Fast Arbitrary Waveform Generator using FPGA

Kyle Romero

IEEE # 80306278

April 2009

E.E.4334-001

Project 25

Instructor: Dr. Zieher

Adviser: John Walter

Abstract

The topic of this paper is a Fast Arbitrary Waveform Generator (AWG). The general design and different operational capabilities of this Arbitrary Waveform Generator are described. Some of the project components that are discussed include: FPGA programming, MATLAB programming, Capture schematic generation, Layout schematic generation, and prototype board creation. Specifically, the Digilent BASYS FPGA board was used to generate an 8 bit digital signal, and a DAC Daughterboard was designed using a TI DAC908 to convert the signal to analog, with an OPA690 providing signal amplification with a gain of 2. The FPGA board has an onboard clock at 100 MHz, and the DAC can sample at up to 165 MS/s.

Table of Contents

[5] = Section Referenced from Kyle Romero, Fall 2008 Final Report

List of Figures

[5] = Section Referenced from Kyle Romero, Fall 2008 Final Report

1. Introduction

An arbitrary waveform generator is a device that allows the creation of any analog signal that is desired by the user. Possible uses for an AWG include testing of devices that measure a signal, such as heart rate, or function generation for testing of circuits.

The main goals for this project were as follows:

1. 8 bit resolution DAC, at least 200 MS/s.

a. Eventually, this output rate needs to approach 2 GS/s.

2. Parallel DAC interface.

3. Initially, calculate waveform on PC and load to FPGA.

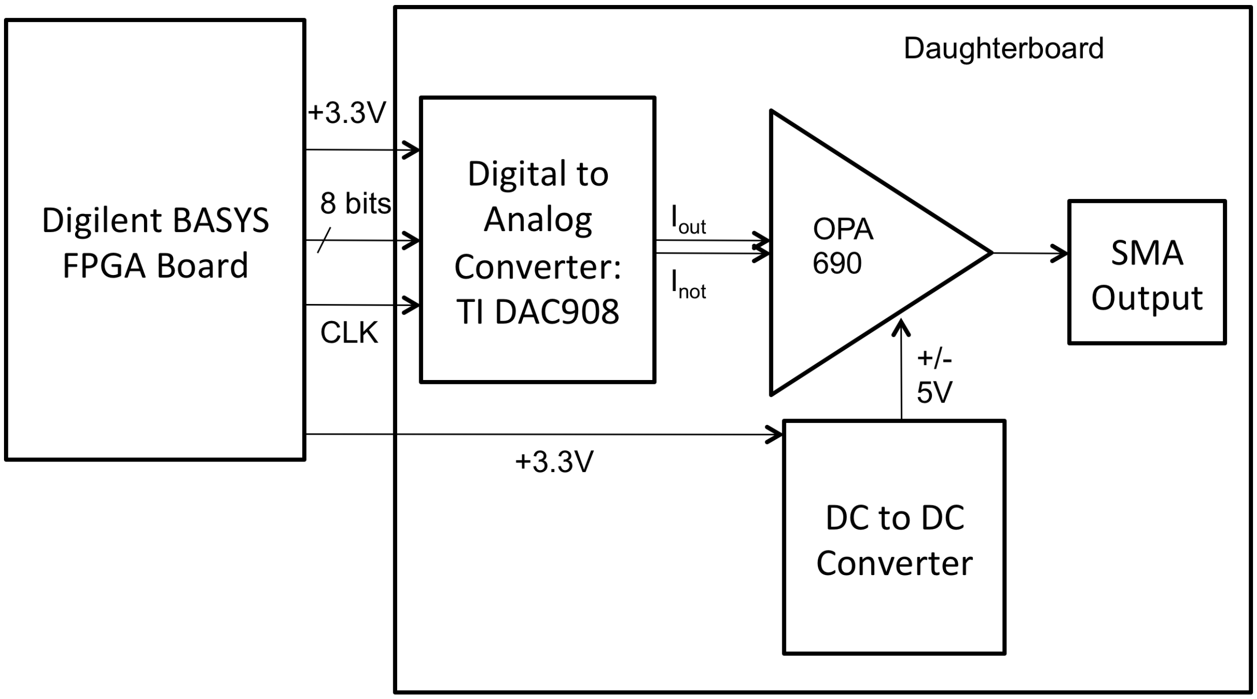
a. Later on, calculate waveform internally.

In the following sections, the different components of the AWG as well as the operation of each component will be described.

2. Components

Several discrete components make up the arbitrary waveform generator, including both software and hardware. These include:

- MATLAB program for creating Waveform File
- VHDL programming for the Xilinx Spartan 3E
- Digilent BASYS FPGA evaluation board
- DAC daughter board (Prototype and Production)



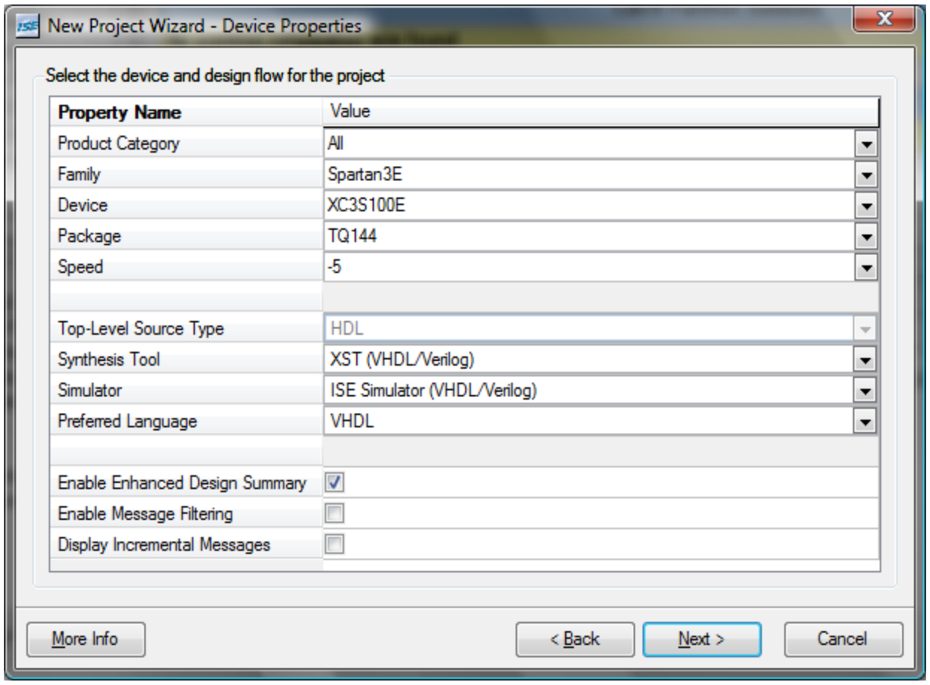
• Figure 1: System Block Chart

i. The Software

Programming of the FPGA encompassed a large part of the project. The FPGA was programmed using the Xilinx ISE programming environment in combination with a waveform text file generated using MATLAB. The programming language chosen to model the hardware was VHDL. Verilog was also considered, but due to greater flexibility, VHDL was chosen. The first part of the software that should be examined is the VHDL program.

a) VHDL Program [5]

As previously stated, the Xilinx Spartan 3E FPGA, which resides on the Digilent BASYS evaluation board (which will be examined in more detail in later sections of this paper), was programmed using the Xilinx ISE programming environment. This environment is specifically designed to ease the programming complexity of Xilinx FPGAs. The main thing to ensure when setting up a Xilinx FPGA project is that the program settings are correct. The settings for this specific project are illustrated in the following picture.


New Project Wizard - Device Properties
Select the device and design flow for the project
More Info
< Back
Next >
Cancel

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S100E
Package	TQ144
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Preferred Language	VHDL
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

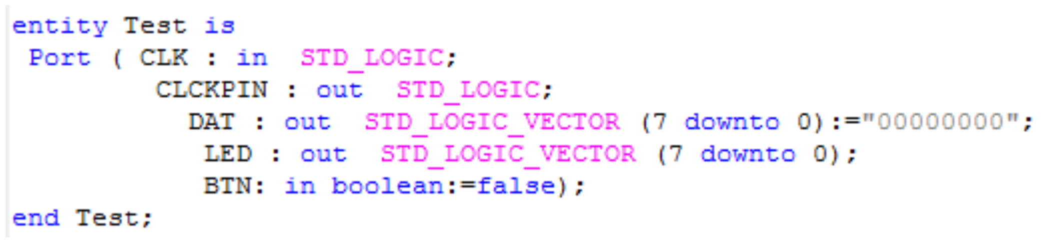
Figure 2: Xilinx Project Setup [5]

As can be seen, VHDL is chosen as the selected language, the Spartan 3E is the selected FPGA target, and the built in Synthesis and Simulation tools are selected.

The VHDL program is designed to do several different things. First, at compile time (.bit file creation), it reads in a user defined waveform text file and stores it internally in a block RAM configured as a ROM. After the file is compiled and loaded onto the FPGA, the program will output the contents of the ROM (the waveform) onto the data bus pins. In order to visually ensure that the program is working, a delay has been linked to Button 1 of the BASYS board, and the LEDs mirror what is being outputted onto the data bus pins. So essentially, if you want to see if the program is working correctly, simply hit button 1, and the LEDs will output the waveform at a rate visible to the human eye. Also, an additional feature of the VHDL is the ability to synthesize a new clock signal. This allows the data to be outputted to the DAC daughterboard at a rate faster than allowable using the internal clock source.

The actual VHDL program consists of several parts, which will now be discussed.

Part 1: Port Definitions [5]



```
entity Test is
 Port ( CLK : in  STD_LOGIC;
          CLCKPIN : out  STD_LOGIC;
            DAT : out  STD_LOGIC_VECTOR (7 downto 0):="00000000";
             LED : out  STD_LOGIC_VECTOR (7 downto 0);
             BTN: in boolean:=false);
end Test;
```

Figure 3: Port Definitions

As you can see in the above illustration, 5 ports are defined in this program:

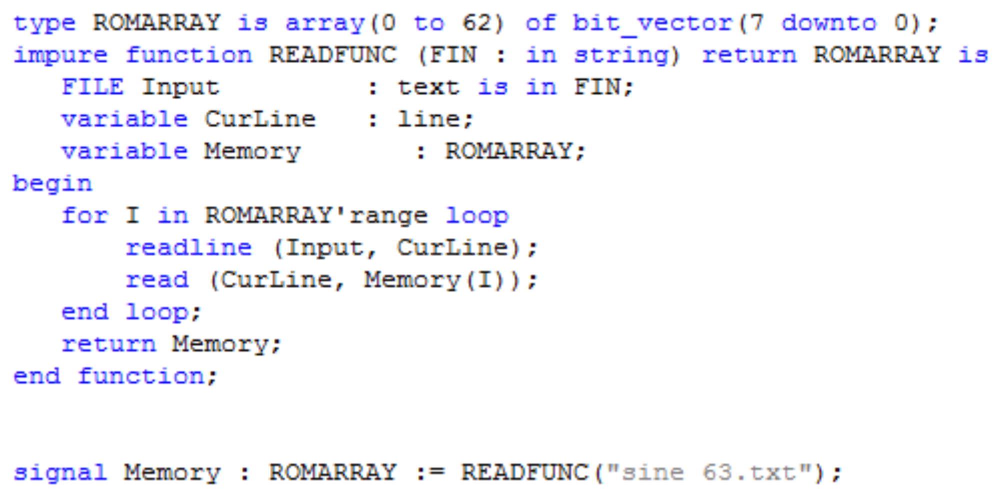
1. CLK – The input clock.
2. CLCKPIN – The output clock pin.
3. DAT – The Data Bus
4. LED – LED output pins
5. BTN – Button 1 on the BASYS Board

A port of type STD_LOGIC_VECTOR is a vector containing a Binary Number String, such as "01010101". The Binary number string is important because this is how the waveform to be generated is stored. To define the bit resolution of the vector, you tell it what the highest and lowest bit number is. Because this project is designed to be 8 bit resolution, the logic vectors are defined as follows:

STD_LOGIC_VECTOR(7 downto 0);

A port of type STD_LOGIC simply contains a “0” or “1”. In the case of this program, it is used to connect to the clock signal. Also, a port of type Boolean will produce a “true” if an input signal is detected and is used to create the delay button functionality.

Part 2: Behavioral Architecture [5]



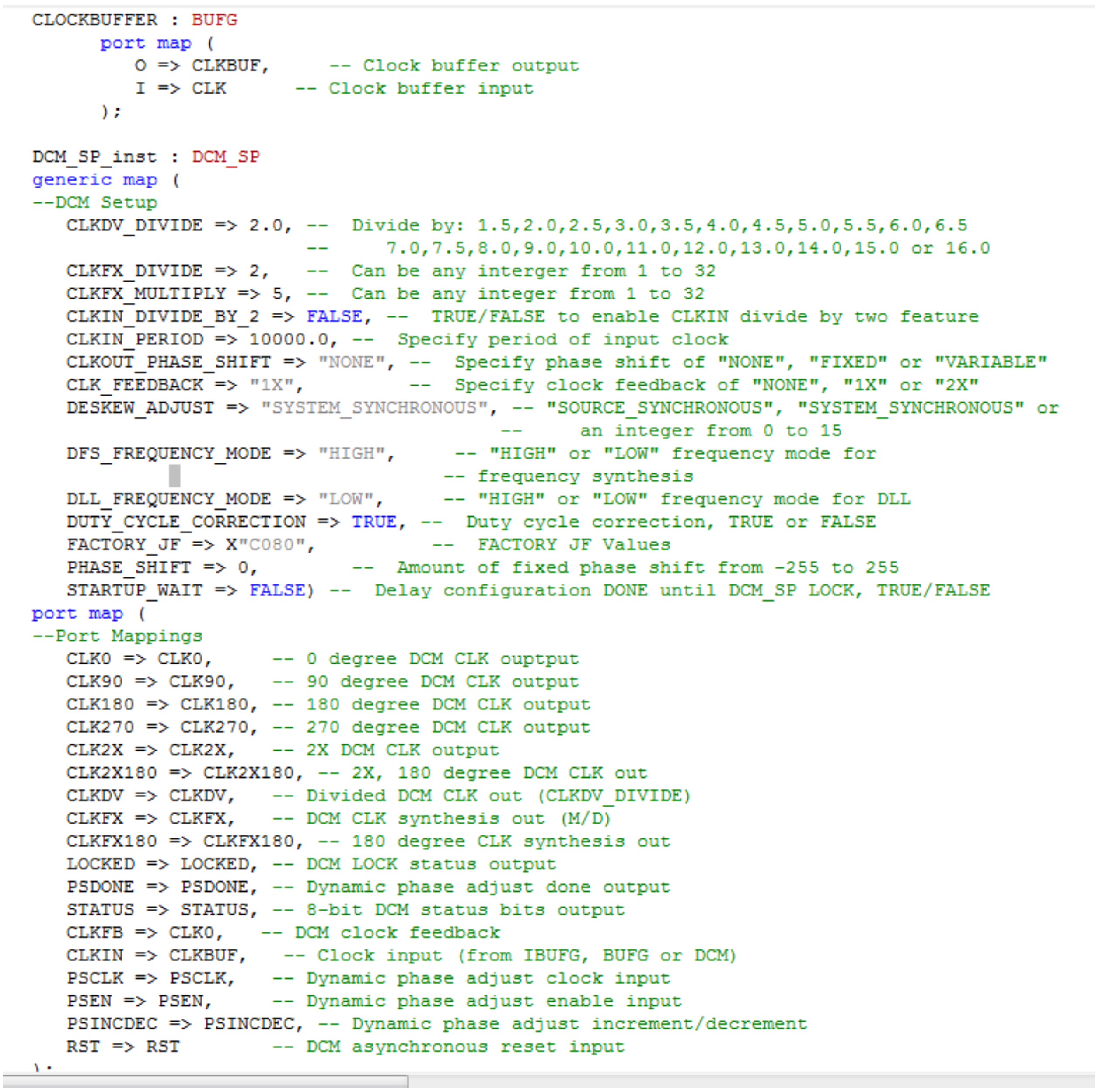
```
type ROMARRAY is array(0 to 62) of bit_vector(7 downto 0);
impure function READFUNC (FIN : in string) return ROMARRAY is
   FILE Input          : text is in FIN;
   variable CurLine    : line;
   variable Memory        : ROMARRAY;
begin
   for I in ROMARRAY'range loop
       readline (Input, CurLine);
       read (CurLine, Memory(I));
   end loop;
   return Memory;
end function;


signal Memory : ROMARRAY := READFUNC("sine 63.txt");
```

Figure 4: Behavioral Architecture [5]

The behavioral architecture section of the VHDL program controls type definitions. For instance, the type ROMARRAY is defined to be an array with enough space allocated to hold the data points of the waveform that are stored as bit_vectors.

The behavioral section of this program also contains the compile time function that reads in the wave form text file and loads it into memory. The function, READFUNC, takes in a string FIN which defines the name of the text file that contains the waveform. It then reads the file line by line and then it stores the read data in the ROM memory that has been created.

Part 3: Digital Clock Manager and Global Clock Buffer

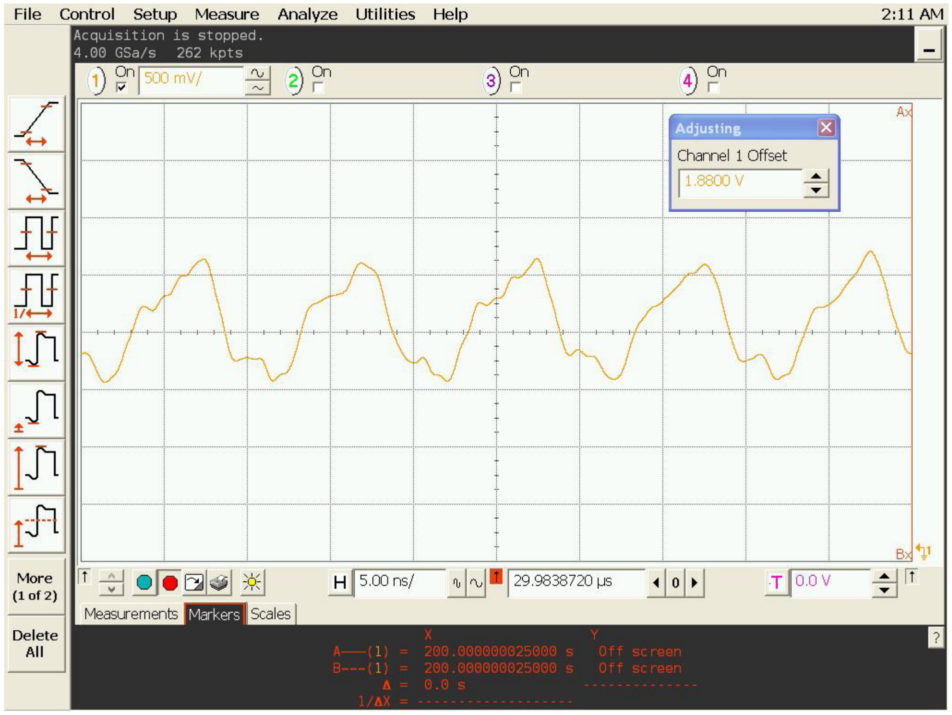


```
CLOCKBUFFER : BUFG
      port map (
         O => CLKBUF,     -- Clock buffer output
         I => CLK      -- Clock buffer input
      );

DCM_SP_inst : DCM_SP
generic map (
--DCM Setup
   CLKDV_DIVIDE => 2.0, --  Divide by: 1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0,5.5,6.0,6.5
                        --     7.0,7.5,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0 or 16.0
   CLKFX_DIVIDE => 2,   --  Can be any interger from 1 to 32
   CLKFX_MULTIPLY => 5, --  Can be any integer from 1 to 32
   CLKIN_DIVIDE_BY_2 => FALSE, --  TRUE/FALSE to enable CLKIN divide by two feature
   CLKIN_PERIOD => 10000.0, --  Specify period of input clock
   CLKOUT_PHASE_SHIFT => "NONE", --  Specify phase shift of "NONE", "FIXED" or "VARIABLE"
   CLK_FEEDBACK => "1X",         --  Specify clock feedback of "NONE", "1X" or "2X"
   DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS", -- "SOURCE_SYNCHRONOUS", "SYSTEM_SYNCHRONOUS" or
                                          --     an integer from 0 to 15
   DFS_FREQUENCY_MODE => "HIGH",     -- "HIGH" or "LOW" frequency mode for
                                    -- frequency synthesis
   DLL_FREQUENCY_MODE => "LOW",     -- "HIGH" or "LOW" frequency mode for DLL
   DUTY_CYCLE_CORRECTION => TRUE, --  Duty cycle correction, TRUE or FALSE
   FACTORY_JF => X"C080",          --  FACTORY JF Values
   PHASE_SHIFT => 0,          --  Amount of fixed phase shift from -255 to 255
   STARTUP_WAIT => FALSE) --  Delay configuration DONE until DCM_SP LOCK, TRUE/FALSE
port map (
--Port Mappings
   CLK0 => CLK0,      -- 0 degree DCM CLK ouptput
   CLK90 => CLK90,    -- 90 degree DCM CLK output
   CLK180 => CLK180, -- 180 degree DCM CLK output
   CLK270 => CLK270, -- 270 degree DCM CLK output
   CLK2X => CLK2X,    -- 2X DCM CLK output
   CLK2X180 => CLK2X180, -- 2X, 180 degree DCM CLK out
   CLKDV => CLKDV,    -- Divided DCM CLK out (CLKDV_DIVIDE)
   CLKFX => CLKFX,    -- DCM CLK synthesis out (M/D)
   CLKFX180 => CLKFX180, -- 180 degree CLK synthesis out
   LOCKED => LOCKED, -- DCM LOCK status output
   PSDONE => PSDONE, -- Dynamic phase adjust done output
   STATUS => STATUS, -- 8-bit DCM status bits output
   CLKFB => CLK0,   -- DCM clock feedback
   CLKIN => CLKBUF,   -- Clock input (from IBUFG, BUFG or DCM)
   PSCLK => PSCLK,    -- Dynamic phase adjust clock input
   PSEN => PSEN,      -- Dynamic phase adjust enable input
   PSINCDEC => PSINCDEC, -- Dynamic phase adjust increment/decrement
   RST => RST         -- DCM asynchronous reset input
);
```

Figure 5: Clock Buffer and Digital Clock Manager

The digital clock manager and global clock buffer are library components used to synthesize a new clock frequency. The original clock signal is passed into the global clock buffer, which provides a low skew signal clock output. This clock buffer output signal is then passed into the DCM, where it is used to synthesize a new clock signal. Various settings in the DCM allows the user to specify what clock output is to be created. The user is able to then use the synthesized clock signal by using CLKFX. There are other signals that can also be used, such as phase shifts, but they were unnecessary for the scope of this project. Currently, the system is set to 250 MHz, which is 2.5x the original clock source.

The output of the clock synthesizer was compared to the output of the original clock source. The results are pictured below:



```
Acquisition is stopped.
4.00 GSa/s   262 kpts
```

1 On 500 mV/

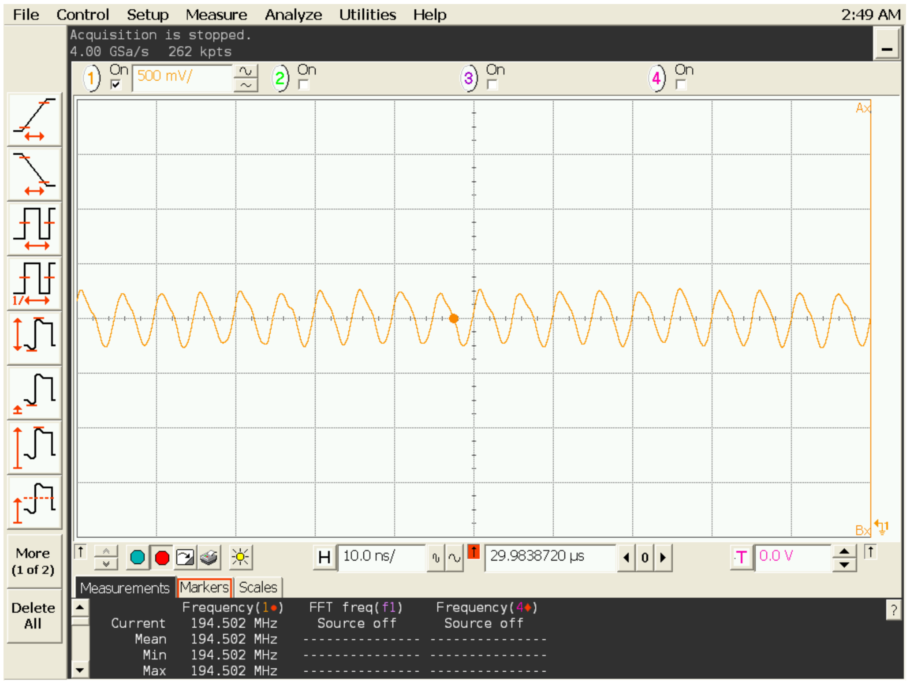
Channel 1 Offset 1.8800 V

H 5.00 ns/ 29.9838720 µs

T 0.0 V

```
              X                       Y
A——(1) =  200.000000025000 s    Off screen
B---(1) =  200.000000025000 s   Off screen
     Δ =  0.0 s               --------------
  1/ΔX = -------------------
```

Figure 6: Original Clock Source

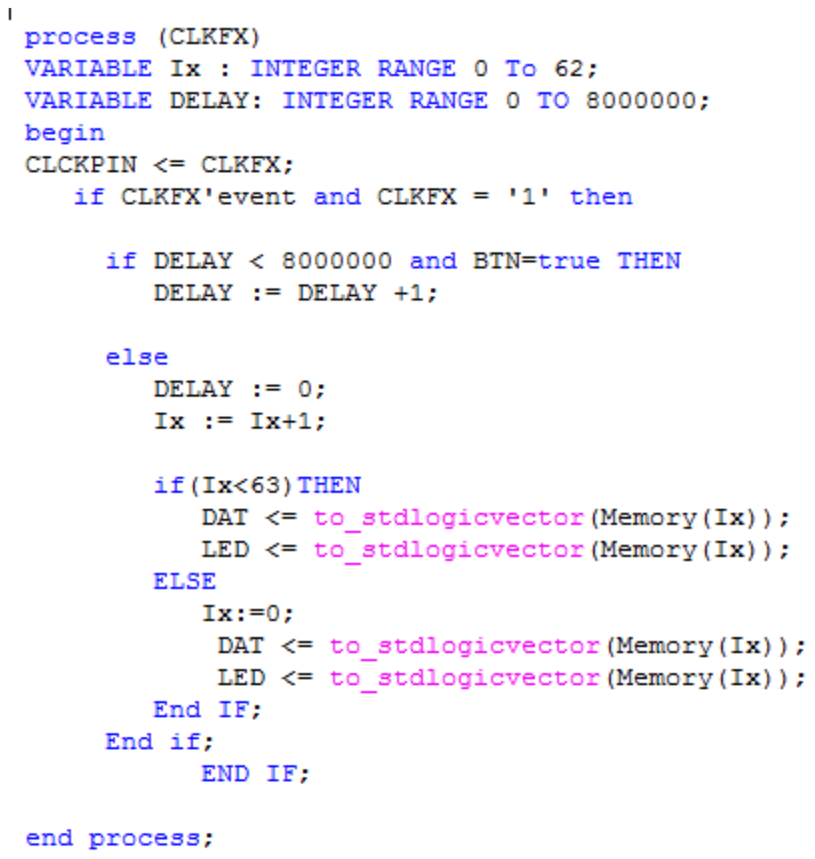


```
Acquisition is stopped.
4.00 GSa/s   262 kpts

          Frequency(1)   FFT freq(f1)    Frequency(4)
Current   194.502 MHz     Source off      Source off
   Mean   194.502 MHz   --------------- ---------------
    Min   194.502 MHz   --------------- ---------------
    Max   194.502 MHz   --------------- ---------------
```

Figure 7: Synthesized Clock Source

As can be seen, the synthesized clock is much more regular and stable than the crystal clock source. Once the clock synthesizer was used, a noticeable improvement in signal performance was noted. The noisy clock on the FPGA is more than likely the source of some noise in the system. However, this is from the crystal itself and cannot be corrected without changing FPGA boards.

Part 4: Process [5]



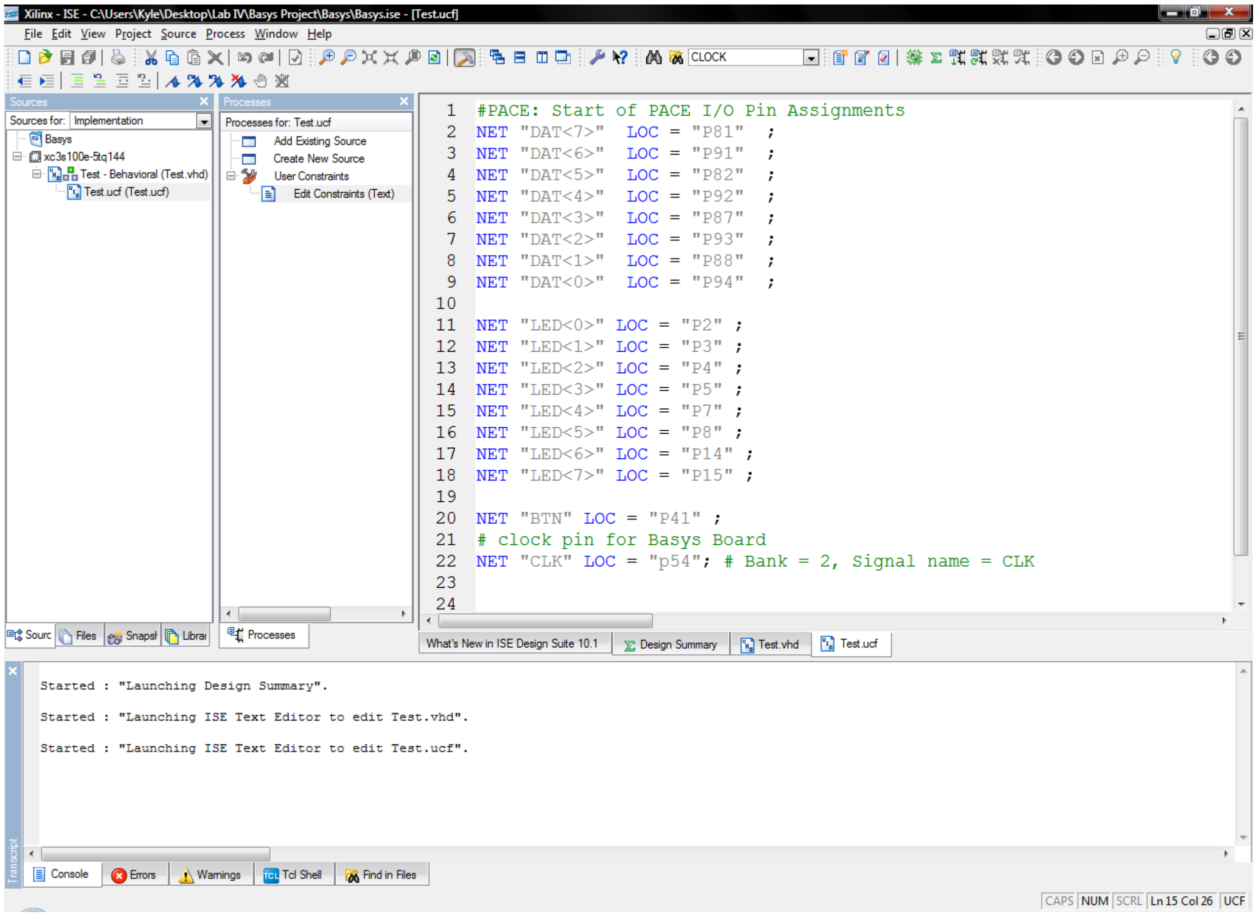
```
process (CLKFX)
VARIABLE Ix : INTEGER RANGE 0 To 62;
VARIABLE DELAY: INTEGER RANGE 0 TO 8000000;
begin
CLCKPIN <= CLKFX;
   if CLKFX'event and CLKFX = '1' then

     if DELAY < 8000000 and BTN=true THEN
        DELAY := DELAY +1;

     else
        DELAY := 0;
        Ix := Ix+1;

        if(Ix<63)THEN
           DAT <= to_stdlogicvector(Memory(Ix));
           LED <= to_stdlogicvector(Memory(Ix));
        ELSE
           Ix:=0;
            DAT <= to_stdlogicvector(Memory(Ix));
            LED <= to_stdlogicvector(Memory(Ix));
        End IF;
     End if;
           END IF;

end process;
```

Figure 8: Process

The Process section of the program is run when the software is loaded onto the FPGA. First, several variables are initialized, Ix and DELAY. Ix is an index number used to track the position in the ROM that is currently being output. It is set in a range from 0 to the size of the ROM. DELAY is the number that defines the speed at which the program runs when the delay button is pushed. In this case, it divides the clock speed by 8 million, slowing the program down to human-visible speeds.

After the variables are defined, the actual function is defined. The first condition tests whether the signal CLKFX (the synthesized clock signal) is high or low. It then tests whether the delay button has been pushed. If it has, it adds in the delay to the system by counting up to 8 million. The program then increments the index number “Ix”, and outputs the signal to the DAT and LED STD_LOGIC_VECTOR ports. The program also resets “Ix” to 0 if it has overflowed the size of the ROM.

Part 5: User Constraints File [5]



```
#PACE: Start of PACE I/O Pin Assignments
NET "DAT<7>"  LOC = "P81"  ;
NET "DAT<6>"  LOC = "P91"  ;
NET "DAT<5>"  LOC = "P82"  ;
NET "DAT<4>"  LOC = "P92"  ;
NET "DAT<3>"  LOC = "P87"  ;
NET "DAT<2>"  LOC = "P93"  ;
NET "DAT<1>"  LOC = "P88"  ;
NET "DAT<0>"  LOC = "P94"  ;

NET "LED<0>" LOC = "P2" ;
NET "LED<1>" LOC = "P3" ;
NET "LED<2>" LOC = "P4" ;
NET "LED<3>" LOC = "P5" ;
NET "LED<4>" LOC = "P7" ;
NET "LED<5>" LOC = "P8" ;
NET "LED<6>" LOC = "P14" ;
NET "LED<7>" LOC = "P15" ;

NET "BTN" LOC = "P41" ;
# clock pin for Basys Board
NET "CLK" LOC = "p54"; # Bank = 2, Signal name = CLK
```

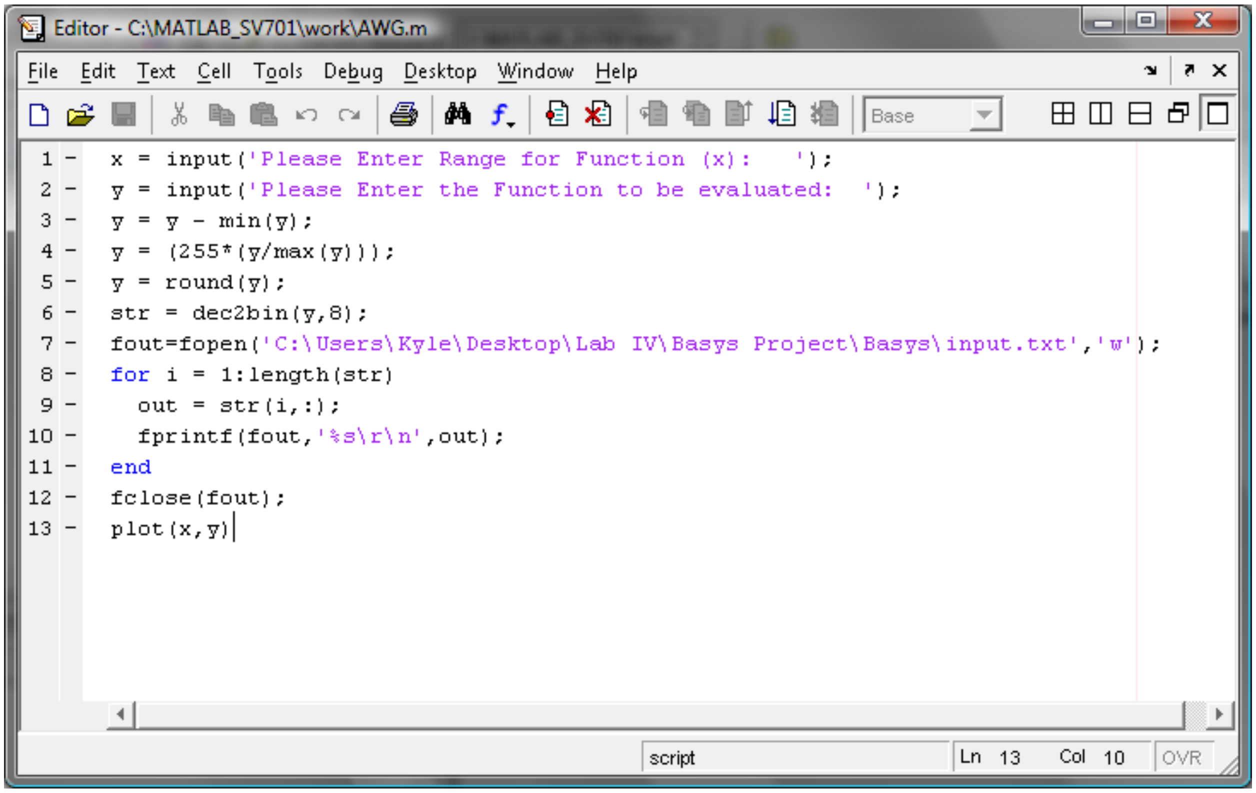
```
Started : "Launching Design Summary".

Started : "Launching ISE Text Editor to edit Test.vhd".

Started : "Launching ISE Text Editor to edit Test.ucf".
```

Figure 9: User Constraints File [5]

The user constraints file (UCF) is simply the file where the actual net list is defined. The net list is links the port name to the physical pins on the board. In this case, each bit of the DAT and LED vectors is set to a different port. Also, all ports used in the software are defined here. These pin settings were taken from documentation provided by Digilent. See Appendix C for BASYS board pin settings.

b) MATLAB Program [5]

The second part of the software that needs to be analyzed is the MATLAB waveform generation program. The code is listed in the following illustration:


Editor - C:\MATLAB_SV701\work\AWG.m
File Edit Text Cell Tools Debug Desktop Window Help
Base
script
Ln 13 Col 10 OVR

```
 1 -  x = input('Please Enter Range for Function (x):   ');
 2 -  y = input('Please Enter the Function to be evaluated:  ');
 3 -  y = y - min(y);
 4 -  y = (255*(y/max(y)));
 5 -  y = round(y);
 6 -  str = dec2bin(y,8);
 7 -  fout=fopen('C:\Users\Kyle\Desktop\Lab IV\Basys Project\Basys\input.txt','w');
 8 -  for i = 1:length(str)
 9 -    out = str(i,:);
10 -    fprintf(fout,'%s\r\n',out);
11 -  end
12 -  fclose(fout);
13 -  plot(x,y)
```

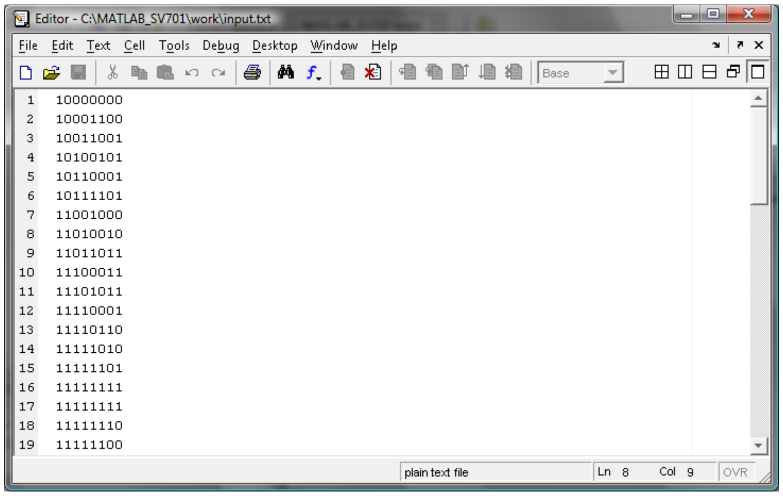
Figure 10: MATLAB Program [5]

The first and second lines of the program receive user input. The program asks for the Range of the function, which determines the number of points generated, and the actual function to be evaluated.

Lines 3 to 6 manipulate the evaluated data points into data usable by the VHDL program. Line 3 shifts all the data points up by the lowest value data pointing, forcing all values greater than or equal to 0. Line 4 then normalizes the data to be between 0 and 255. 255 was chosen because of the 8 bit resolution of the DAC. Because 2^8 = 256, no values above 255 can be outputted using 8 bit resolution.

Line 5 rounds all numbers up so that only integers are present. Finally, line 6 converts the integer values into binary number strings.

Line 7 through 12 opens and outputs the data generated in the previous step to the file location that will be read by the VHDL program. To illustrate what the final file generated is like, the following illustration contains the output for function sin(x) over range 0 to (2 * pi), which is one period of sin(x).


Editor - C:\MATLAB_SV701\work\input.txt
File Edit Text Cell Tools Debug Desktop Window Help
Base
plain text file
Ln 8
Col 9
OVR

```
10000000
10001100
10011001
10100101
10110001
10111101
11001000
11010010
11011011
11100011
11101011
11110001
11110110
11111010
11111101
11111111
11111111
11111110
11111100
```

Figure 11: Sine waveform file [5]

Finally, the last line generates a plot of the data to ensure the proper waveform has been generated.

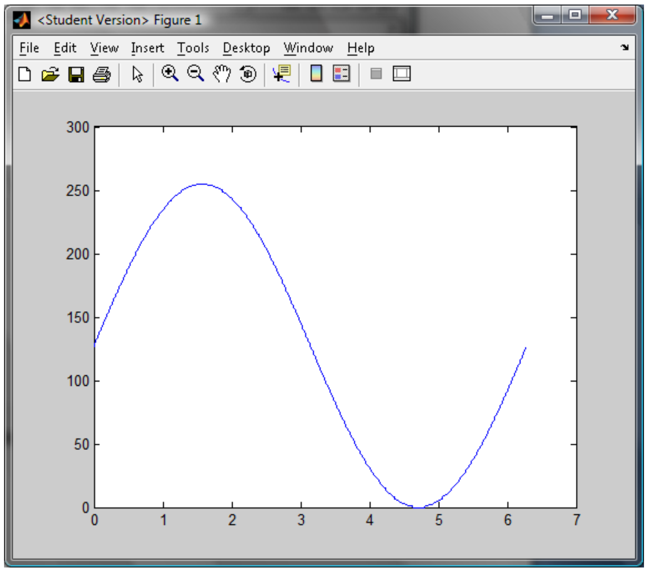


Figure 12: Sine Waveform plot [5]

This concludes the discussion of the software components of the AWG. The following sections will now discuss the hardware components.

3. The Hardware

Another crucial element of the AWG is its hardware implementation. The hardware components are as follows:

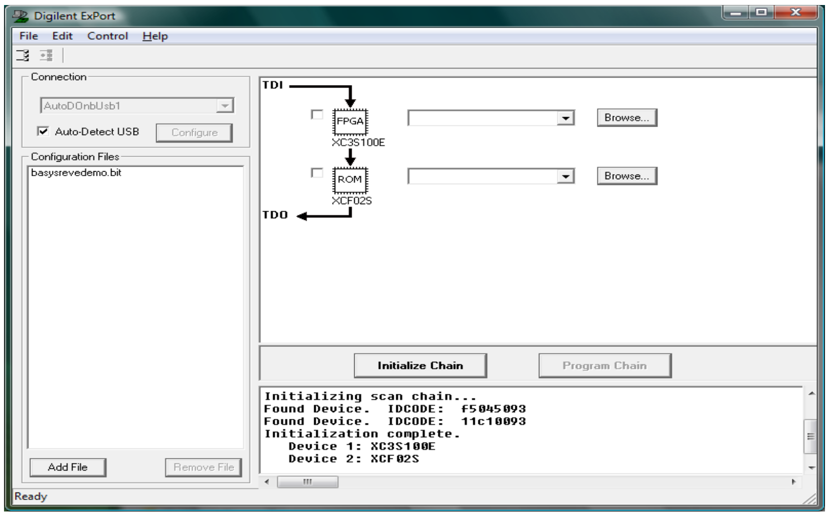
- Digilent BASYS FPGA evaluation board
- Digital to Analog conversion daughterboard
- USB interface to PC

i. BASYS Board [5]

The Digilent BASYS board is designed to test the functionality of the Spartan 3E FPGA. It contains 4 connection headers, each of which contain 4 GPIO pins as well a power and ground pin. The board can be powered from either an external power source or from the included USB cable. The internal logic and output power pins on the board run at 3.3 V.

The BASYS board also features a configurable system clock, which can be set to 25, 50, or 100 MHz by setting a jumper. Other features that the board offers are: 4 general purpose buttons, 8 programmable switches, a reset button, 4 character LCD, PS/2 connection, VGA output, 8 programmable LEDs, and power source selection. See Appendix C for a complete illustration of the board and its pin settings.

Programming the board is a two step process. First of all, the FPGA software must be programmed using the Xilinx ISE as described in previous sections of this paper. After the program has been designed and compiled into a .bit file, it must then be transferred to the FPGA via USB. This transfer is accomplished by using the Digilent Adept software package.



[✓ Auto-Detect USB]

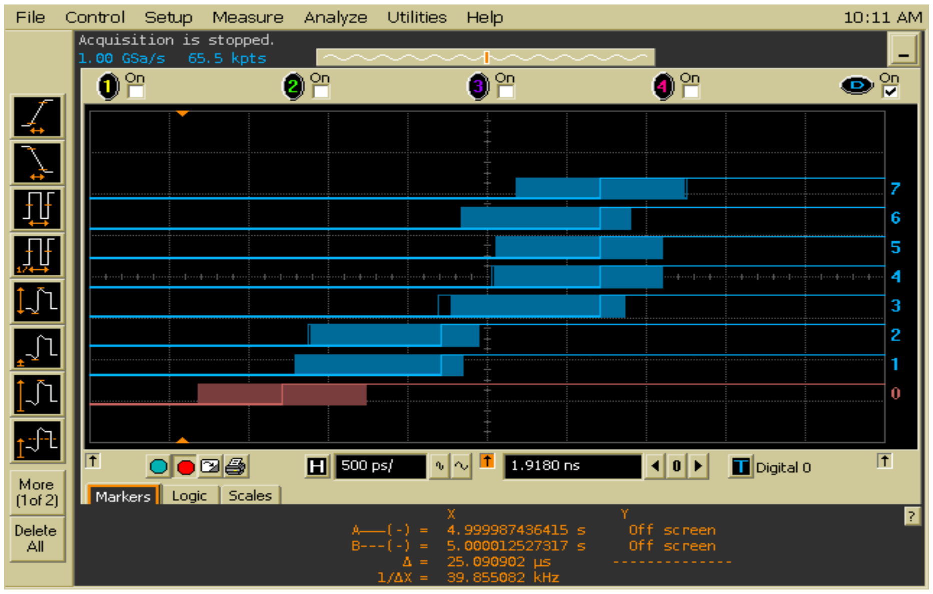
Configuration Files: basysrevedemo.bit

```
Initializing scan chain...
Found Device.  IDCODE:  f5045093
Found Device.  IDCODE:  11c10093
Initialization complete.
   Device 1: XC3S100E
   Device 2: XCF02S
```

Figure 13: Digilent Export [5]

The user may choose whether to program the FPGA or the ROM and browse to choose the appropriate .bit file. Note that a different clock source must be defined in the Xilinx software, based on the location to which you are programming. Also, the FPGA is volatile and must be reprogrammed once power is lost. Once the .bit file has been chosen, Program Chain must be selected and the program will be loaded onto the board. The VHDL program will then be run, and thus the waveform will be outputted to the GPIO pins chosen in the user constraints file.

One main problem that was encountered when using this board was the fact that the GPIO pins were not laid out for high speed output. Thus, a noticeable pin delay is detected at fast output frequencies. The following illustration shows this, as well as highlighting signal length jitter:



```
                 X                        Y
A——(-) =   4.999987436415 s       Off screen
B---(-) =  5.000012527317 s       Off screen
      Δ =  25.090902 µs          ---------------
   1/ΔX =  39.855082 kHz
```

Figure 14: Pin Delay and Jitter [5]

The shaded areas highlight all areas where the signal goes high. Therefore, the signal is not completely stable. Also, a pin delay exists between the first pin going high and the last pin going high. However, this could not be avoided due to the layout of the board, and the jitter and pin delays were not enough to cause serious problems for this project.

ii. DAC Daughterboard

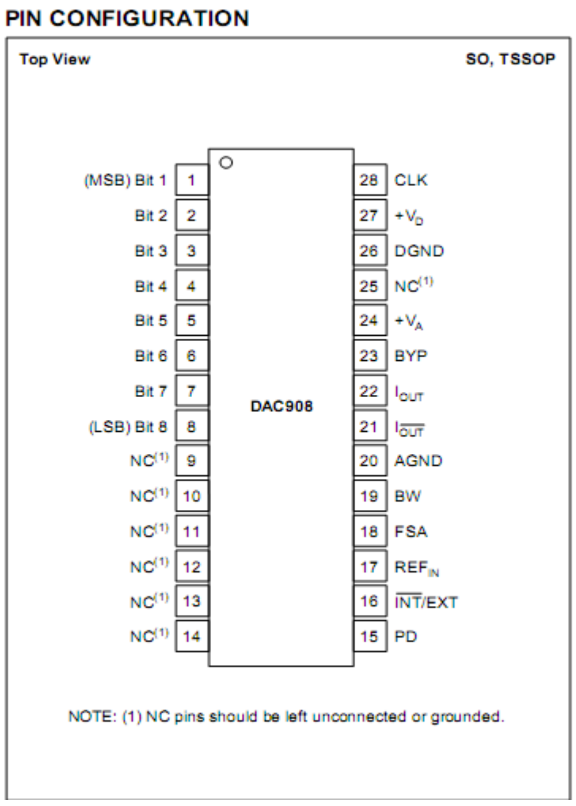
The DAC daughterboard was designed to take the output digital waveform signal from the GPIO pins of the BASYS board and convert it into an analog output signal. Several different components had to be chosen for this board.

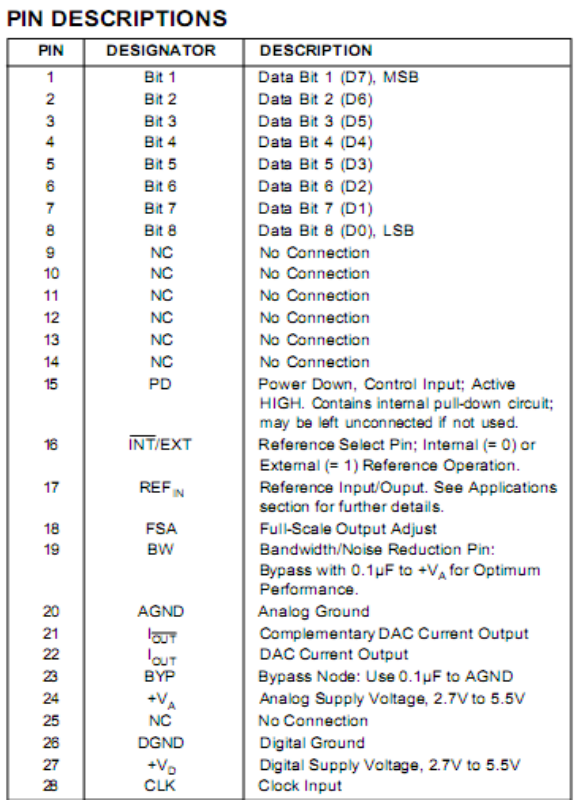
a) The DAC [5]

The Digital to Analog convertor is the most important part of the DAC daughterboard. It is the component that actually converts the signal from digital to analog, as the name implies. The particular DAC chosen was Texas Instruments' DAC908. Some of the features of this particular DAC are:

- 165 MS/S sampling rate
- Settling time of 30 ns
- +3 V or +5 V operation
- Parallel data input
- Scalable differential current output
- Internal or external reference voltage

This DAC was chosen for several reasons. The 165 MS/s sampling rate ensures that it can handle the default 100 MHz output clock speed of the BASYS board. However, it has been tested to work up to 250 MHz overclocked, even though at these speeds, reliability is reduced. Also, it works at 3.3 V and thus can be powered from the header voltage output pins. Finally, the parallel data interface allows an entire data point of the waveform to be input in one more clock cycle. As an added bonus, arbitrary waveform generation is specifically designated in the data sheet.




PIN DESCRIPTIONS

PIN	DESIGNATOR	DESCRIPTION
1	Bit 1	Data Bit 1 (D7), MSB
2	Bit 2	Data Bit 2 (D6)
3	Bit 3	Data Bit 3 (D5)
4	Bit 4	Data Bit 4 (D4)
5	Bit 5	Data Bit 5 (D3)
6	Bit 6	Data Bit 6 (D2)
7	Bit 7	Data Bit 7 (D1)
8	Bit 8	Data Bit 8 (D0), LSB
9	NC	No Connection
10	NC	No Connection
11	NC	No Connection
12	NC	No Connection
13	NC	No Connection
14	NC	No Connection
15	PD	Power Down, Control Input; Active HIGH. Contains internal pull-down circuit; may be left unconnected if not used.
16	$\overline{\text{INT}}$/EXT	Reference Select Pin; Internal (= 0) or External (= 1) Reference Operation.
17	REF_{IN}	Reference Input/Ouput. See Applications section for further details.
18	FSA	Full-Scale Output Adjust
19	BW	Bandwidth/Noise Reduction Pin: Bypass with 0.1µF to $+V_A$ for Optimum Performance.
20	AGND	Analog Ground
21	$I_{\overline{\text{OUT}}}$	Complementary DAC Current Output
22	I_{OUT}	DAC Current Output
23	BYP	Bypass Node: Use 0.1µF to AGND
24	$+V_A$	Analog Supply Voltage, 2.7V to 5.5V
25	NC	No Connection
26	DGND	Digital Ground
27	$+V_D$	Digital Supply Voltage, 2.7V to 5.5V
28	CLK	Clock Input

Figure 15: DAC908 Pin out [1][5]

b) Op-Amp [5]

A differential amplifier was required to amplify the differential current output of the DAC to drive 50 ohms, as well as converting the signal to a voltage signal. The chosen op-amp was the OPA690, also made by Texas Instruments and suggested by the DAC908 datasheet.

c) DC-DC Converter [5]

For the operation of the op-amp, a +/- 5 V supply is recommended for dual rail setups. Therefore, a DC to DC converter was needed to convert the inputted 3.3V to +/- 5 V. The converter chosen for this task is the Murata NTA0305MC.

d) Recommended Setup [5]

The DAC908 datasheet provides details on how to appropriately setup the DAC with a differential Op-Amp to achieve the desired functionality. The recommended setup is illustrated in the following picture:

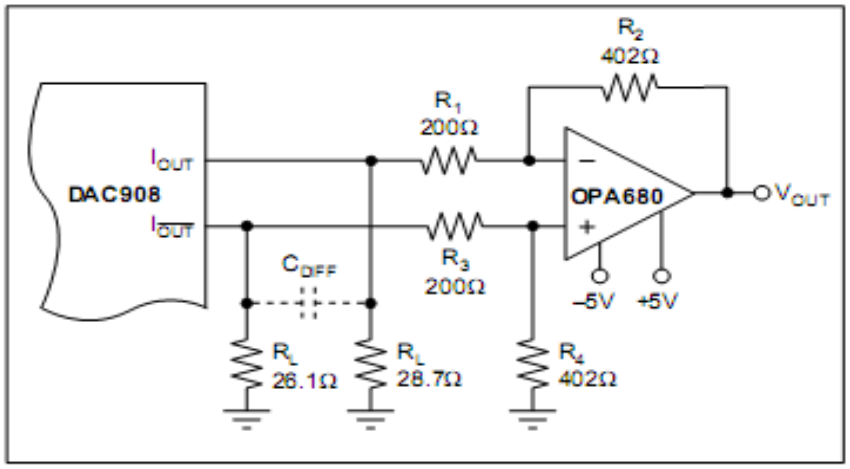
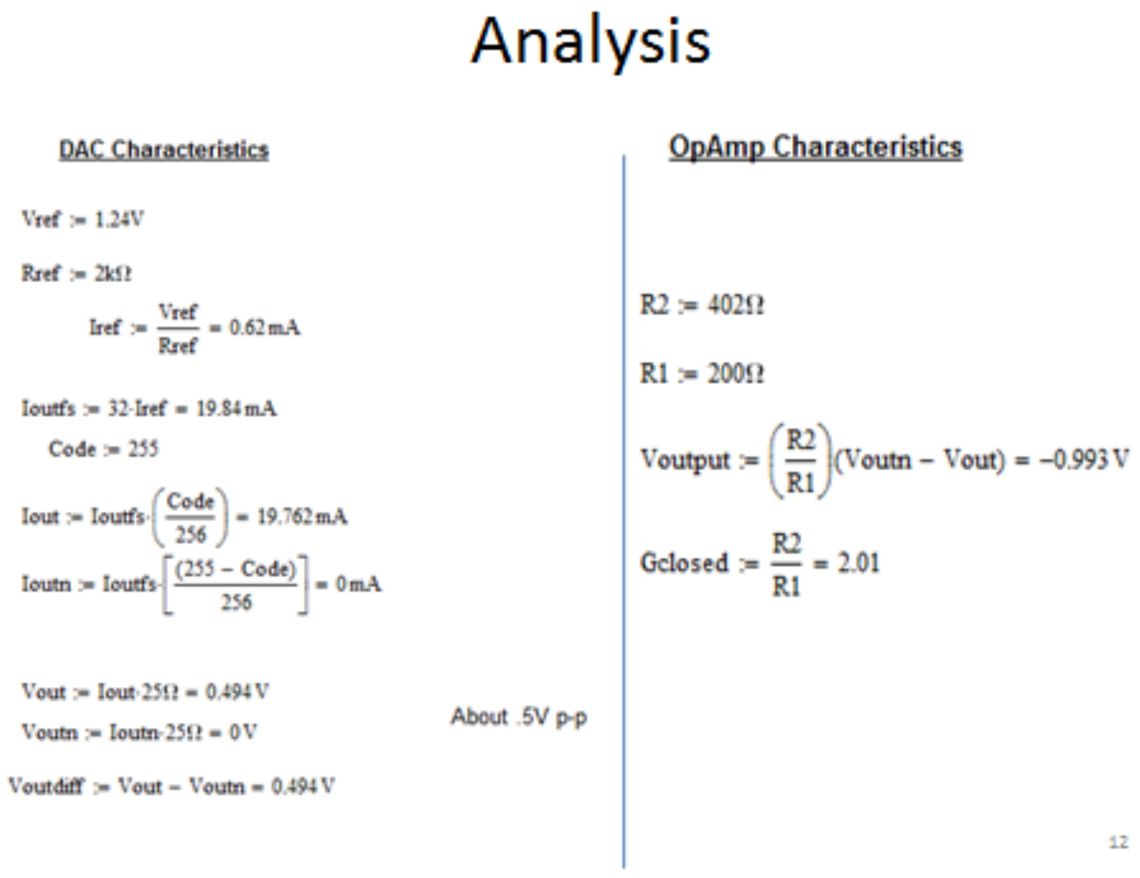


Figure 16: Recommended Setup [5][1]

The datasheet recommends the OPA680. However, this part is no longer in production. The OPA690 is equivalent to the OPA680 and was chosen instead. The op-amp is configured as a differential amplifier with a closed loop gain of ~2, due to the ratio R2/R1. Also, two load resistors have been added in parallel with the input impedance of the OPA690 to bring the impedance on each current output pin to 25 ohms. An analysis of the component selections are provided in the following illustration.


Analysis

DAC Characteristics

$Vref := 1.24V$

$Rref := 2k\Omega$

$Iref := \frac{Vref}{Rref} = 0.62\,mA$

$Ioutfs := 32 \cdot Iref = 19.84\,mA$

$Code := 255$

$Iout := Ioutfs \cdot \left(\frac{Code}{256}\right) = 19.762\,mA$

$Ioutn := Ioutfs \cdot \left[\frac{(255 - Code)}{256}\right] = 0\,mA$

$Vout := Iout \cdot 25\Omega = 0.494\,V$

$Voutn := Ioutn \cdot 25\Omega = 0\,V$

$Voutdiff := Vout - Voutn = 0.494\,V$

About .5V p-p

OpAmp Characteristics

$R2 := 402\Omega$

$R1 := 200\Omega$

$Voutput := \left(\frac{R2}{R1}\right)(Voutn - Vout) = -0.993\,V$

$Gclosed := \frac{R2}{R1} = 2.01$

12

Figure 17: Component Analysis [5]

iii. Prototype Board [5]

Following the recommended design settings for the board, the following schematic and layout were produced for the first prototype board:

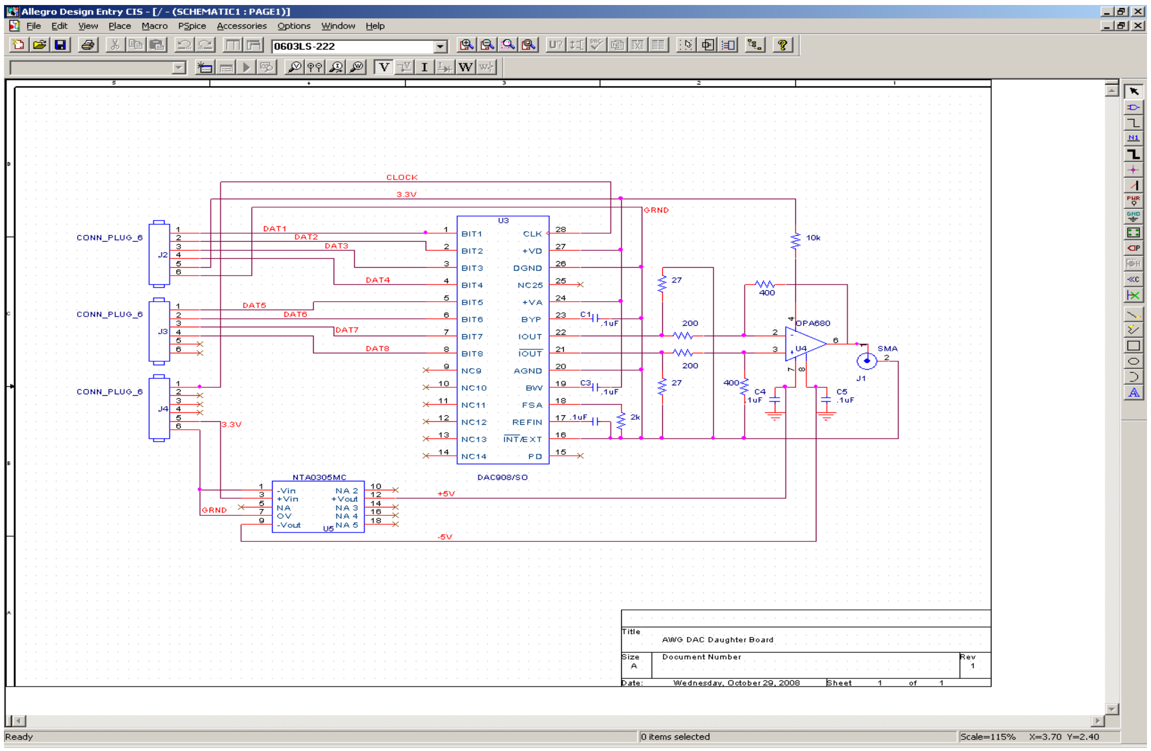
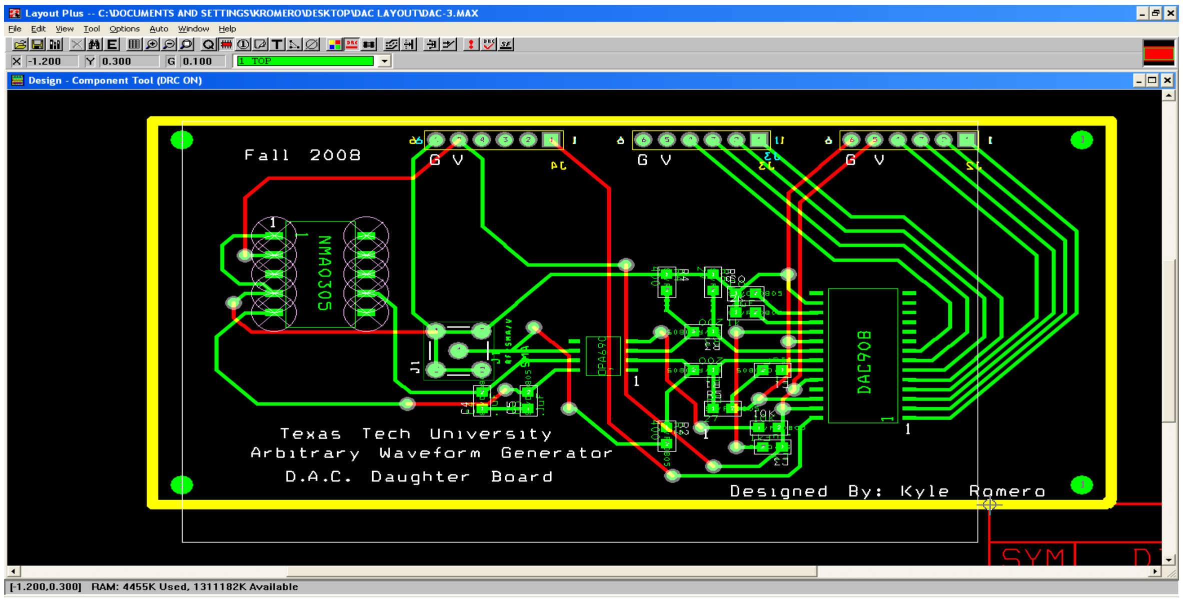


Figure 18: Schematic [5]



Layout Plus -- C:\DOCUMENTS AND SETTINGS\KROMERO\DESKTOP\DAC LAYOUT\DAC-3.MAX
Design - Component Tool (DRC ON)
Fall 2008
NMA0305
DAC908
OPA680
Texas Tech University
Arbitrary Waveform Generator
D.A.C. Daughter Board
Designed By: Kyle Romero
[-1.200,0.300] RAM: 4455K Used, 1311182K Available

Figure 19: Layout (Prototype Board) [5]

The DAC daughterboard was prototyped using the Electrical and Computer Engineering Department’s automated milling machine. This prototype board was then populated using the required components. The following figure is a photograph of the prototyped board.

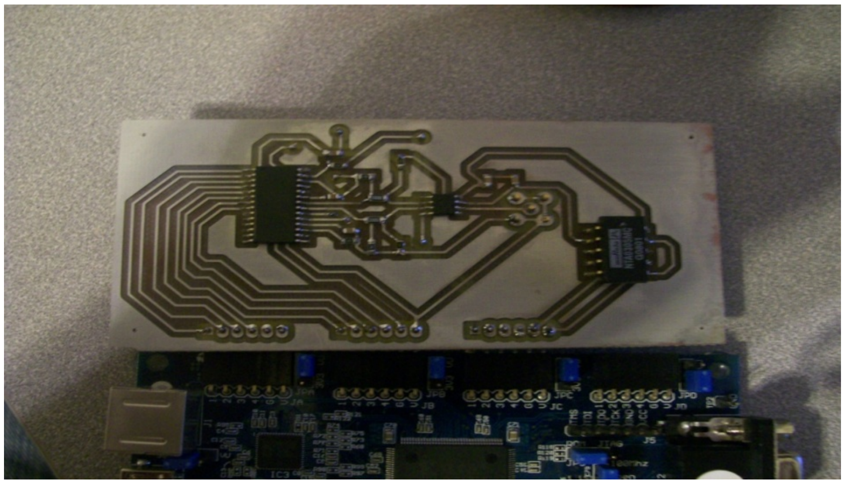
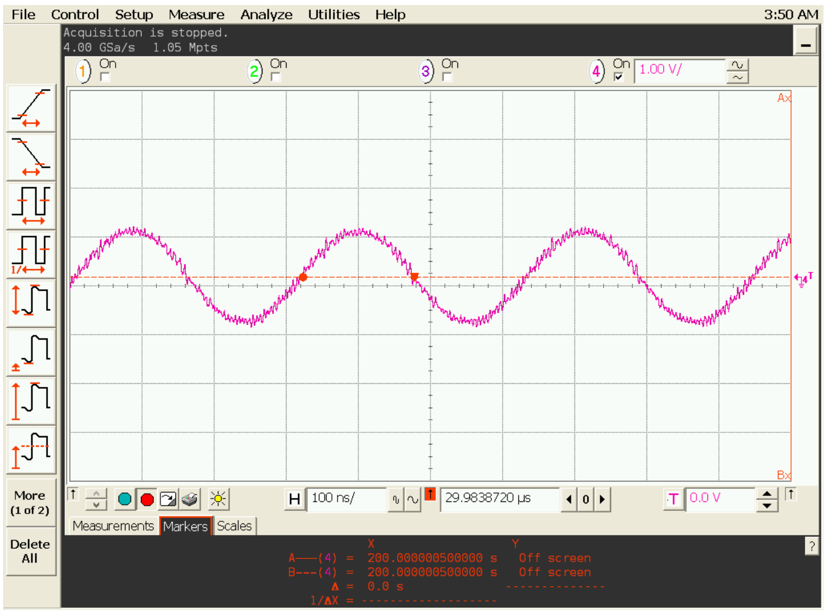


Figure 20: Prototype Board [5]

After correcting numerous errors, the board was brought up to a working status. The following figure shows the output from the DAC Daughterboard (50 ohm load):



```
Acquisition is stopped.
4.00 GSa/s   1.05 Mpts
```

Horizontal: 100 ns/, 29.9838720 µs

```
                  X                         Y
A——(4) =  200.000000500000 s   Off screen
B---(4) =  200.000000500000 s   Off screen
      Δ =  0.0 s              --------------
  1/ΔX = -------------------
```

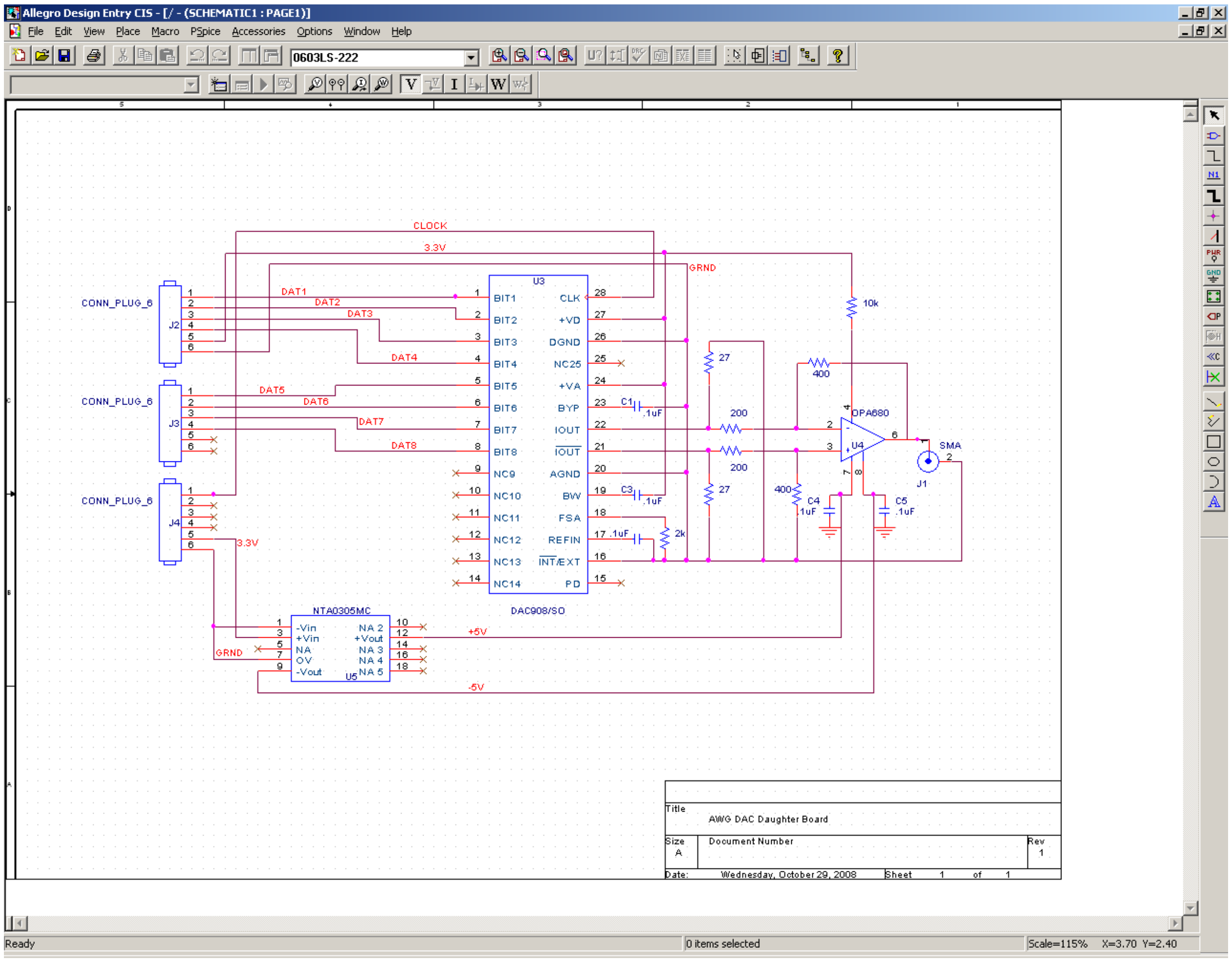
Figure 21: Sine output @ 1.58 MHz (50 Ω Load)

However, due to board design errors, the output of the board was too noisy to be useful.

Therefore, a new board was designed to incorporate corrections to improve board performance.

iv. Production Board

Specific improvements that were made over the prototype board are: separate analog and digital voltage sources, power supply capacitance added, digital and analog ground planes added, and data line lengths were standardized. These changes and additions to the layout were done to reduce overall system noise. The final product, as well as schematic and layout, are pictured below:



Allegro Design Entry CIS - [/ - (SCHEMATIC1 : PAGE1)]
File Edit View Place Macro PSpice Accessories Options Window Help
0603LS-222
CLOCK
3.3V
GRND
CONN_PLUG_6
J2
J3
J4
DAT1
DAT2
DAT3
DAT4
DAT5
DAT6
DAT7
DAT8
U3
BIT1 BIT2 BIT3 BIT4 BIT5 BIT6 BIT7 BIT8
NC9 NC10 NC11 NC12 NC13 NC14
CLK +VD DGND NC25 +VA BYP IOUT IOUT AGND BW FSA REFIN INT/EXT PD
DAC908/SO
C1 .1uF
C3 .1uF
.1uF
2k
27
200
400
10k
OPA680
U4
C4 .1uF
C5 .1uF
SMA
J1
NTA0305MC
-Vin +Vin NA 0V -Vout
NA 2 +Vout NA 3 NA 4 NA 5
U5
+5V
-5V
Title
AWG DAC Daughter Board
Size A
Document Number
Rev 1
Date: Wednesday, October 29, 2008
Sheet 1 of 1
Ready
0 items selected
Scale=115% X=3.70 Y=2.40

Figure 22: Production board Schematic

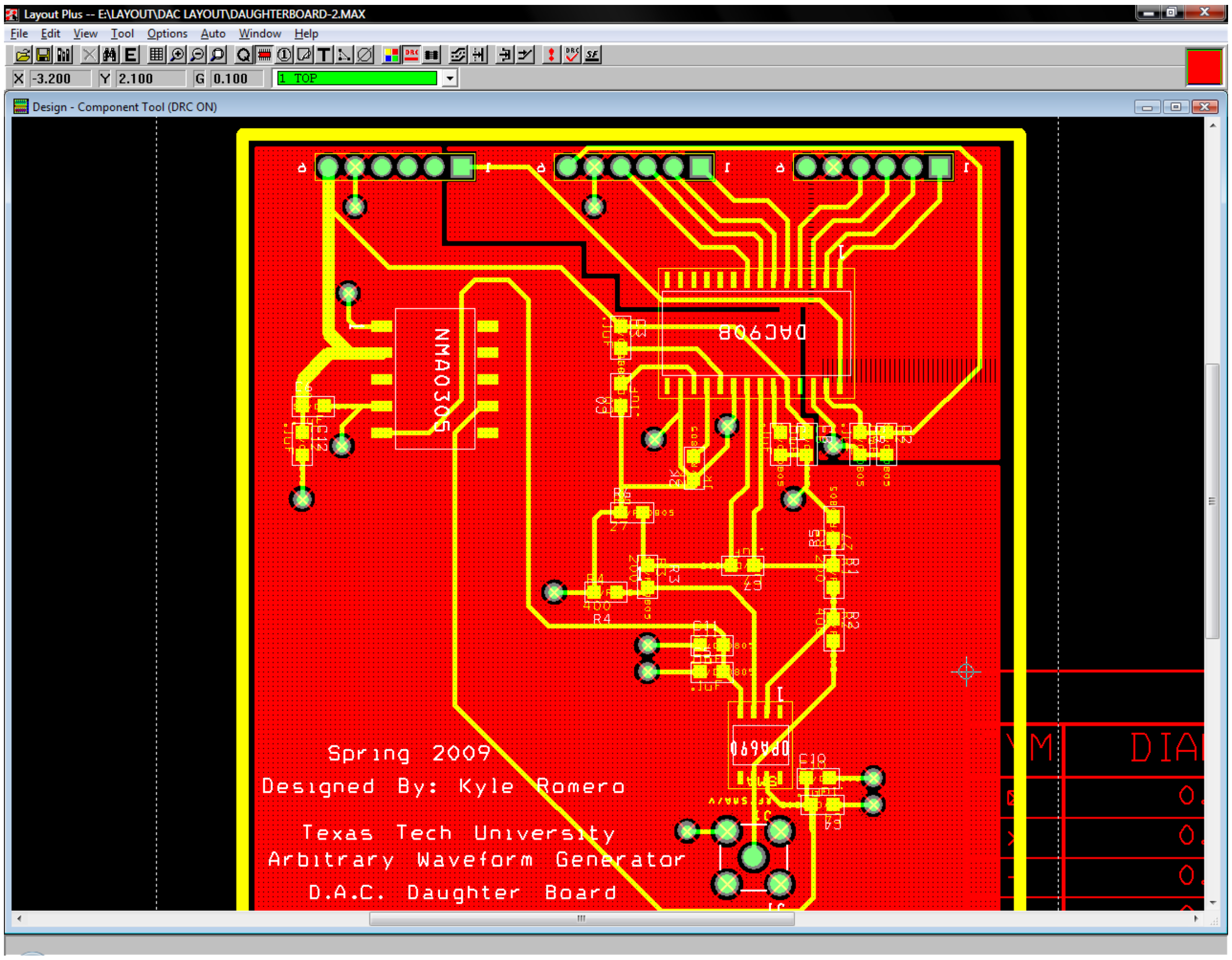

Layout Plus -- E:\LAYOUT\DAC LAYOUT\DAUGHTERBOARD-2.MAX
File Edit View Tool Options Auto Window Help
X -3.200 Y 2.100 G 0.100 1 TOP
Design - Component Tool (DRC ON)
NMA0305
DAC908
OPA690
Spring 2009
Designed By: Kyle Romero
Texas Tech University
Arbitrary Waveform Generator
D.A.C. Daughter Board

Figure 23: Production Board Layout

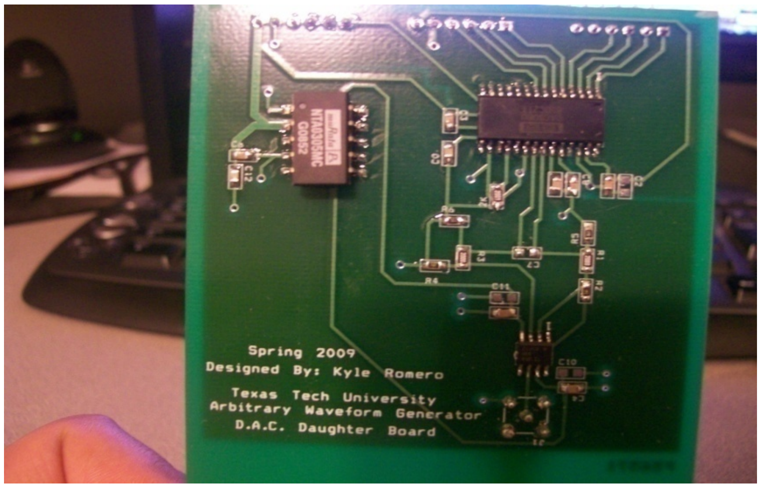
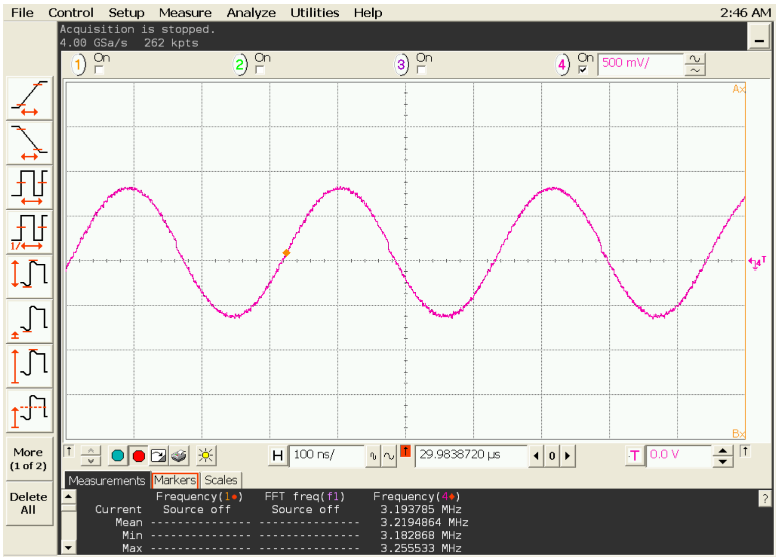


Figure 24: Production Board

Overall, this board performed much better than the prototype board. The signal lacked much of the noise that was found in the prototype board. Also, due to the fact that many corrections were made in the layout before it was ordered, no corrections were necessary on the board after population. An example of the board output is pictured below:



```
Acquisition is stopped.
4.00 GSa/s   262 kpts
```

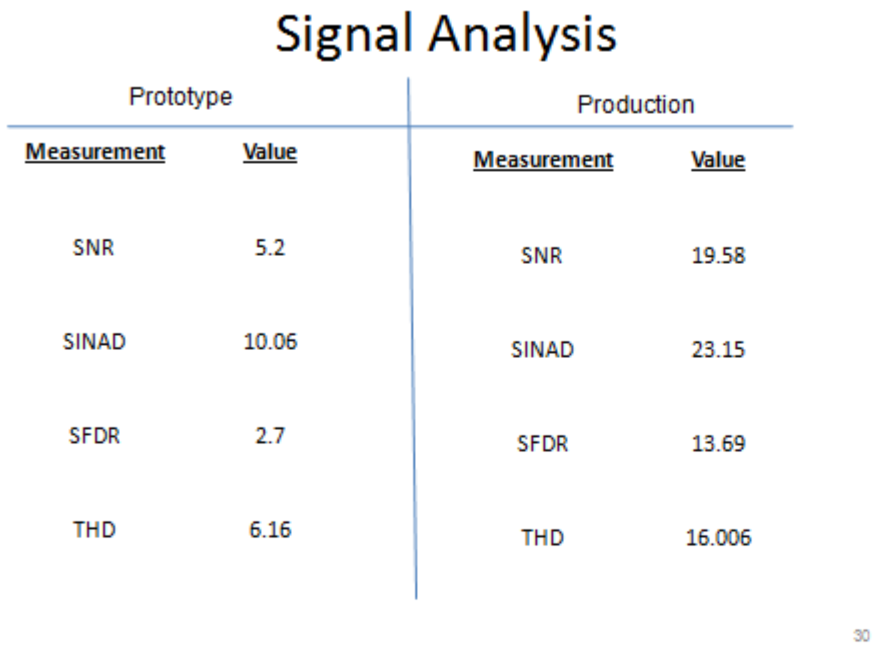
```
              Frequency(1)    FFT freq(f1)     Frequency(4)
    Current   Source off      Source off        3.193785 MHz
       Mean  ---------------  ---------------   3.2194864 MHz
        Min  ---------------  ---------------   3.182868 MHz
        Max  ---------------  ---------------   3.255533 MHz
```

Figure 25: Production board output

As can be seen, this signal is much cleaner than that output by the prototype board.

4. Board Performance

The performance of the DAC daughter boards were measured and analyzed after the production board was successfully populated. The results show that the prototype board does indeed produce a much cleaner output signal. A comparison of the prototype vs. production board performance is listed below. For information on the specific measurements taken, see Appendix.



Signal Analysis

Prototype		Production	
Measurement	**Value**	**Measurement**	**Value**
SNR	5.2	SNR	19.58
SINAD	10.06	SINAD	23.15
SFDR	2.7	SFDR	13.69
THD	6.16	THD	16.006

30

Figure 26: Board Performance Comparison

To make these measurements, a Spectrum analyzer was used. Specifically, measurements were taken of the following power levels: the fundamental frequency, the noise floor, and the 1st harmonic frequency. To see why the production board had an improved output over the prototype board, the next section will discuss high speed board design aspects that were incorporated into the design of the production board.

5. High Speed Board Design

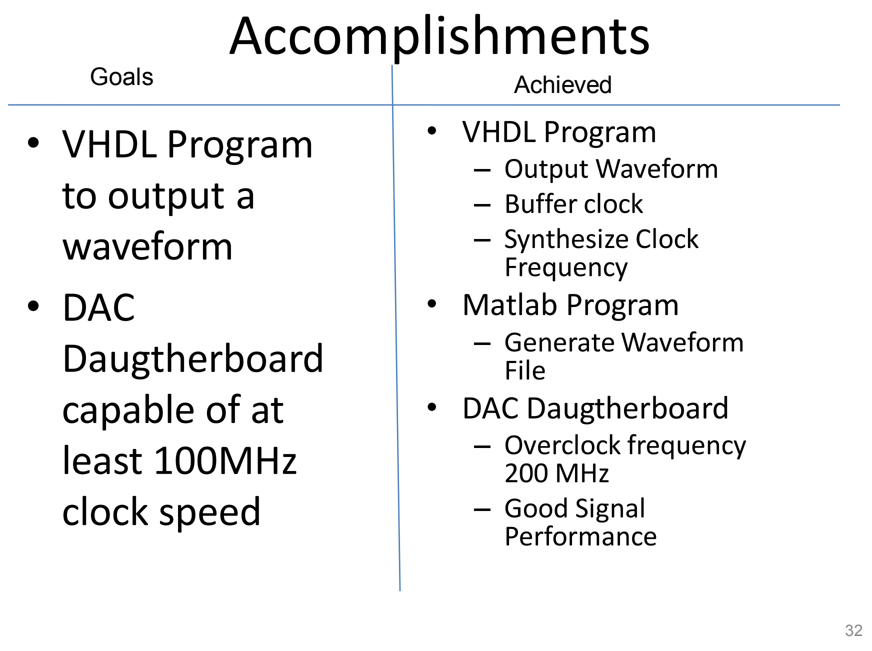
One of the first aspects of high speed design taken into account with the production board design was to minimize signal trace length. This minimizes the distance the signal had to travel, thus preventing noise from being introduced into the signal, as well as preventing inductance/capacitance effects from taking effect in the transmission line. Also, to prevent latency between bits in the digital bus, all signal traces were made roughly the same length. This insures that all bits of the signal will be at the DAC when the clock flips.

A second big consideration when designing the board was the creation of appropriate digital and analog ground planes. The digital plane was placed below all digital lines on the board, and the analog plane was likewise placed below the analog planes. This minimizes current paths that the different current flows must take. Also, it prevents any coupling between the different traces, and removes interference between digital and analog voltage sources. The two planes were connected below the DAC to allow a common ground, and because it was recommended in the data sheet.

One final big consideration in the board design was the proper separation of the digital and analog sources. In the prototype board, the voltage sources for the DAC were connected together. This caused interference in the analog supply because the digital supply was dragging the current down when it was being used, causing unstable voltage levels at the analog supply pin. In the production board, the voltage source pins are sourced differently, causing less noisy voltage supplies. Also, to further stabilize the power signals, decoupling capacitors were connected to the power traces as close to the ICs as possible. This decouples the power system and makes for a more steady power source.

6. Conclusion

All in all, the project was successful. A first generation arbitrary waveform generator was created, and performed reasonably well. Specific accomplishments vs. project goals are pictured below:



Accomplishments

Goals	Achieved
• VHDL Program to output a waveform • DAC Daugtherboard capable of at least 100MHz clock speed	• VHDL Program – Output Waveform – Buffer clock – Synthesize Clock Frequency • Matlab Program – Generate Waveform File • DAC Daugtherboard – Overclock frequency 200 MHz – Good Signal Performance

32

Figure 27: Accomplishments

As can be seen, all stated goals of the project were successfully completed. In addition, several additional features were added to the system, including the ability to over-clock the FPGA board as well as a separate Matlab program to generate a waveform file automatically.

Future goals of the project will mainly focus on improving the DAC sampling rate, as well as improving the overall performance of the system. Also, there should be an emphasis on improving user friendliness for operating the system, for at the moment, the system has a steep learning curve.

Works Cited

1. Texas Instruments, “DAC908.pdf”, DAC908 Data Sheet, October 8, 2008.

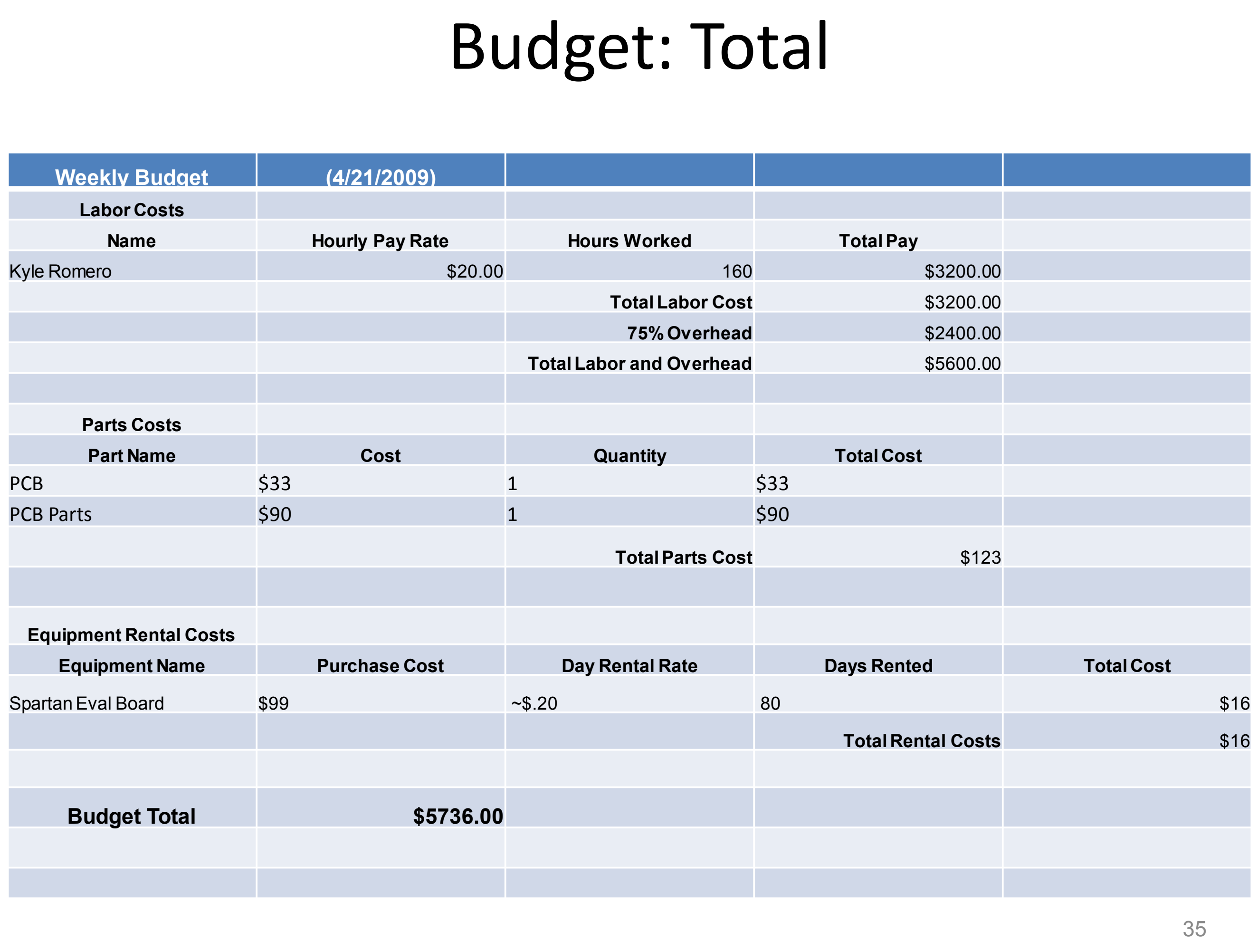
2. Texas Instruments, “OPA690.pdf”, OPA690 Data Sheet, October 8,2008.

3. Murata Power Solutions, “kdc_nta.pdf”, NTA series DC-DC Converter Data Sheet, October 8, 2008.

4. John Walter, Advising Sessions, Fall 2008 – Spring 2009.

5. Kyle Romero, Final Report, Fall 2008

Appendix A: Budget

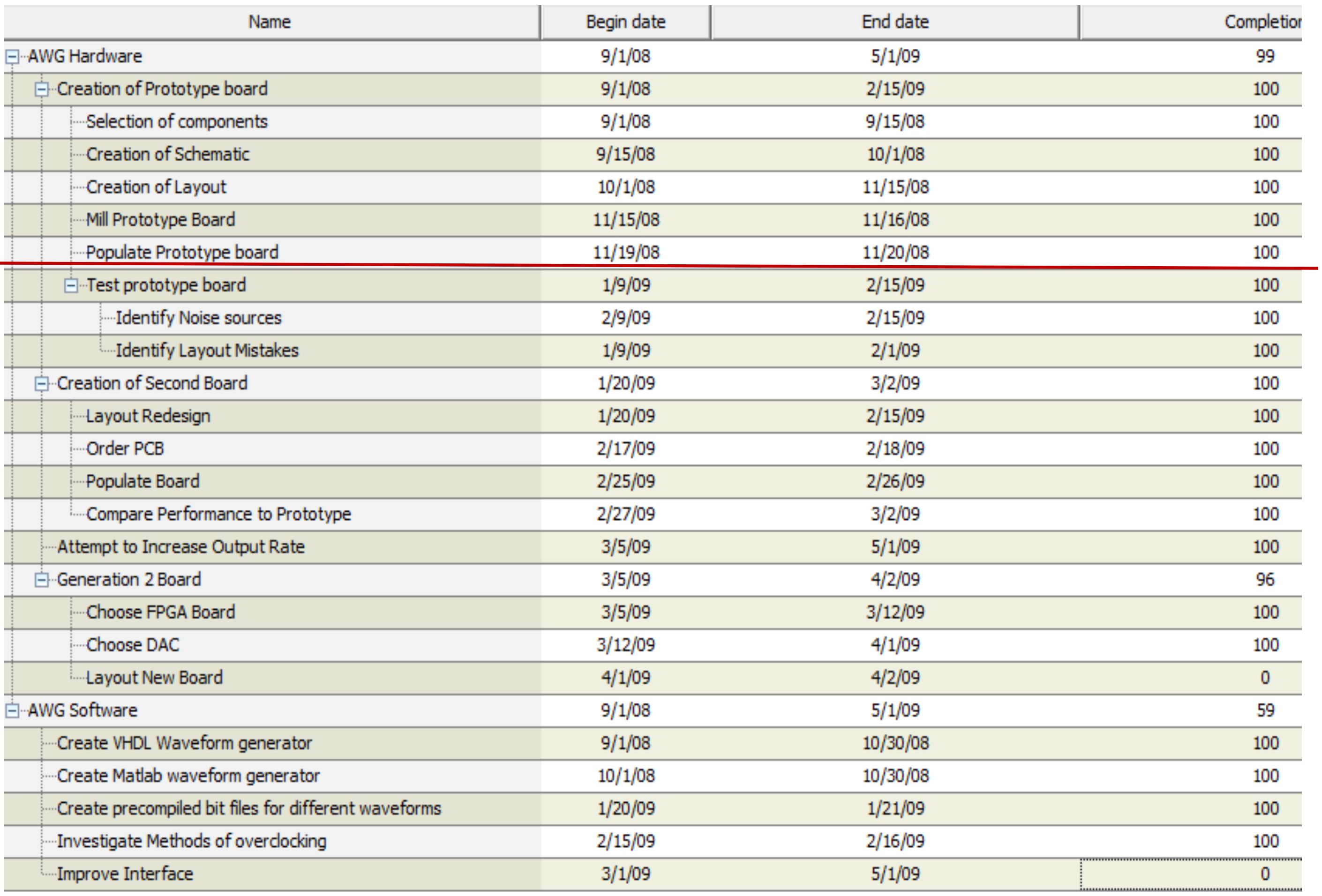


Budget: Total

Weekly Budget	(4/21/2009)			
Labor Costs				
Name	**Hourly Pay Rate**	**Hours Worked**	**Total Pay**	
Kyle Romero	$20.00	160	$3200.00	
		Total Labor Cost	$3200.00	
		75% Overhead	$2400.00	
		Total Labor and Overhead	$5600.00	
Parts Costs				
Part Name	**Cost**	**Quantity**	**Total Cost**	
PCB	$33	1	$33	
PCB Parts	$90	1	$90	
		Total Parts Cost	$123	
Equipment Rental Costs				
Equipment Name	**Purchase Cost**	**Day Rental Rate**	**Days Rented**	**Total Cost**
Spartan Eval Board	$99	~$.20	80	$16
			Total Rental Costs	$16
Budget Total	**$5736.00**			

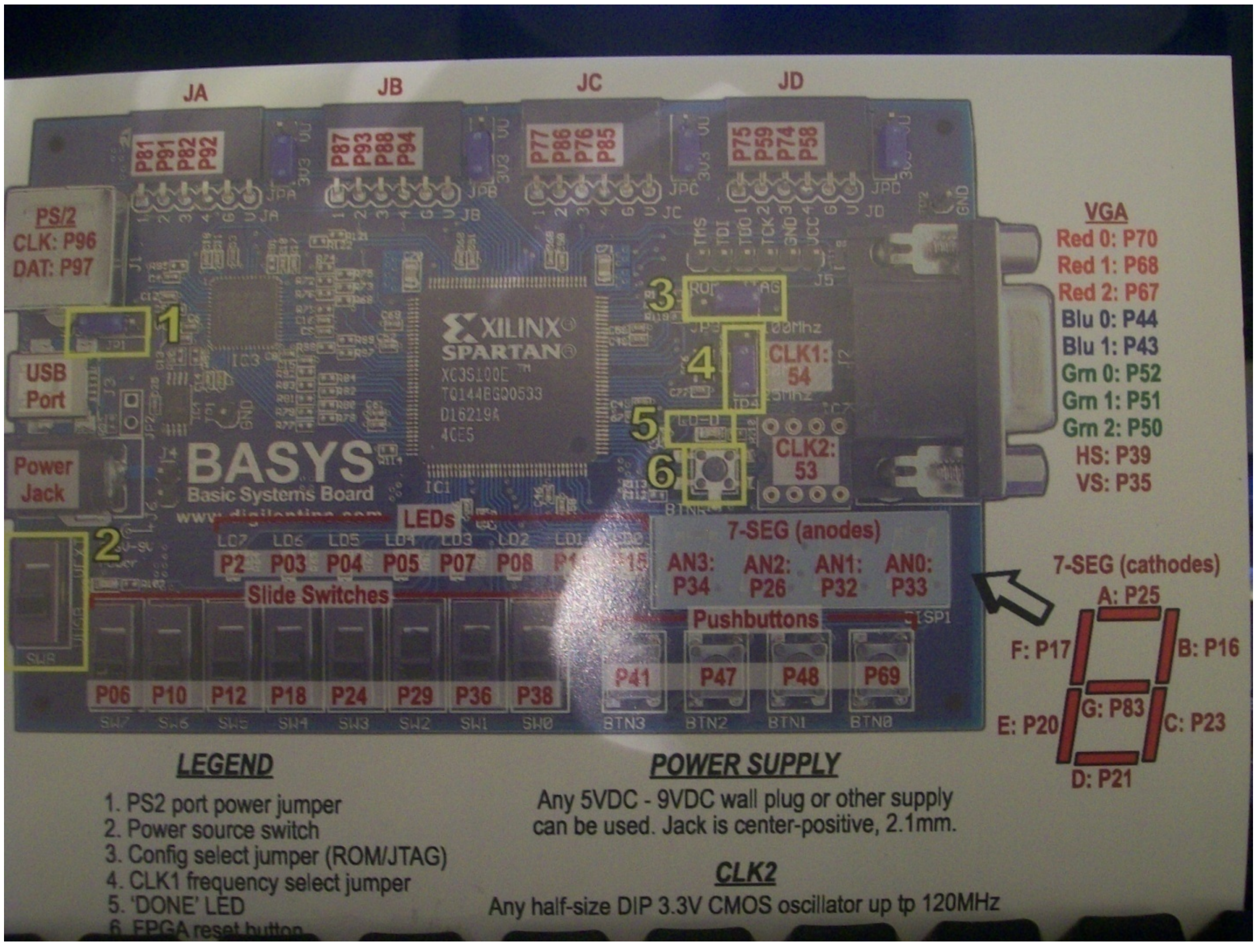
35

Appendix B: Gantt Chart



Name	Begin date	End date	Completio
AWG Hardware	9/1/08	5/1/09	99
Creation of Prototype board	9/1/08	2/15/09	100
Selection of components	9/1/08	9/15/08	100
Creation of Schematic	9/15/08	10/1/08	100
Creation of Layout	10/1/08	11/15/08	100
Mill Prototype Board	11/15/08	11/16/08	100
Populate Prototype board	11/19/08	11/20/08	100
Test prototype board	1/9/09	2/15/09	100
Identify Noise sources	2/9/09	2/15/09	100
Identify Layout Mistakes	1/9/09	2/1/09	100
Creation of Second Board	1/20/09	3/2/09	100
Layout Redesign	1/20/09	2/15/09	100
Order PCB	2/17/09	2/18/09	100
Populate Board	2/25/09	2/26/09	100
Compare Performance to Prototype	2/27/09	3/2/09	100
Attempt to Increase Output Rate	3/5/09	5/1/09	100
Generation 2 Board	3/5/09	4/2/09	96
Choose FPGA Board	3/5/09	3/12/09	100
Choose DAC	3/12/09	4/1/09	100
Layout New Board	4/1/09	4/2/09	0
AWG Software	9/1/08	5/1/09	59
Create VHDL Waveform generator	9/1/08	10/30/08	100
Create Matlab waveform generator	10/1/08	10/30/08	100
Create precompiled bit files for different waveforms	1/20/09	1/21/09	100
Investigate Methods of overclocking	2/15/09	2/16/09	100
Improve Interface	3/1/09	5/1/09	0

Appendix C: BASYS Board Pin Out



LEGEND

1. PS2 port power jumper
2. Power source switch
3. Config select jumper (ROM/JTAG)
4. CLK1 frequency select jumper
5. 'DONE' LED
6. FPGA reset button

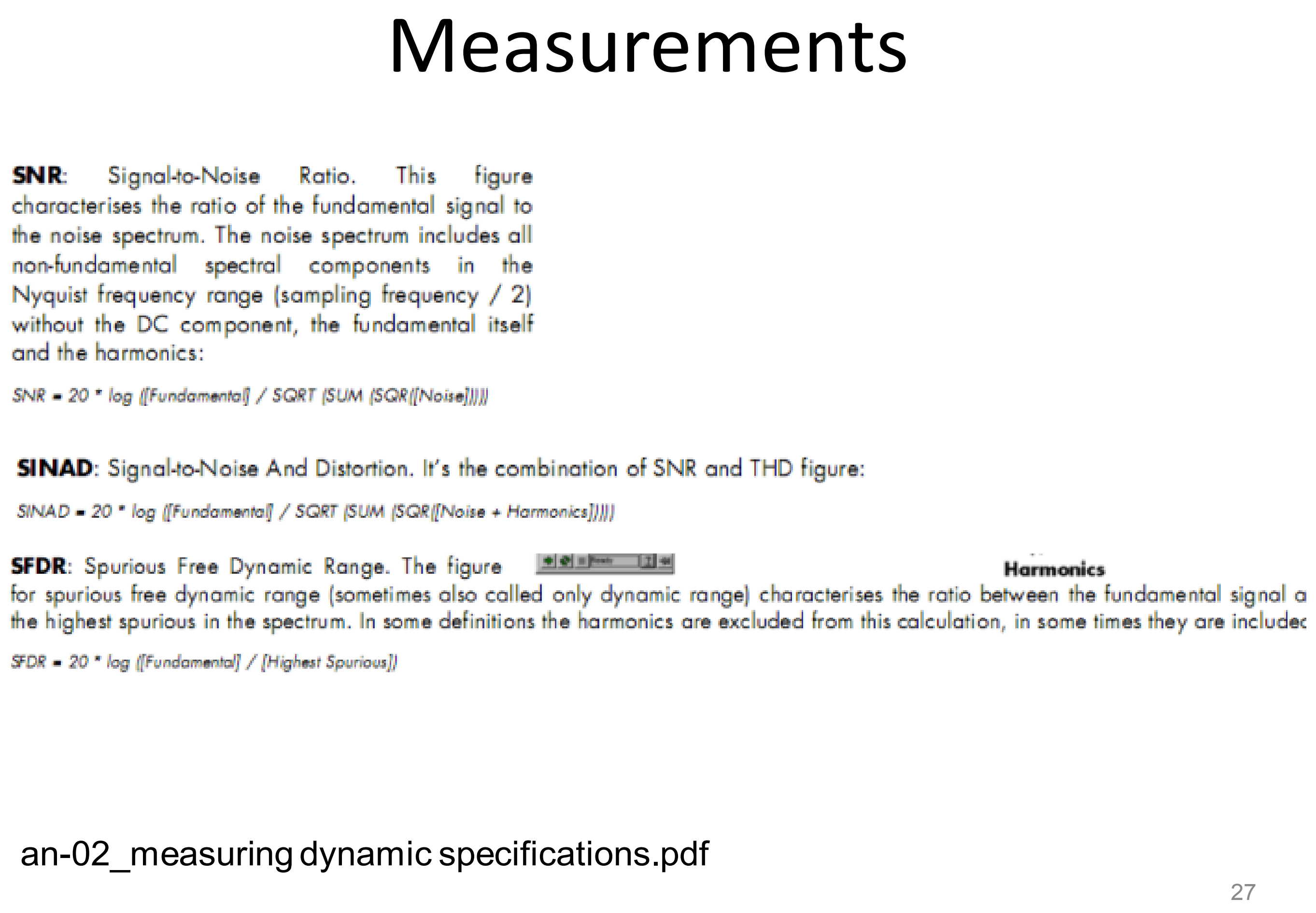
POWER SUPPLY

Any 5VDC - 9VDC wall plug or other supply can be used. Jack is center-positive, 2.1mm.

CLK2

Any half-size DIP 3.3V CMOS oscillator up tp 120MHz

Appendix D: Signal Analysis



Measurements

SNR: Signal-to-Noise Ratio. This figure characterises the ratio of the fundamental signal to the noise spectrum. The noise spectrum includes all non-fundamental spectral components in the Nyquist frequency range (sampling frequency / 2) without the DC component, the fundamental itself and the harmonics:

*SNR = 20 * log ([Fundamental] / SQRT (SUM (SQR([Noise]))))*

SINAD: Signal-to-Noise And Distortion. It's the combination of SNR and THD figure:

*SINAD = 20 * log ([Fundamental] / SQRT (SUM (SQR([Noise + Harmonics]))))*

SFDR: Spurious Free Dynamic Range. The figure **Harmonics**
for spurious free dynamic range (sometimes also called only dynamic range) characterises the ratio between the fundamental signal a
the highest spurious in the spectrum. In some definitions the harmonics are excluded from this calculation, in some times they are includec

*SFDR = 20 * log ([Fundamental] / [Highest Spurious])*

an-02_measuring dynamic specifications.pdf

27

Appendix E: WRITTEN LAB REPORT EVALUATION FORM

Student Name: **Kyle Romero**

Course Number: **4334-001**

Please score the student by circling one of the responses following each of the statements.

1) The student's writing style (clarity, directness, grammar, spelling, style, format, etc)

A B C D F Zero

2) The quality and level of technical content of the student's report

A B C D F Zero

3) The quality of results and conclusions

A B C D F Zero

4) Quality of measurements planned/ taken

A B C D F Zero

Grade: