University of Toronto
Department of Electrical & Computer Engineering

Digital Hardware Final Project Group Report

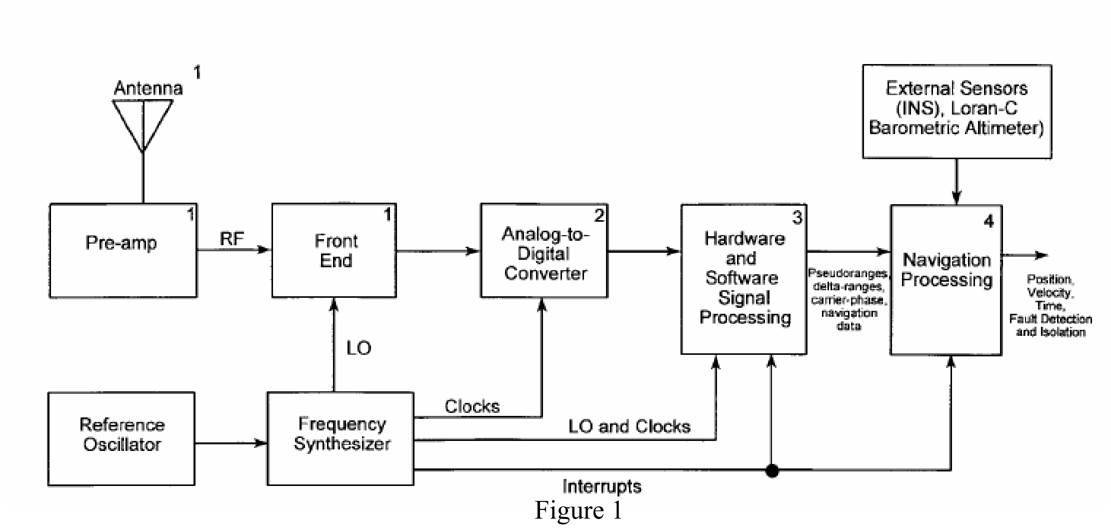
Implementation of the back-end of a GPS receiver system in Xilinx Multimedia Board featuring Virtex II FPGA and MicroBlaze Soft processor

Alborz Jooyaie

Meysam Roodi

Table of Contents

Introduction


Antenna
1
Pre-amp
1
RF
Front End
1
Analog-to-Digital Converter
2
Hardware and Software Signal Processing
3
Pseudoranges, delta-ranges, carrier-phase, navigation data
Navigation Processing
4
External Sensors (INS), Loran-C Barometric Altimeter)
Position, Velocity, Time, Fault Detection and Isolation
LO
Reference Oscillator
Frequency Synthesizer
Clocks
LO and Clocks
Interrupts
Figure 1

Global Positioning System (GPS) is a system composed of 32 satellites and is incorporated in navigational systems. The principle of operation relies on the time of arrival and the equivalent of triangulation in three dimensional coordinate system, called Trilateration. From the thirty-two satellite vehicles, twenty-four are actually in use for navigational purposes and the spare ones are in place for emergency uses, such as replacement and repair. Four satellite vehicles circle each of the six orbits around the earth and together, with the Earth, form the system of Trilateration used for locating the position of a receiver system. From the Radio Frequency (RF) signals the receiver receives from the satellites, it extracts the geometrical distance to each of the four satellites and evaluates its own location on earth. The signals sent by the satellites are composed of three components: the carrier at 1.57 GHz, the Pseudo-Random Noise (PRN) at 10.23 MHz, and the Data at 50 bits/second. The Pseudo-random noise signal is used to determine which satellite is transmitting the signal and is also used to determine the pseudo-range. Data signal at 50 Hz is used to inform the receiver of the actual position of the satellite vehicle, its velocity and heading, as well as some maintenance and health stats used by ground-based GPS monitoring stations.

A GPS receiver unit is composed of an analog front-end, and a digital back-end, an elaborated diagram of which is illustrated in figure 1. The analog front-end is in charge of processing the received RF signal to minimize the noise component and amplify the actual transmitted signal at 1.57 GHz which is embedded in the inherited noise from the atmosphere as well as device mismatches. Moreover, the front end lowers the frequency of the RF signal from 1.57 GHz to 10 MHz, which is in turn converted to binary sequence using an over-sampled Delta-Sigma A/D converter. The digital data is sent to the backend, which is in charge of performing the mathematical operations required in extracting the location of the receiver.

Our objective in this project was to model the backend of a GPS receiver system in the Xilinx Multimedia board incorporating the Virtex-II FPGA. The actual digital data from the front-end of a commercial receiver was to be fed into the Xilinx board for digital base-band processing. The back-end operations were modeled and studied first with

Matlab program and then implemented in hardware with Verilog and appropriate IP Cores.

Overview

The implementation of the GPS back-end is a cumbersome and elaborate job, and professionals in the industry have spent much time optimizing and designing the different components. Due to time constraints and feasibility issues the implementation of the main components of the back-end, which are related to the extraction of the pseudo range, have been aimed for. In order to achieve this goal, the signal received has to be associated with the particular satellite vehicle sending it. After detecting the satellite vehicle, the signal received is correlated with the locally generated PRN code associated with that satellite in order to compute the phase shift affecting the transmitted PRN signal. This phase shift will be later on converted to appropriate units of time (using well known mathematical equations applied to each particular satellite) and upon multiplication with the speed of light will result in the pseudo range. The range found is not the actual geometrical range due to the clock mismatches which are to be taken into account at a later stage.

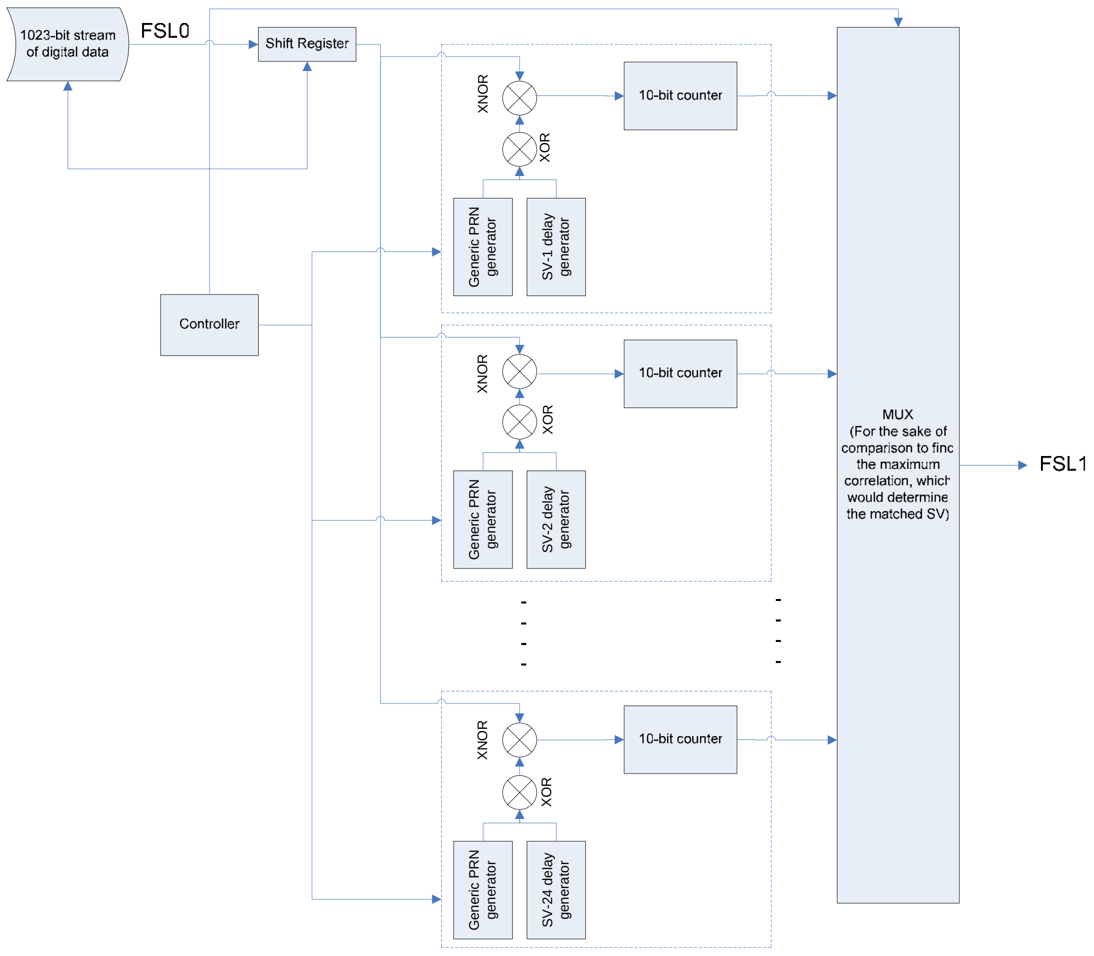


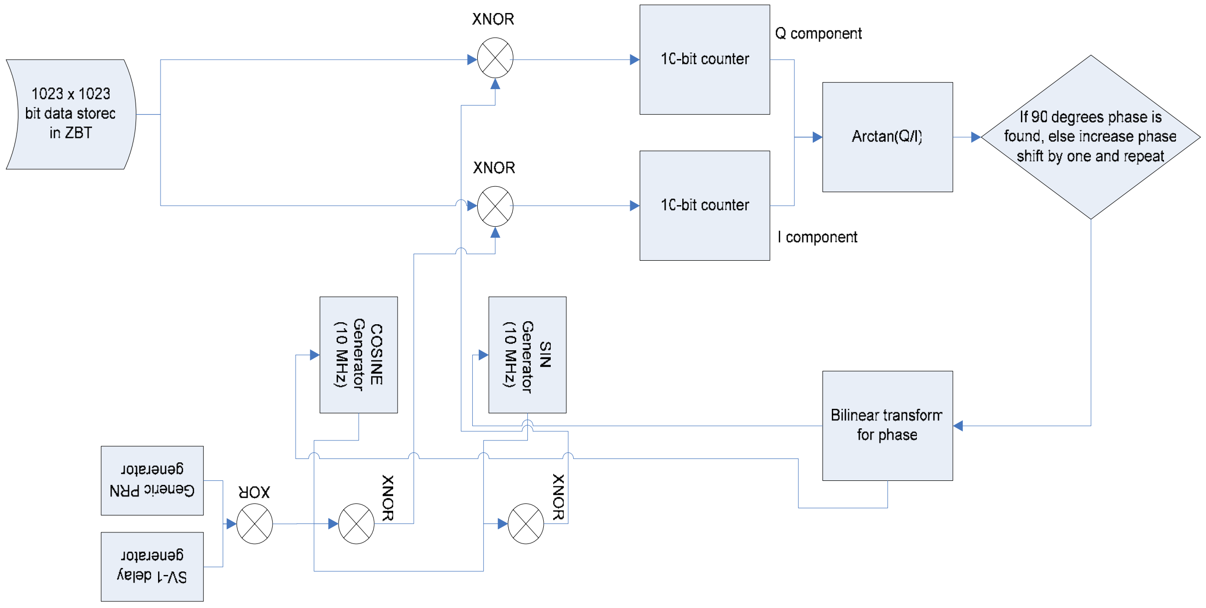
Figure 2

Before sending data, each satellite generates a Pseudo Random Noise (PRN) code which uses it to encode the data (the encoding used is Binary Phase Shift Keying, BPSK). The PRN code of all 24 satellites are the same, but every satellite applies a delay before encoding data with the generated code. The amount of delay for each of the satellites is specified. The delay varies from 5 to 512 for satellite 1 to 24. The standard way of generating delay in hardware is using shift registers, in which we need as much flip flop (register) as the number of delay. In [1], another way of generating delays for satellites is introduced. In this way, the delay is generated using a simple 10 bit shift register and the output is made of gating two bits of the shift register. (Refer to individual reports for the delay block architecture)

The receiver should generate the same codes of all 24 satellites and compares the incoming data with them. The satellite which has the longest match with the incoming data is the sender. After detecting the sender satellite the receiver can continue with the rest of the activities to have the position.

As it can be seen from figure 2, the system consists of repeated blocks for satellites. The difference among them is the amount of delay which is taken into account in hardware design. Each satellite block has a PRN code generator, a delay generator and a counter which counts the number of matches between the incoming data and the generated data.

Once the satellite is detected, the amount of Phase shift is found as shown in the figure below, Figure 3. The 1023 x 1023 bit data is the digital data extracted from the output of the A/D converter of a commercial GPS front-end which has been first sampled using a deep memory logic analyzer and then stored on a single 2MB ZBT of the Xilinx board. The functionality of the figure is as explained below.



1023 x 1023 bit data stored in ZBT
XNOR
XNOR
10-bit counter
10-bit counter
Q component
I component
Arctan(Q/I)
If 90 degrees phase is found, else increase phase shift by one and repeat
Bilinear transform for phase
COSINE Generator (10 MHz)
SIN Generator (10 MHz)
Generic PRN generator
SV-1 delay generator
XOR
XNOR
XNOR

Figure 3

In the diagram above it is assumed that the satellite vehicle detected is the first one. The PRN code associated with that satellite is locally generated, with the appropriate delay, and a phase shift of zero. Then, the PRN code is XNORed with the output of a sin and cosine look up tables, stored in two separate ZBT blocks, which are sampled at the same rate as the input digital data of 10 MHz. The result of the two XNOR is the quadrature and in-phase data components. Afterwards, the Quadrature and In-phase component are each correlated with the incoming digital data stream from the ZBT. The correlation procedure employed is the same as that in the satellite detection part, applying a XNOR and counter. The phase is matched once the Quadrature component (sin) is zero. Matching is evaluated with an Arctangent block, which determines the Arctangent(Q/I) in radians. The output of the Arctangent function is fed to a block which determines the amount of phase shift required. The phase shift is fed back into the sin and cosine look up tables, after a bilinear transformation. This process repeats and the phase shift is adjusted with the Arctangent and bilinear transform blocks until the Quadrature component becomes zero. Once this amount of shift is determined it is fed into the mathematical computational blocks which are in charge of applying the pre-defined equations in order to extract the time of arrival from the phase shift.

The mathematical modeling and studying of the satellite detection and phase shifting/matching were first carried with Matlab. A sample of the Matlab code for PRN and Course Acquisition bit pattern is documented below. Satellite detection part of the system is coded in Verilog HDL and the top module of the code follows the MATLAB code. The Verilog and Matlab codes employed are documented in detail in the individual reports.

Matlab Code

```
-----------------------------------------------------------------
%This function generates C/A sattelite signal code, for all SV PRN
Numbers. ( 1 <= No. <=32 )
%The length of one code sequence is 1023 bits.  The code will be held
in a 1023*32 matrix such that
%the column corresponds to each respective sattelite code.
CAcodegeninit

CAcode=zeros(1023, 32);
registerLength=10;
  for codeNumber=1:1:32

  tapSel = CAtapselection(codeNumber,:);     %read appropriate XOR
operands
  registerBdelay = CAdelay(codeNumber,:);    %read appropriate register
B delay
  registerA = CAfirstchips(codeNumber,:);    %read appropriate register
A init. bits
  codeRegister=zeros(1, 1023+registerLength);  %holds registerA
generated code
  trimmedCode=zeros(1, 1023);        %codeRegister - registerA parts
  finalCode=zeros(1, 1023);                  %holds final CA code for
the SV PRN No.
  codeRegister(1, 1024:1:(1023+registerLength)) = registerA;


  %In a hardware implementation registerA would shift out code bits to
the right.  The sequence of shifted
```

```
  %out bits would comprise the 1023 bit code.  In software this can be
done differently, since one can create
  %a 1023+10=1033 matrix and generate bits towards the LEFT direction.
The last 11->1033 bits will comprise
  %the 1023 bit unXORed code.  To create registerB the registerA matrix
can be shifted by the appropriate
  %amount.  The two matrixes will then be XORed with eachother to
generate the final CA code for the SV PRN No.
  %of interest.

  offset1=tapSel(1, 1);
  offset2=tapSel(1, 2);
  for codeIndex=1023:-1:1
   codeRegister(1, codeIndex) = xor(codeRegister(1,
(codeIndex+offset1)), codeRegister(1, (codeIndex+offset2)));
  end
  trimmedCode(1, :) = codeRegister(1,
(1+registerLength):(1023+registerLength));
  for codeIndex2=1:1:1023
      if(registerBdelay<codeIndex2)
        finalCode(1, codeIndex2) = xor(trimmedCode(1, codeIndex2),
trimmedCode(1, (codeIndex2-registerBdelay)));
      else
        finalCode(1, codeIndex2) = xor(trimmedCode(1, codeIndex2),
trimmedCode(1, mod((codeIndex2-registerBdelay-1), 1024)));
      end
  end
  CAcode(:, codeNumber) = finalCode';
  fprintf(stderr, "Finished generating SV PRN Code No. %02d of 32\r",
codeNumber);
  fflush(stderr);
  end
fprintf(stderr,"
\r");
fflush(stderr);
save -ascii CAcode.txt CAcode
%The End
----------------------------------------------------------------------
```

Verilog Code

```
module Sat_Detector ( clk, reset_b, data_in,
                    fifo_data_exists, load, fifo_full, fifo_out_write,
                    data_out);

        input clk, reset_b, fifo_data_exists , fifo_full;
        output load, fifo_out_write ;
        input [31:0] data_in ;
        output [9:0] data_out ;

        wire datain, data_feed_enable ;
        wire [4:0] mux_sel ;
        wire [9:0] cnt_out1, cnt_out2 , cnt_out3, cnt_out4, cnt_out5, cnt_out6, cnt_out7,
        cnt_out8  ,cnt_out9,  cnt_out10,  cnt_out11,  cnt_out12,  cnt_out13,  cnt_out14,
        cnt_out15,  cnt_out16,  cnt_out17,  cnt_out18  ,  cnt_out19,  cnt_out20,  cnt_out21,
        cnt_out22, cnt_out23, cnt_out24 ;

        wire [24:1] ser_out ;
        wire [10:1] par_out1, par_out2, par_out3, par_out4, par_out5, par_out6, par_out7,
        par_out8,par_out9,   par_out10,   par_out11,   par_out12,   par_out13,   par_out14,
        par_out15,  par_out16,  par_out17,  par_out18,  par_out19,  par_out20,  par_out21,
        par_out22, par_out23, par_out24;
```

```
system_controller    sat_controller    (.clk(clk),    .reset_b(reset_b),    .data_exists
(fifo_data_exists)   ,   .data_in   (data_in),   .shifted_data   (datain),   .data_out_write
(fifo_out_write),    .select_data    (mux_sel),    .data_feed_enable    (data_feed_enable),
.load(load), .full(fifo_full) ) ;

mux10bit_24to1 OutputFeeder (
        .MA(cnt_out1),
        .MB(cnt_out2),
        .MC(cnt_out3),
        .MD(cnt_out4),
        .ME(cnt_out5),
        .MF(cnt_out6),
        .MG(cnt_out7),
        .MH(cnt_out8),
        .MAA(cnt_out9),
        .MAB(cnt_out10),
        .MAC(cnt_out11),
        .MAD(cnt_out12),
        .MAE(cnt_out13),
        .MAF(cnt_out14),
        .MAG(cnt_out15),
        .MAH(cnt_out16),
        .MBA(cnt_out17),
        .MBB(cnt_out18),
        .MBC(cnt_out19),
        .MBD(cnt_out20),
        .MBE(cnt_out21),
        .MBF(cnt_out22),
        .MBG(cnt_out23),
        .MBH(cnt_out24),
        .S(mux_sel),
        .O(data_out));

//Satelite No. 1
        Basic_PRN   satu1   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[1]),
.G2_parout(par_out1) , .enable(data_feed_enable));
        Match_Counter counteru1 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[1]^(par_out1[2]^par_out1[6]))~^datain) , .cnt_out(cnt_out1) );

//Satelite No. 2
        Basic_PRN   satu2   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[2]),
.G2_parout(par_out2), .enable(data_feed_enable));
        Match_Counter counteru2 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[2]^(par_out2[3]^par_out2[7]))~^datain) , .cnt_out(cnt_out2) );

//Satelite No. 3
        Basic_PRN   satu3   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[3]),
.G2_parout(par_out3), .enable(data_feed_enable));
        Match_Counter counteru3 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[3]^(par_out3[4]^par_out3[4]))~^datain) , .cnt_out(cnt_out3) );

//Satelite No. 4
        Basic_PRN   satu4   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[4]),
.G2_parout(par_out4), .enable(data_feed_enable));
        Match_Counter counteru4 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[4]^(par_out4[5]^par_out4[9]))~^datain) , .cnt_out(cnt_out4) );

//Satelite No. 5
        Basic_PRN   satu5   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[5]),
.G2_parout(par_out5), .enable(data_feed_enable));
        Match_Counter counteru5 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[5]^(par_out5[1]^par_out5[9]))~^datain) , .cnt_out(cnt_out5) );

//Satelite No. 6
        Basic_PRN   satu6   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[6]),
.G2_parout(par_out6), .enable(data_feed_enable));
        Match_Counter counteru6 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[6]^(par_out6[2]^par_out6[10]))~^datain) , .cnt_out(cnt_out6) );

//Satelite No. 7
```

```
	Basic_PRN   satu7   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[7]),
.G2_parout(par_out7), .enable(data_feed_enable));
	Match_Counter counteru7 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[7]^(par_out7[1]^par_out7[8]))~^datain) , .cnt_out(cnt_out7) );

//Satelite No. 8
	Basic_PRN   satu8   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[8]),
.G2_parout(par_out8), .enable(data_feed_enable));
	Match_Counter counteru8 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[8]^(par_out8[2]^par_out8[9]))~^datain) , .cnt_out(cnt_out8) );

////////////////////////////////////////////////////////////////////////////////
//Satelite No. 9
	Basic_PRN   satu9   (   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[9]),
.G2_parout(par_out9), .enable(data_feed_enable));
	Match_Counter counteru9 ( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[9]^(par_out9[3]^par_out9[10]))~^datain) , .cnt_out(cnt_out9) );

//Satelite No. 10
	Basic_PRN   satu10(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[10]),
.G2_parout(par_out10), .enable(data_feed_enable));
	Match_Counter counteru10( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[10]^(par_out10[3]^par_out10[2]))~^datain) , .cnt_out(cnt_out10) );

//Satelite No. 11
	Basic_PRN   satu11(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[11]),
.G2_parout(par_out11), .enable(data_feed_enable));
	Match_Counter counteru11( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[11]^(par_out11[3]^par_out11[4]))~^datain) , .cnt_out(cnt_out11) );

//Satelite No. 12
	Basic_PRN   satu12(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[12]),
.G2_parout(par_out12), .enable(data_feed_enable));
	Match_Counter counteru12( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[12]^(par_out12[5]^par_out12[6]))~^datain) , .cnt_out(cnt_out12) );

//Satelite No. 13
	Basic_PRN   satu13(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[13]),
.G2_parout(par_out13), .enable(data_feed_enable));
	Match_Counter counteru13( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[13]^(par_out13[6]^par_out13[7]))~^datain) , .cnt_out(cnt_out13) );

//Satelite No. 14
	Basic_PRN   satu14(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[14]),
.G2_parout(par_out14), .enable(data_feed_enable));
	Match_Counter counteru14( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[14]^(par_out14[7]^par_out14[8]))~^datain) , .cnt_out(cnt_out14) );

//Satelite No. 15
	Basic_PRN   satu15(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[15]),
.G2_parout(par_out15), .enable(data_feed_enable));
	Match_Counter counteru15( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[15]^(par_out15[8]^par_out15[9]))~^datain) , .cnt_out(cnt_out15) );

//Satelite No. 16
	Basic_PRN   satu16(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[16]),
.G2_parout(par_out16), .enable(data_feed_enable));
	Match_Counter counteru16( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[16]^(par_out16[9]^par_out16[10]))~^datain) , .cnt_out(cnt_out16) );

////////////////////////////////////////////////////////////////////////////////
//Satelite No. 17
	Basic_PRN   satu17(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[17]),
.G2_parout(par_out17), .enable(data_feed_enable));
	Match_Counter counteru17( .clk(clk), .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[17]^(par_out17[1]^par_out17[4]))~^datain) , .cnt_out(cnt_out17) );

//Satelite No. 18
	Basic_PRN   satu18(   .clk(clk),   .reset_b(reset_b),   .G1_serout(ser_out[18]),
.G2_parout(par_out18), .enable(data_feed_enable));
```

```
        Match_Counter  counteru18(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[18]^(par_out18[2]^par_out18[5]))~^datain) , .cnt_out(cnt_out18) );

//Satelite No. 19
        Basic_PRN    satu19(    .clk(clk),    .reset_b(reset_b),    .G1_serout(ser_out[19]),
.G2_parout(par_out19), .enable(data_feed_enable));
        Match_Counter  counteru19(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[19]^(par_out19[3]^par_out19[6]))~^datain) , .cnt_out(cnt_out19) );

//Satelite No. 20
        Basic_PRN    satu20(    .clk(clk),    .reset_b(reset_b),    .G1_serout(ser_out[20]),
.G2_parout(par_out20), .enable(data_feed_enable));
        Match_Counter  counteru20(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[20]^(par_out20[4]^par_out20[7]))~^datain) , .cnt_out(cnt_out20) );

//Satelite No. 21
        Basic_PRN    satu21(    .clk(clk),    .reset_b(reset_b),    .G1_serout(ser_out[21]),
.G2_parout(par_out21), .enable(data_feed_enable));
        Match_Counter  counteru21(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[21]^(par_out21[5]^par_out21[8]))~^datain) , .cnt_out(cnt_out21) );

//Satelite No. 22
        Basic_PRN    satu22(    .clk(clk),    .reset_b(reset_b),    .G1_serout(ser_out[22]),
.G2_parout(par_out22), .enable(data_feed_enable));
        Match_Counter  counteru22(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[22]^(par_out22[9]^par_out22[6]))~^datain) , .cnt_out(cnt_out22) );

//Satelite No. 23
        Basic_PRN    satu23(    .clk(clk),    .reset_b(reset_b),    .G1_serout(ser_out[23]),
.G2_parout(par_out23), .enable(data_feed_enable));
        Match_Counter  counteru23(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
, .cnt_en((ser_out[23]^(par_out23[1]^par_out23[3]))~^datain) , .cnt_out(cnt_out23) );

//Satelite No. 24
        Basic_PRN    satu24(    .clk(clk),    .reset_b(reset_b),    .G1_serout(ser_out[24]),
.G2_parout(par_out24), .enable(data_feed_enable));
        Match_Counter  counteru24(  .clk(clk),  .reset_b(reset_b),.cnt_en2(data_feed_enable)
,.cnt_en((ser_out[24]^(par_out24[4]^par_out24[6]))~^datain) , .cnt_out(cnt_out24) );
endmodule
```

For the satellite detection, Fast Simplicity Link (FSL) was used to route the data from the processor to the Cores, including the custom made PRN Core. The digital bit-stream was stored in one ZBT (ZBT0), which was handled by the External Memory Controller (EMC). For the phase matching part two other ZBT memory blocks (ZBT1, ZBT2) were used to contain the look up tables generated by the SIN and COSINE cores. The SIN/COSINE Core was employed in generating the look up tables. The CORDIC Core was used to implement the Arctangent function. Moreover, two Digital Clock Managers (DCM) were used to control the feed through and clock matching of the ZBT interfaces. FSL was also employed in the phase matching section to mediate data transfer from/to different Cores and ZBT. The details of each of these Cores and building blocks are as explained in the next section.

Hardware Description (Description of Blocks)

The detailed description of the hardware implementation of the satellite vehicle detection section is as follows. The schematic of the Cores and interfaces employed are as depicted in figure 4. The block named PRN which is connected through FSL bus to the Microblaze processor in the block diagram, is indeed the satellite detector. A simple C program is developed in order to feed data from Microblaze to the block and read back the generated information from the PRN satellite detector core. Because of "adding a user-designed core" to the Microblaze through FSL bus, the lab instructions of the 5th and 10th lab sessions were very useful in our work.

We were to place the sample data for the block in the ZBT RAM, but we could not implement it completely. The ZBT External memory and its controller, External Memory Controller, in the block diagram show this intention. BRAM interface/Controller is used automatically by XPS to store the instructions and data of the C code. The set of tools to communicate with serial port of the PC is by default mandatory for downloading the assembly of the C code.

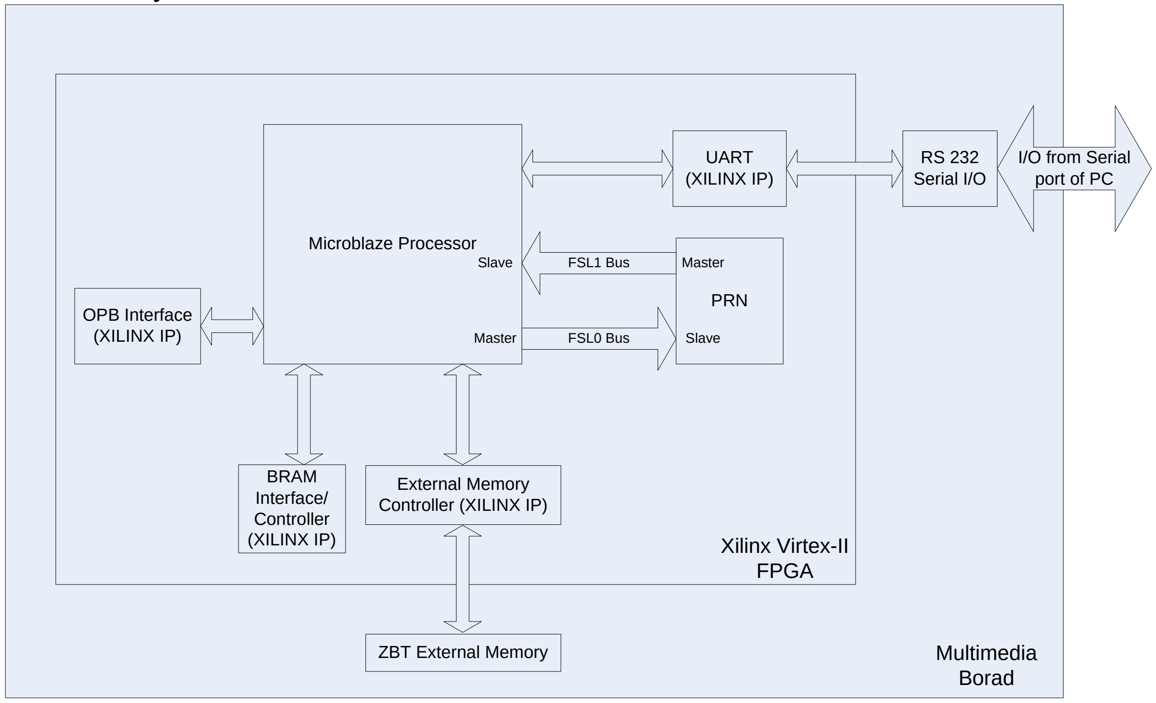


Figure 4

PRN satellite detector core consists of a couple minor building blocks which are mainly developed using Verilog HDL. Figure 2 shows the architecture of the developed core. In fact, this is what we have inside the PRN core in the main block diagram. The dataflow of the architecture is obvious from the direction of the arrows in the figure.

The core has a controller which is in charge of the system control. The incoming data enters from Microblaze through FSL0 bus. This data is 32 bit parallel and is saved into a 32 bit shift register. In the next step shift register starts shifting the data and providing the incoming data in serial for PRN blocks. During the shift process no recent data is read from FSL0.

Each of PRN blocks is a representative of a GPS satellite. It consists of a PRN code generator and a delay generator according to GPS specification. The outputs of these two blocks are XORed with each other again due to the specification. Correlation phase starts from here: the incoming serial data from the shift register is compared to the output of the result using an XNOR gate. Finally, each satellite site has a counter to count the number of matches between the incoming data and the generated PRN core. It should be mentioned that the controller drives the enable signals of shift register, PRN code generators and counters in appropriate times.

The controller receives 32 parallel inputs from FSL0 which comes to 1023 bits. After the completion of receiving 1023 bits, the controller enters a new set of states in which it sends back the output of the system to the Microblaze system.

The output part of the system has a big bus multiplexer. It is a 24 to 1, 10 bits wide multiplexer which is used to feed the outputs of the satellite counters to FSL1 bus. This multiplexer is developed using "XILINX Core Generator". "mux10bit_24to1.v" is made up of BUFTs and is not a registered multiplexer. The controller drives the select signal of the multiplexer along with the write enable of the FSL1 to write the output of the counters in FSL1.

The system is supported with a C code, which is similar to m10 lab code. At the beginning of the code, the program simply writes 32 values to FSL0 bus. These data are used by PRN satellite code to generate the output. After that the code reads back the 24 outputs of the match counters and finds the longest match in the last part of the code. Finally, it prints the satellite which data is come from. All these functions are done using XILINX predefined functions.

PRN satellite detector is tested using Modelsim simulator. The input data is read from a file and applied to the generated codes. The functionality of the whole system along with controller is tested in this test case. The other blocks of the system are tested using individual test benches.

Furthermore, the system is implemented on the XILINX multimedia board and the C code was successfully run on the board. The output of the code for the set of data of the program was 17, which means that the system detects that data is generated from the satellite 17.

Unfortunately, time limitation did not allow us to have a complete test; applying similar data and acquiring the outputs from both simulation and real hardware implementation model and then comparing them . This would be a good verification method for the system.

The SIN/COSINE Core module is used to implement the sine and cosine look up tables. The Core used is Sine/Cosine Look-Up Table v5.0. The CORDIC Core could also have been used, but the SIN/COSINE was chosen due to its simplicity of integration. The Sine/Cosine module accepts an unsigned input value THETA, whose width is from 3-10 bits. We used an 8-bit THETA. THETA is converted from an integer input angle to the radian angle Θ using the following equation:

$$\theta = \mathrm{THETA}\frac{2\Pi}{2^{\mathrm{THETA\_WIDTH}}}\ \mathrm{radians}$$

The Core computes sin(Θ) and cos(Θ) and outputs the results in the two's complement format. The output widths are 4 to 32 bits long, and can be stored in distributed or block memory. In this design the outputs are routed using FSL to the ZBT memories. Sin is stored in ZBT1 and Cosine in ZBT2. The frequency of the generated sin/cosine is set to 10 MHz in accordance with that of the input binary data from the A/D converter of the front end, which is already stored in a ZBT memory, ZBT0. For further detail on how the phase shift is controlled please refer to the individual report.

CORDIC v3.0 Core (Coordinate Rotational Digital Computer) is used in calculating the Arctangent function, employed in determining the phase shift. CORDIC has the capabilities to calculate Vector Rotation (polar to rectangular), Vector Translation (rectangular to polar), sinusoid and cosine, hyperbolic sin and cosine, Arctangent and hyperbolic Arctangent, as well as the square root function. The block diagram of the CORDIC is as shown in figure 5.

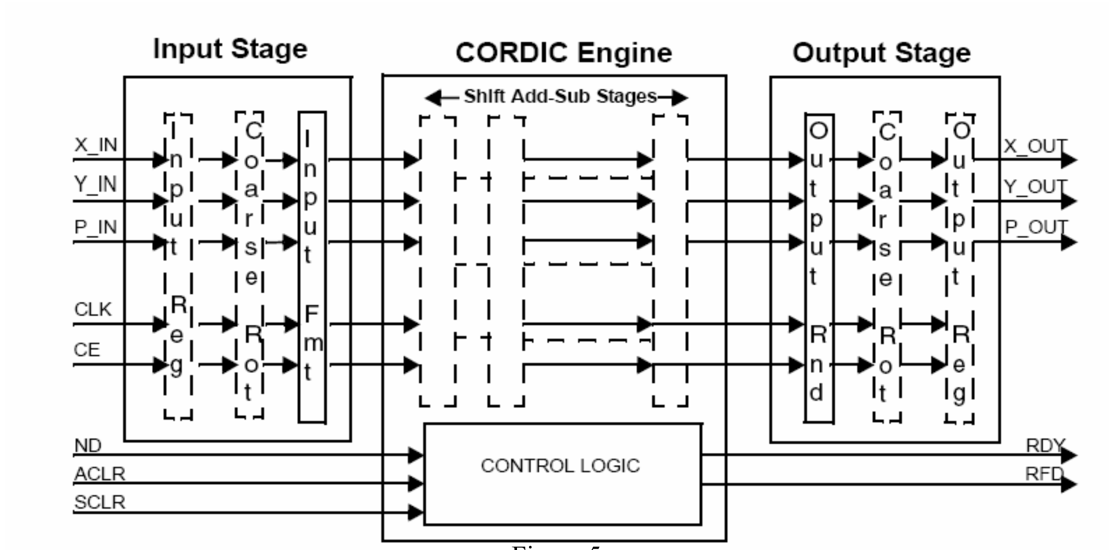


Figure 5

When the Arc Tan functional configuration is selected, the input vector (X,Y) is rotated (using the CORDIC algorithm) until the Y component is zero. This generates the output angle, Atan(Y/X). The three signal (I/O) X_IN, Y_IN, and P_OUT are used. X_IN, Y_IN are expressed as signed binary numbers of 1QN format, and P_OUT is expressed as signed binary 2QN format. Phase data formats are in Radian and Pi Radian. For the details of the implementation of 1QN and 2QN please refer to the individual report. According to the 1QN formatting, the limits for the X_IN and Y_IN are +1 and -1, where as the limits for the P_OUT are from –Pi to +Pi. The way the Arctan is incorporated is that the output of the two correlation counters, one for Quadrature and one for In phase components, are applied as X_IN and Y_IN respectively and the output of P_OUT is used as the input to the bilinear transform block which determines the applicable phase shift to the sin and cosine cores. Due to time constraints and many other feasibility issues many problems were faced during the implementation of the bilinear transform control block (which is purely a mathematical equation block), and as a result the phase detection algorithm wasn't fully implemented in the hardware. However, the sin/cosine and CORDIC cores were implemented and the outputs were successfully stored in look up tables of ZBT1 and ZBT2. For the implementation of the ZBT memory interface, the EMC (External Memory Controller) module receives instructions from the OPB to read and write to external memory devices. The controller provides basic read/write control signals and the ability to configure the access times for read, read-inpage, write, and recovery times when switching from read to write or write to read. Three ZBT were incorporated with the following addresses:

MEM0_BASEADDR	0x80600000
MEM0_HIGHADDR	0x807fffff
MEM1_BASEADDR	0x80800000
MEM1_HIGHADDR	0x809fffff
MEM2_BASEADDR	0x80a00000
MEM2_HIGHADDR	0x80bfffff

Since the interfacing with the ZBT incorporated the OPB the instructions in tutorial module m8 available on the website were followed to an extent, and the system was tested using "Dhrystone" ZBT read/write program available on the course website's tutorial sections. For the details of the implementation of the EMC and the ZBT interfacing please refer to the individual report. Once again FSL was planned to be used for the transfer of data among different blocks, which due to difficulties faced wasn't fully functional. The structure employed was tested to be functional in the last minutes, but its correctness and validity wasn't checked thoroughly. For a thorough test, the simulation results of the MultiSim program should have been compared with those generated by the hardware.

Goals Accomplished/Changes

During the course of this project many changes have been made, which originally were not considered due to lack of familiarity with the complexity of the devices used, as well as an underestimation of the amount of work to be handled. Originally the analog data was supposed to be fed from the front end of a commercial GPS receiver system to the Xilinx board for the back-end processing operations. However, the need for a high sampling rate Analog to Digital converter arose. Although the board does have a delta-sigma ADC core, the access to the analog input pins was troublesome. As a result, it was concluded that the digital data of the front end be sampled by a deep memory logic analyzer, and stored on a flash ROM and then fed into the Xilinx board. But, according to the TA's advice (due to the lack of time) interfacing the flash ROM also proved challenging and time consuming. Thus, the last alternative, which is storing the digital bit-stream in onboard ZBT memory blocks, with the computer acting as the mediator was resorted to. The different Cores employed in performing the functions described in the previous sections were implemented properly and were each tested individually to work properly. However, due to the difficulties faced for interfacing with the FSL and using it as a bus control master for routing the data among different cores and blocks, the whole system wasn't successfully integrated together. The FSL system was tested to work in the last minutes, however its correct functionality verification (which could have been done by comparison with MultiSim simulation results) wasn't thoroughly accomplished.

Future Plans/Improvements

The project was not fully realized due to lack of time and other constraints, including the lack of familiarity of the design group members with the hardware and the complexities. The FSL has been implemented but its correct functionality has not been tested thoroughly. Consequently, the immediate goal would be to test the implemented FSL thoroughly by comparing its results with the simulation results obtained from a hardware simulator such as MultiSim. If we were to undertake the project again, we would have broken up the tasks more spread and would have tried to use alternatives other than the FSL. Moreover, we spent much time trying to optimize our correlation function. We could have instead employed the Correlation Core available.

Conclusion

The implementation of the backend of a GPS receiver unit was performed in Xilinx multimedia boards incorporating Virtex II FPGA. The project was a bit short of its promised goals and achievements due to many difficulties/constraints faced during the design and incorporation of different Cores and functional blocks. The two main blocks implemented were the satellite search/detection and the phase matching. The working of the system was initially studied with the Matlab codes written and simulated. At the later phase, the Verilog HDL code, as well as a partial integration of the cores with the ZBT and processor, was accomplished. The designers achieved more familiarity with the Xilinx multimedia board and Virtex II FPGA Cores and building blocks. At first they attempted to implement each function employed themselves from scratch, with Verilog HDL, but upon the completion of the project they have realized that many of them are already available to be used by Xilinx and they will apply them in their future designs and endeavors.

References

[1] E. D. Kaplan, Editor, Understanding GPS principles and applications, Artech House Publishers, 1998.