(FILE* cf, AnsiString* Line);
     bool RunPreprocessing(void);
     bool RunSimulation(void);
     bool VerifyInputs(void);
     AnsiString ProgramDir;
     int Errors;
     int Warnings;
public:          // User declarations
        __fastcall TMainForm(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TMainForm *MainForm;
//---------------------------------------------------------------------------
#endif
```

main_form.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
```

```
#pragma hdrstop

#include "main_form.h"
#include <SysUtils.hpp>
#include <stdio.h>
#define     PI    ((double)3.1415926535898)
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TMainForm *MainForm;
//---------------------------------------------------------------------------
__fastcall TMainForm::TMainForm(TComponent* Owner)
        : TForm(Owner)
{
      //Initialize Class Variables
     Errors = 0;
     Warnings = 0;
     ProgramDir = GetCurrentDir();

     //Set initial state of controls
     DatePicker->Date = Date();
     TimePicker->Time = Time();
     RadDegClick(this);
     DegreesLatDropdown->ItemIndex = 0;
     DegreesLonDropdown->ItemIndex = 0;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::TrajectoryFileSelectButtonClick(TObject *Sender)
{
      if(TrajectoryFileOpenDialog->Execute())
      TrajectoryFileEdit->Text = TrajectoryFileOpenDialog->FileName;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::AlmanacFileSelectButtonClick(TObject *Sender)
{
      if(AlmanacFileOpenDialog->Execute())
      AlmanacFileEdit->Text = AlmanacFileOpenDialog->FileName;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::AntennaFileSelectButtonClick(TObject *Sender)
{
      if(AntennaFileOpenDialog->Execute())
      AntennaFileEdit->Text = AntennaFileOpenDialog->FileName;
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::RadDegClick(TObject *Sender)
{
      if(RadiansRadioButton->Checked) {
          RadiansLatEdit->Visible = true;
          RadiansLatLabel->Visible = true;
          RadiansLonEdit->Visible = true;
          RadiansLonLabel->Visible = true;
          DegreesLatEdit->Visible = false;
          DegreesLatLabel->Visible = false;
          DegreesLatDropdown->Visible = false;
          DegreesLonEdit->Visible = false;
          DegreesLonLabel->Visible = false;
          DegreesLonDropdown->Visible = false;
          MinutesLatEdit->Visible = false;
          MinutesLatLabel->Visible = false;
          MinutesLonEdit->Visible = false;
          MinutesLonLabel->Visible = false;
          SecondsLatEdit->Visible = false;
          SecondsLatLabel->Visible = false;
          SecondsLonEdit->Visible = false;
          SecondsLonLabel->Visible = false;
     } else {
          DegreesLatEdit->Visible = true;
          DegreesLatLabel->Visible = true;
          DegreesLatDropdown->Visible = true;
          DegreesLonEdit->Visible = true;
          DegreesLonLabel->Visible = true;
          DegreesLonDropdown->Visible = true;
          MinutesLatEdit->Visible = true;
          MinutesLatLabel->Visible = true;
          MinutesLonEdit->Visible = true;
          MinutesLonLabel->Visible = true;
          SecondsLatEdit->Visible = true;
          SecondsLatLabel->Visible = true;
          SecondsLonEdit->Visible = true;
          SecondsLonLabel->Visible = true;

          RadiansLatEdit->Visible = false;
          RadiansLatLabel->Visible = false;
          RadiansLonEdit->Visible = false;
          RadiansLonLabel->Visible = false;
     }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::Exit1Click(TObject *Sender)
{
      Application->Terminate();
}
//---------------------------------------------------------------------------

void TMainForm::Message(TMsgDlgType type, AnsiString msg)
{
      if (type == mtInformation) {
      MessageEdit->SelAttributes->Color = clBlack;
          MessageEdit->Lines->Add("INFO: " + msg);
     } else if (type == mtError) {
      MessageEdit->SelAttributes->Color = clRed;
          MessageEdit->Lines->Add("ERROR: " + msg);
          Errors++;
     } else if (type == mtWarning) {
      MessageEdit->SelAttributes->Color = clBlue;
          MessageEdit->Lines->Add("WARNING: " + msg);
          Warnings++;
     } else if (type == mtCustom) {
      MessageEdit->SelAttributes->Color = clGreen;
          MessageEdit->Lines->Add(msg);
     }
     MessageEdit->Update();
}
//---------------------------------------------------------------------------

void TMainForm::ReadMessagesFromFile(void)
{
     AnsiString Line;
     AnsiString MessageType;
     char c = NULL;

     //open file
     Line = ProgramDir + "\\output.txt";
      FILE* mf = fopen(Line.c_str() ,"r");
     if(!mf) {
      Message(mtError,"Error reading data from program");
          return;
     }

     //read file
     while(GetFileLine(mf,&Line)) {
            TMsgDlgType type;
          MessageType = Line[1];
          try {
            type = (TMsgDlgType) MessageType.ToInt();
          } catch (...) {
            Message(mtError,"Error reading data from program");
                return;
          }
          Line = Line.SubString(2,Line.Length() - 1);
          Message(type,Line);
     }

     //clean up
     fclose(mf);
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::RunButtonClick(TObject *Sender)
{
      //Clear message window
      MessageEdit->Clear();
     bool OK = true;

     /**********************
     * Input Verification *
     **********************/

      Message(mtCustom,"Verifying Inputs...");

     //Reset error counts
     Errors = 0;
     Warnings = 0;
     OK = VerifyInputs();

     if(OK) {
      Message(mtCustom,(AnsiString)"Input Verification Completed with " +
Errors + " Errors and " + Warnings + " Warnings.");
          Message(mtCustom,"");
      } else {
      Message(mtCustom,(AnsiString) "Input Verification Failed with " +
Errors + " Errors and " + Warnings + " Warnings.");
           return;
     }

      /****************
     * Preprocessing *
     ****************/

      Message(mtCustom,"Start of Preprocessing...");

     //Reset error counts
     Errors = 0;
     Warnings = 0;

     ReRunButton->Enabled = false;

     //Call PreProcesing function
     OK = RunPreprocessing();

     if(OK) {
            Message(mtCustom,(AnsiString) "Preprocessing Completed with " +
Errors + " Errors and " + Warnings + " Warnings.");
          Message(mtCustom,"");
      } else {
            Message(mtCustom,(AnsiString) "Preprocessing Failed with " +
Errors + " Errors and " + Warnings + " Warnings.");
          return;
     }

      /*************
     * Simulation *
     *************/

      Message(mtCustom,"Start of Simulation...");

     //Reset error counts
     Errors = 0;
     Warnings = 0;

     StopButton->Enabled = true;

     //Call Simulation function
     OK = RunSimulation();

     StopButton->Enabled = false;

     if(OK) {
            Message(mtCustom,(AnsiString) "Simulation Completed with " +
Errors + " Errors and " + Warnings + " Warnings.");
```

```
          Message(mtCustom,"");
      } else {
            Message(mtCustom,(AnsiString) "Simulation Failed with " + Errors
+ " Errors and " + Warnings + " Warnings.");
          return;
     }
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::ReRunButtonClick(TObject *Sender)
{
      //Clear message window
      MessageEdit->Clear();
}
//---------------------------------------------------------------------------

bool TMainForm::VerifyInputs(void)
{
     bool OK = true;

     if(!FileExists(TrajectoryFileEdit->Text)) {
      Message(mtError,"Trajectory File Not Found");
          OK = false;
     }

     if(!FileExists(AlmanacFileEdit->Text)) {
      Message(mtError,"Almanac File Not Found");
          OK = false;
     }

     if(!FileExists(AntennaFileEdit->Text)) {
      Message(mtError,"Antenna File Not Found");
          OK = false;
     }

     if(RadiansRadioButton->Checked) {

           //verify radians input
      double test;
          bool flag;

          //verify latitude value
          flag = true;
      try {
            test = RadiansLatEdit->Text.ToDouble();
          } catch(...) {
            OK = false;
               flag = false;
               Message(mtError,"Latitude value is not a valid number.");
          }

          if(flag) {
            if(test < 0.0 || test > 2 * PI) {
                  OK = false;
                    Message(mtError,"Latitude value is not valid.  Please
enter a number between 0 and 2 pi.");
               }
          }

          //verify longitude value
          flag = true;
          try {
            test = RadiansLonEdit->Text.ToDouble();
          } catch(...) {
            OK = false;
               flag = false;
               Message(mtError,"Longitude value is not a valid number.");
          }

          if(flag) {
               if(test < -PI || test > PI) {
                  OK = false;
                    Message(mtError,"Longitude value is not valid.  Please
enter a number between -pi and pi.");
               }
          }

     } else {

      //verify degrees input
          double test;
          int testi;
          bool flag;

          //verify latitude degrees value
          flag = true;
          try{
            testi = DegreesLatEdit->Text.ToInt();
          } catch(...) {
            OK = false;
               flag = false;
               Message(mtError,"Latitude Degrees value is not a valid
number.");
          }

          if(flag) {
            if(testi < 0 || testi > 90) {
                  OK = false;
                    flag = false;
                    Message(mtError,"Latitude Degrees value is not valid.
Please enter an integer between 0 and 90.");
               }
          }

          //verify latitude minutes value
          if(flag) {
            try {
                  testi = MinutesLatEdit->Text.ToInt();
               } catch(...) {
                  OK = false;
                    flag = false;
                    Message(mtError,"Latitude Minutes value is not a valid
number.");
               }
               if(flag) {
                  if(testi < 0 || testi > 60) {
                        OK = false;
                         flag = false;
                         Message(mtError,"Latitude Minutes value is not
valid.  Please enter an integer between 0 and 60.");
                    }
               }
          }

          //verify latitude seconds value
          if(flag) {
            try {
                  test = SecondsLatEdit->Text.ToDouble();
               } catch(...) {
                  OK = false;
                    flag = false;
                    Message(mtError,"Latitude Seconds value is not a valid
number.");
               }

               if(flag) {
                  if(test < 0.0 || test > 60.0) {
                        OK = false;
                         flag = false;
                         Message(mtError,"Latitude Seconds value is not
valid.  Please enter a number between 0 and 60.");
                    }
               }
          }

          //special case
          if(flag) {
            if(DegreesLatEdit->Text.ToInt() == 90 && MinutesLatEdit-
>Text.ToInt() != 0 && SecondsLatEdit->Text.ToDouble() != 0.0) {
                  OK = false;
                    Message(mtError,"Total latitude is greater than 90
degrees.");
               }
          }

          //verify longitude degrees value
          flag = true;
          try{
            testi = DegreesLonEdit->Text.ToInt();
          } catch(...) {
            OK = false;
               flag = false;
               Message(mtError,"Longitude Degrees value is not a valid
number.");
          }

          if(flag) {
            if(testi < 0 || testi > 90) {
                  OK = false;
                    flag = false;
                    Message(mtError,"Longitude Degrees value is not valid.
Please enter an integer between 0 and 90.");
               }
          }

          //verify Longitude minutes value
          if(flag) {
            try {
                  testi = MinutesLonEdit->Text.ToInt();
               } catch(...) {
                  OK = false;
                    flag = false;
                    Message(mtError,"Longitude Minutes value is not a valid
number.");
               }

               if(flag) {
                  if(testi < 0 || testi > 60) {
                        OK = false;
                         flag = false;
                         Message(mtError,"Longitude Minutes value is not
valid.  Please enter an integer between 0 and 60.");
                    }
               }
          }

          //verify Longitude seconds value
          if(flag) {
            try {
                  test = SecondsLonEdit->Text.ToDouble();
               } catch(...) {
                  OK = false;
                    flag = false;
                    Message(mtError,"Longitude Seconds value is not a valid
number.");
               }

               if(flag) {
                  if(test < 0.0 || test > 60.0) {
                        OK = false;
                         flag = false;
                         Message(mtError,"Longitude Seconds value is not
valid.  Please enter a number between 0 and 60.");
                    }
               }
          }

          //special case
          if(flag) {
            if(DegreesLonEdit->Text.ToInt() == 90 && MinutesLonEdit-
>Text.ToInt() != 0 && SecondsLonEdit->Text.ToDouble() != 0.0) {
                  OK = false;
                    Message(mtError,"Total longitude is greater than 90
degrees.");
               }
          }
     }
     return OK;
}
//---------------------------------------------------------------------------

bool TMainForm::RunPreprocessing(void)
{
     /****************
     * Preprocessing *
     ****************/

     SimulationProgressBar->Position = 0;
     PreprocessingProgressBar->Position = 0;

     FILE *TrajFile,*AlmFile,*AntFile;

      //open files
      bool OK = true;

     TrajFile = fopen(TrajectoryFileEdit->Text.c_str(),"r");
     if(!TrajFile) {
          Message(mtError,"Error Opening Trajectory File");
          OK = false;
     }

     AlmFile = fopen(AlmanacFileEdit->Text.c_str(),"r");
     if(!AlmFile) {
          Message(mtError,"Error Opening Almanac File");
          fclose(TrajFile);
          OK = false;
     }

     AntFile = fopen(AntennaFileEdit->Text.c_str(),"r");
     if(!AntFile) {
          Message(mtError,"Error Opening Antenna File");
          fclose(TrajFile);
          fclose(AlmFile);
          OK = false;
     }

     if(!OK) return 0;

      //Run TISI Programs
     AnsiString command;

     //run Traj
     Message(mtInformation,"Running Program 'Traj.exe'");
     ChDir(ProgramDir + "\\TISICode\\Traj");
     command = "Traj.exe -fn\"" + TrajectoryFileEdit->Text + "\"";
     system(command.c_str());
      ReadMessagesFromFile();
      if(Errors) return 0;
     PreprocessingProgressBar->Position = 10;

     //clean up
     fclose(TrajFile);
     fclose(AlmFile);
     fclose(AntFile);

     PreprocessingProgressBar->Position = 100;
     return 1;
}
//---------------------------------------------------------------------------

bool TMainForm::RunSimulation(void)
{
     SimulationProgressBar->Position = 0;

      //Verify Input
     FILE *PhaseFile, *VisFile;

     //Open Parallel Port
      HANDLE hParallelPort;
     DWORD OK;

     Message(mtInformation,"Opening LPT1");
      hParallelPort = CreateFile (
        "LPT1",                                 // file or device name
        GENERIC_WRITE,                   // access mode: write only
        NULL,                           // sharing mode: no sharing
        NULL,                           // must be NULL, not supported on all
platforms
        OPEN_EXISTING,                  // whether to create a new file
        NULL,                           // attributes
        NULL                              // must be NULL, not supported on
all platforms
     );

     if(hParallelPort == INVALID_HANDLE_VALUE) {
      Message(mtError,"Unable to open LPT1, make sure device is not in
use.");
          return 0;
     }

     //Set Timeout to 1 second per byte to transfer plus 3 seconds
     COMMTIMEOUTS ct = {0,0,0,1000,3000};
     OK = SetCommTimeouts(hParallelPort,&ct);

      if(!OK) {
            Message(mtWarning,"Unable to set LPT1 Timeouts, program may hang
for several minutes on write failure.");
     }

     char buffer[10];
     DWORD size = 10;
     DWORD done;
     for(int i = 0; i < size; i++)
      buffer[i] = i;

     Message(mtInformation,"Writing to LPT1...");
      OK = WriteFile(
                  hParallelPort,             // file or device handle
            &buffer,                   // address of buffer with data to send
            size,                      // how many bytes should be sent
            &done,                      // how many bytes were actually written
            NULL                        // must be NULL, not supported
      );

      if(!OK) {
            Message(mtError,"Write to LPT1 Failed, make sure hardware is
properly connected and powered on");
          CloseHandle(hParallelPort);
          return 0;
     }

      Message(mtInformation,(AnsiString)"INFO: Wrote " + done + " Bytes");

     //Clean Up
     CloseHandle(hParallelPort);

     SimulationProgressBar->Position = 100;
      return 1;
}
//---------------------------------------------------------------------------

//unbuffered (dynamic) alternative to fgets, does NOT include \n at end of
line
bool TMainForm::GetFileLine(FILE* cf, AnsiString* Line)
{
      *Line = "";
      char NextCharacter;

      while(1) {
            NextCharacter = fgetc(cf);
            if(NextCharacter == EOF)
                  return(0);
            if(NextCharacter == '\n')
                  break;
            *Line = *Line + NextCharacter;
      }

      return(1);
}
//---------------------------------------------------------------------------
```