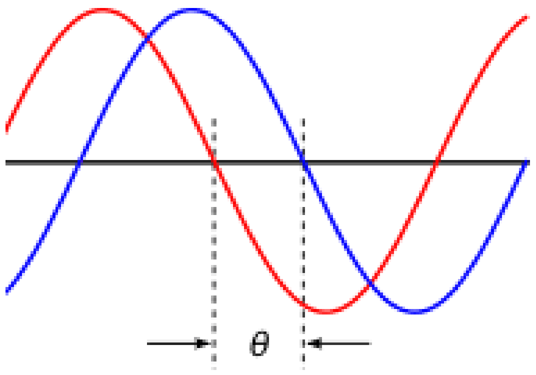
Autumn Semester 2017-18

FREQUENCY INDEPENDENT SINUSOIDAL PHASE SHIFTER

USING SPARTAN-3E FPGA STARTER KIT BOARD



Submitted as project work for completion of Embedded System Course (EEN-360)

Submitted To:

Dr. Vishal Kumar

Associate Professor,

Department of Electrical Engineering,

Indian Institute of Technology Roorkee

Submitted By:

Aviral Singhal (15115036)
Hemant Verma (15115056)
Himanshu Kumar(15115057)
Himanshu Matharu(15115058)
Kalp Garg(15115065)

B.Tech 3rd Year Electrical Engineering Department

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

INTRODUCTION

We have developed a frequency independent sinusoidal wave phase shifter using FPGA Spartan-3E. Here, we are using the on-board ADC (Analog to Digital Converter) LTC1407A-1 to convert the analog input from function generator to a 14 bit digital signal. This output data is further processed through the CORDIC algorithm and then accordingly the required phase shift is performed, which interestingly does not depend on the input frequency. Then, the phase shifted digital signal is sent to the on-board DAC (Digital to Analog Converter) LTC2624 and the phase shifted analog output is obtained at the DAC A Header. The phase shifted output signal is seen on a digitaloscilloscope.

ADC

The Spartan-3E FPGA Starter Kit board includes a two-channel analog capture circuit,consisting of a programmable scaling *pre-amplifier* and an *analog-to-digital converter*(ADC). Analog inputs are supplied on the J7 header.The analog capture circuit consists of a Linear Technology LTC6912-1 programmable preamplifierthat scales the incoming analog signal on header J7. The outputof pre-amplifier connects to a Linear Technology LTC1407A-1 ADC.

DIGITAL OUTPUTS BY ADC

The analog capture circuit converts the analog voltage on VINA or VINB and converts it toa 14-bit digital representation,

$$D[13:0] = \{GAIN * \left(\frac{VIN - 1.65\,V}{1.25V}\right)\} * 8192$$

The GAIN is the current setting loaded into the programmable pre-amplifier. The reference voltage for the amplifier and the ADC is 1.65V, the ADC presents a 14-bit, and two’s complement digital output.

Since, we are using a gain of -1 for both VINA and VINB channels; therefore the input voltage range is 0.4 V to 2.9 V.

SPI CONTROL INTERFACE

The gainfor each amplifier is sent as an 8-bit command word, consisting of two 4-bit fieldsThe SPI bus transaction starts when the FPGA asserts AMP_CS Low. Theamplifier captures serial data on SPI_MOSI on the rising edge of the SPI_SCK clock signal. The amplifier presents serial data on AMP_DOUT on the falling edge of SPI_SCK.The AD_CONV signal is not a traditionalSPI slave select enable. Be sure to provide enough SPI_SCK clock cycles so that the ADCleaves the SPI_MISO signal in the high-impedance state. Otherwise, the ADC blocks communication to the other SPI peripherals. The ADC 3-states its data output for two clock cycles beforeand after each 14-bit data transfer

DAC

The Spartan-3E FPGA Starter Kit board includes an SPI-compatible, four-channel, serialDigital-to-Analog Converter (DAC). The DAC device is a Linear Technology LTC2624quad DAC with 12-bit unsigned resolution. The four outputs from the DAC appear on theJ5 header.

SPI COMMUNICATION

The FPGA uses a Serial Peripheral Interface (SPI) to communicatedigital values to each of the four DAC channels. The SPI bus is a full-duplex, synchronous, character-oriented channel employing a simple four-wire interface. A bus master—the FPGA in this example—drives the bus clock signal (SPI_SCK) and transmits serial data(SPI_MOSI) to the selected bus slave— the DAC. At the same time, the busslave provides serial data (SPI_MISO) back to the bus master.

Each bit is transmitted orreceived relative to the SPI_SCK clock signal. The bus is fully static and supports clocksrate up to the maximum of 50 MHz, after driving the DAC_CS slave select signal Low, the FPGA transmits data on theSPI_MOSI signal, MSB first. The LTC2624 captures input data (SPI_MOSI) on the risingedge of SPI_SCK; the data must be valid for at least 4 ns relative to the rising clock edge.

The LTC2624 DAC transmits its data on the SPI_MISO signal on the falling edge of SPI_SCK. The FPGA captures this data on the next rising SPI_SCK edge. The FPGA mustread the first SPI_MISO value on the first rising SPI_SCK edge after DAC_CS goes Low, otherwise bit 31 is missed.

After transmitting all 32 data bits, the FPGA completes the SPI bus transaction byreturning the DAC_CS slave select signal High. The High-going edge starts the actualdigital-to-analog conversion process within the DAC.

COMMUNICATION PROTOCOL

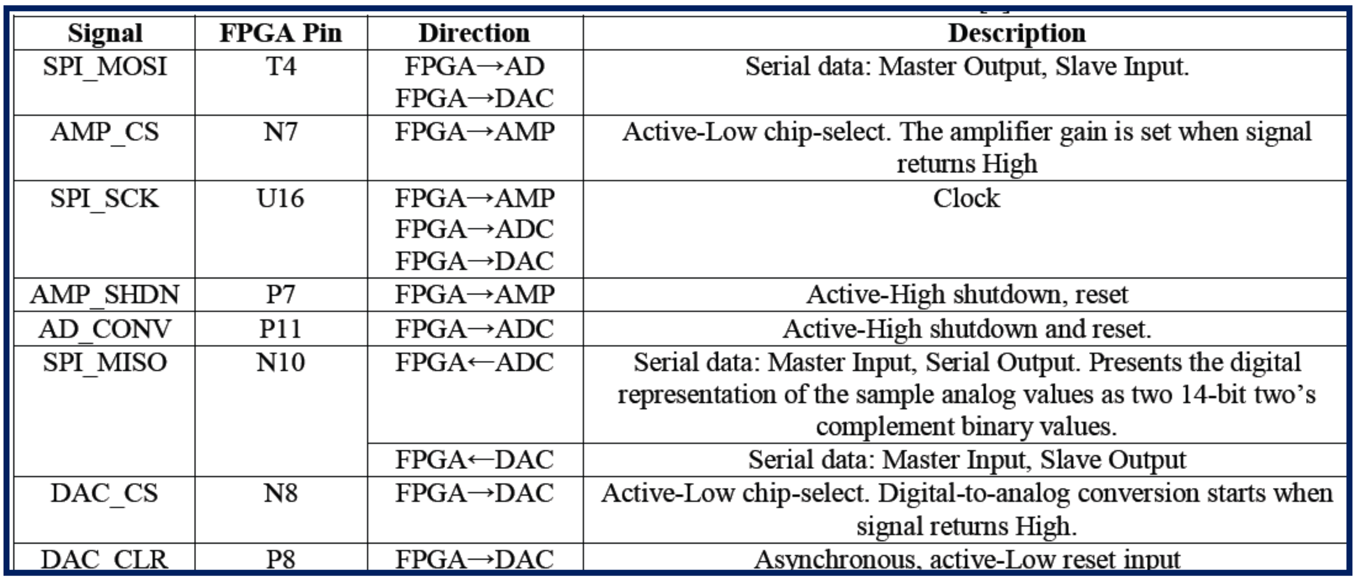
Inside the D/A converter, the SPI interface is formed by a 32-bit shift register. Each 32-bitcommand word consists of a command, an address, followed by data value.The FPGA first sends eight dummy or “don’t care” bits, followed by a 4-bit command. Themost commonly used command with the board is COMMAND [3:0] = “0011”, whichimmediately updates the selected DAC output with the specified data value. Following thecommand, the FPGA selects one or all the DAC output channels via a 4-bit address field.Following the address field, the FPGA sends a 12-bit unsigned data value that the DACconverts to an analog value on the selected output. Finally, four additional dummy or*don’t care* bits pad the 32-bit command word.Each DAC output level is the analog equivalent of a 12-bitunsigned digital value, D [11:0], written by the FPGA to the DAC via the SPI interface.

The voltage on a specific output is

$$VOUT \ = \left(\frac{D[11:0]}{4096}\right) * V\_REFERENCE$$

Channels A and B use a3.3V reference voltage and Channels C and D use a 2.5V reference. Since, we are using Channel A for output signal

$$VOUT \ = \left(\frac{D[11:0]}{4096}\right) * (3.3\,V \ \pm 5\%)$$



Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA→AD FPGA→DAC	Serial data: Master Output, Slave Input.
AMP_CS	N7	FPGA→AMP	Active-Low chip-select. The amplifier gain is set when signal returns High
SPI_SCK	U16	FPGA→AMP FPGA→ADC FPGA→DAC	Clock
AMP_SHDN	P7	FPGA→AMP	Active-High shutdown, reset
AD_CONV	P11	FPGA→ADC	Active-High shutdown and reset.
SPI_MISO	N10	FPGA←ADC	Serial data: Master Input, Serial Output. Presents the digital representation of the sample analog values as two 14-bit two’s complement binary values.
		FPGA←DAC	Serial data: Master Input, Slave Output
DAC_CS	N8	FPGA→DAC	Active-Low chip-select. Digital-to-analog conversion starts when signal returns High.
DAC_CLR	P8	FPGA→DAC	Asynchronous, active-Low reset input

Figure 1 : AMP, ADC and DAC Interface Signals

CORDICALGORITHM

Coordinate Rotation Digital Computer (CORDIC) is an efficient and versatile algorithm that can implement a phase shift without using multipliers. The phase shifting is done by multiplying the complex quantity:

$$e^{j\theta} = \cos\theta + j\sin\theta$$

To the incoming modified wave, by modifying we mean to assume the incoming wave as the sine component and calculate the corresponding cosine component by the knowledge of input wave magnitude.

The equations that shift the phase of complex number$I_0 + jQ_0$ by an angle θ to produce $I_1 + jQ_1$ are as follows:

$$\begin{aligned} I_1 &= I_0\,(\cos(\theta)) - Q_0(\sin(\theta)) \\ Q_1 &= I_0\,(\sin(\theta)) + Q_0(\cos(\theta)) \end{aligned}$$

These equations can be manipulated to provide

$$\begin{aligned} I_1 &= \cos(\theta)\,[I_0 - Q_0(\tan(\theta)) \\ Q_1 &= \cos(\theta)\,[Q_0 + I_0(\tan(\theta)) \end{aligned}$$

The CORDIC algorithm takes advantage of this relationship to approximate an arbitrary phase shift by implementing multiple stages of phase shifts, where the tangent of the phase shift in each successive stage is the next smaller fractional power of 2, and multiplication by this number can be implemented by shifting the input data bits an integer number of places. The first few stages are as follows:

$$I_1 = \cos(\theta_0)\,[I_0 - Q_0(\tan(\theta_0))]$$
$$= \cos(\theta_0)\,[I_0 - Q_0(1)]$$

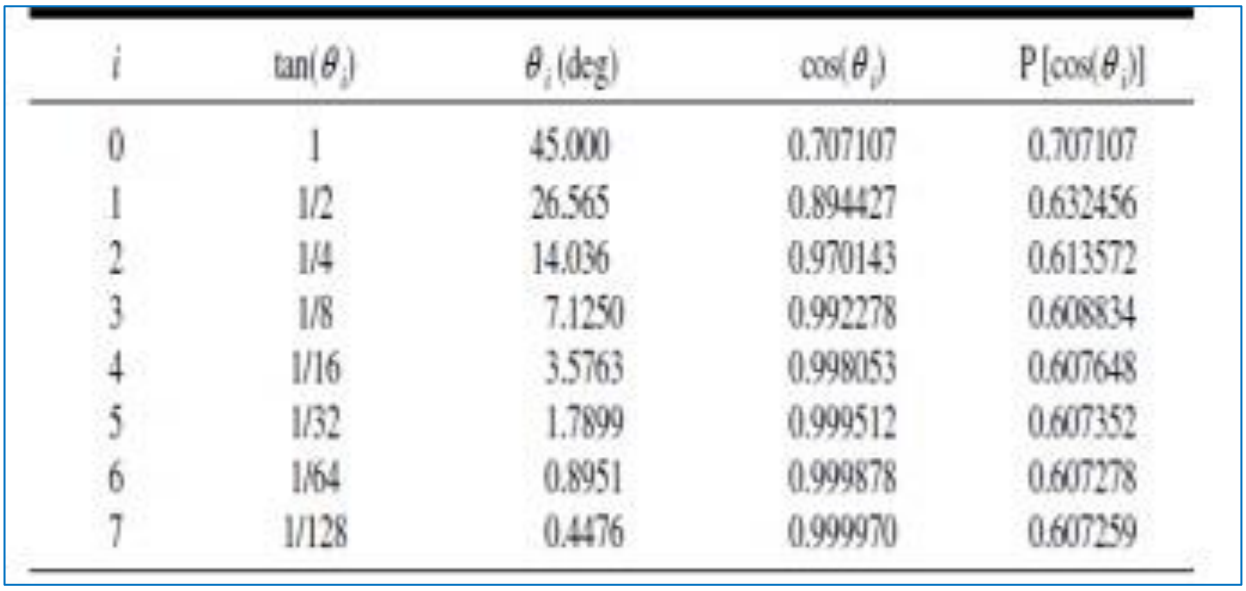
$$Q_1 = \cos(\theta_0)\,[Q_0 + I_0(\tan(\theta_0))]$$
$$= \cos(\theta_0)\,[Q_0 + I_0(1)]$$

$$I_2 = \cos(\theta_1)\,[I_1 - Q_1(\tan(\theta_1))]$$
$$= \cos(\theta_1)\,[I_1 - Q_1\left(\tfrac{1}{2}\right)]$$

$$Q_2 = \cos(\theta_1)\,[Q_1 + I_1(\tan(\theta_1))]$$
$$= \cos(\theta_1)\left[Q_1 + I_1\left(\tfrac{1}{2}\right)\right]$$

Table.1 shows these parameters for an eight-stageCORDIC processor. Each row of the table represents successive iterations of the algorithm. The $tan\ (\theta i)$ column shows the factor by which I and Q values are multiplied for each iteration. Note that these values are fractional powers of 2, so the multiplication can be realized by shifting the binary I and Q values right by i places. The Θ_i column shows the arc tangent of this factor, which can also be thought of as the phase shift applied during each iteration.

Table 1 : CORDIC Parameters for first Eight Stages



i	$\tan(\theta_i)$	θ_i (deg)	$\cos(\theta_i)$	P [$\cos(\theta_i)$]
0	1	45.000	0.707107	0.707107
1	1/2	26.565	0.894427	0.632456
2	1/4	14.036	0.970143	0.613572
3	1/8	7.1250	0.992278	0.608834
4	1/16	3.5763	0.998053	0.607648
5	1/32	1.7899	0.999512	0.607352
6	1/64	0.8951	0.999878	0.607278
7	1/128	0.4476	0.999970	0.607259

The $cos\ (\theta i)$ column shows the cosine of this angle, whichshould be multiplied by the result of each iteration, as shown in the equations above. In actual applications, however, this cosine multiplication is not performed at every iteration. Ateach stage, the implied factor that needs to be multiplied bythe I&Q outputs of the stage in order to provide the correctanswer is the product of all of the cosines up to that point, asshown in the $P[cos(\Theta i)]$ column.

For a large number of iterations, this product of cosines converges to a value of *0.607253*. In most cases, this scaling can be compensated for in later stages of processing. The inverse of this factor,1.64676 for a large number of iterations, is the processing gain imposed on the I&Q results of the CORDIC. If integer arithmetic is performed, an extra bit should be provided at the most significant end of the adders in order to accommodate this increased signal level.

Figure.2 is a flow chart that represents the CORDIC algorithm to implement a phase shift. The inputs to thealgorithm are the $I_{in}, Q_{in}, and\ \phi_{in}$(the desired phase shift).The variable i will keep track of the processing stage being performed and is initialized to zero. The basic algorithm canperform a phase shift between ±90°. If the desired phase shiftis outside of that range, the input I and Q values are first negated, imposing a 180° phase shift, and 180° is subtracted from the desired phase shift. The new phase shift is now within ±90°, and the algorithm proceeds normally.

Next, the algorithm loops through N iterations with the goal of driving the residual phase error,ϕ to zero. In each iteration, a new ϕ is calculated by subtracting or adding the phase shift for that stage (Θ_i from the table) to the previous value ofϕ. If $\phi < 0$, $\Theta_i$is added to ϕ. Otherwise, Θi is subtracted from ϕ. In each stage, the Q (or I) input is divided by a factor of 2^i by shifting the number to the right by i bits, and then added to or subtracted from the I (or Q) input,depending on the sign of ϕ. The variable i is incremented and the process repeats until,$i > N$ at which point the phase-shifted results are available.

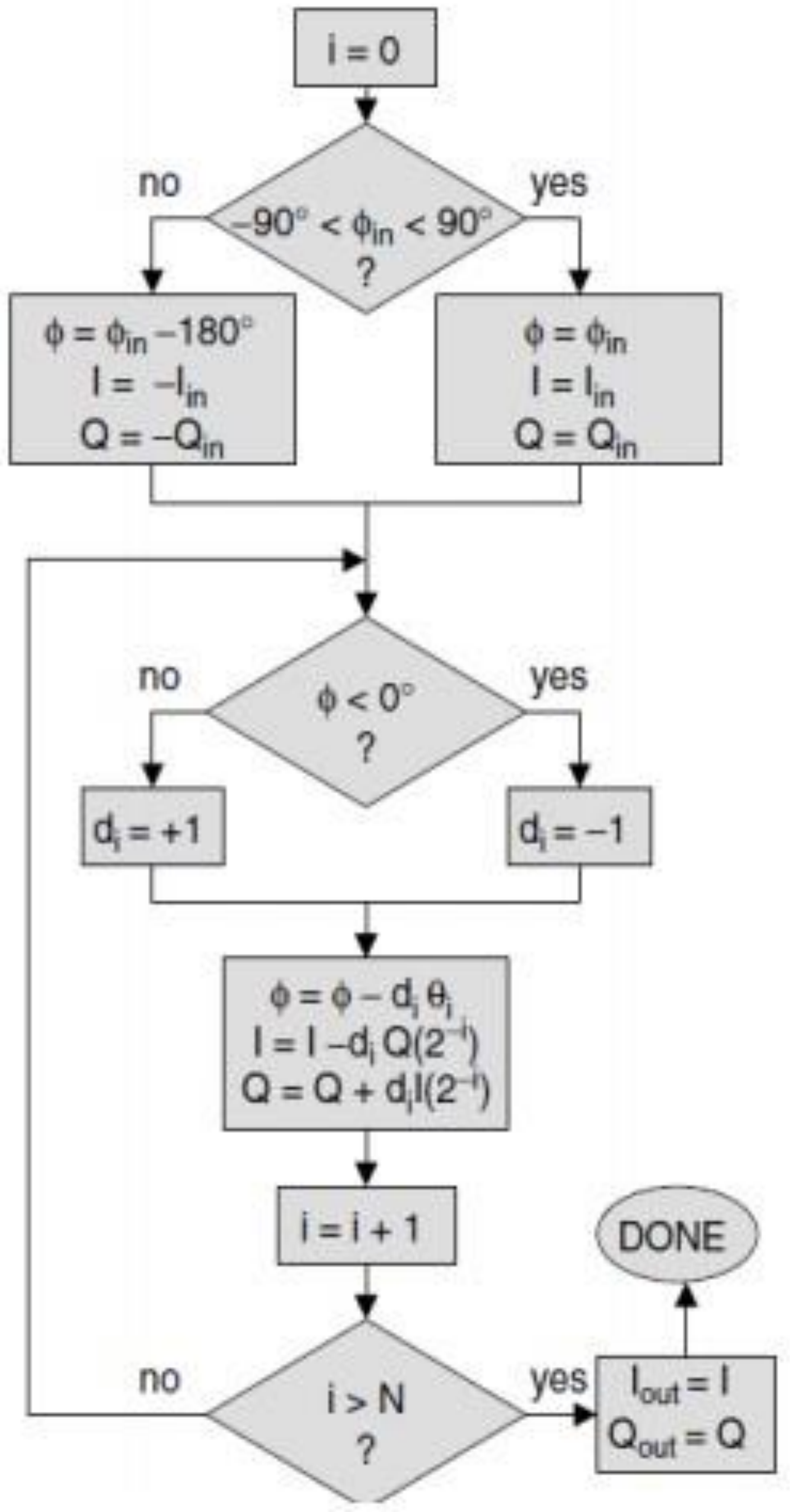


Figure 2 : CORDIC Algorithm Flow Chart

WORKING

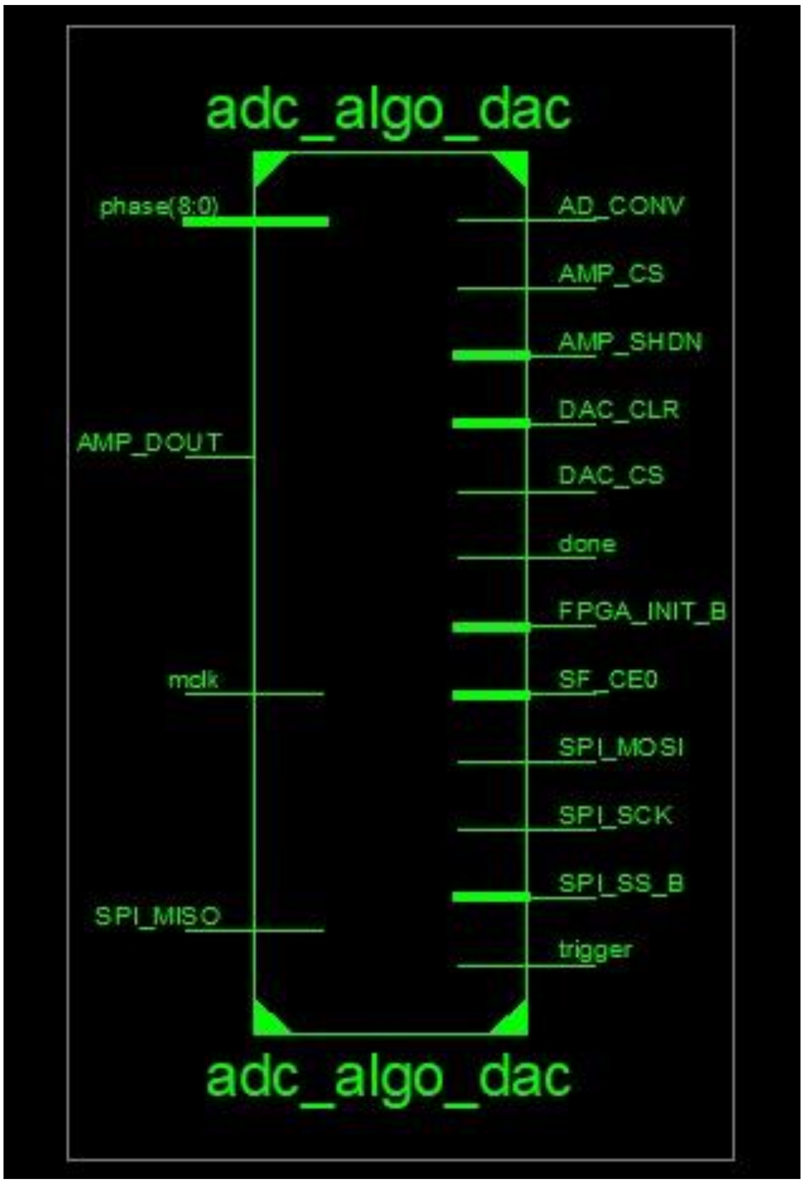


Figure 3 : RTL Schematic

Input Signals:

1) Phase[8:0]	phase shift to be performed
2) AMP_DOUT	echo signal of previous amplifier gain settings
3) mclk	clock signal
4) SPI_MISO	Serial data: Master Input, Slave Output

Output Signals:

1)	AD_CONV	Active High shutdown and reset for ADC
2)	AMP_CS	Active low chip select for pre- amplifier. The amplifier gain is set when signal returns high.
3)	AMO_SHDN	Active High shutdown, reset for pre- amplifier
4)	DAC_CLR	Asynchronous, active-LOW reset input for DAC
5)	DAC_CS	Active-Low chip select for DAC. Digit-to-analog conversion starts when signal returns high.
6)	Done	Done signal
7)	FPGA_INT_B	Device to be disabled on SPI bus. Disabled value is 0.
8)	SF_CE0	Device to be disabled on SPI bus. Disabled value is 1.
9)	SPI_MOSI	Serial data: Master Output, Slave Input.
10)	SPI_SCK	Clock signal
11)	SPI_SS_B	Device to be disabled on SPI bus. Disabled value is 1.
12)	trigger	trigger signal used in ADC

We are using a 1.5MHz clock for the whole process which obtained by using a clock divider on 50MHz internal clock available on the FPGA board. In our algorithm, 3 states are required for ADC, one state for the algorithm and 8 states for DAC.

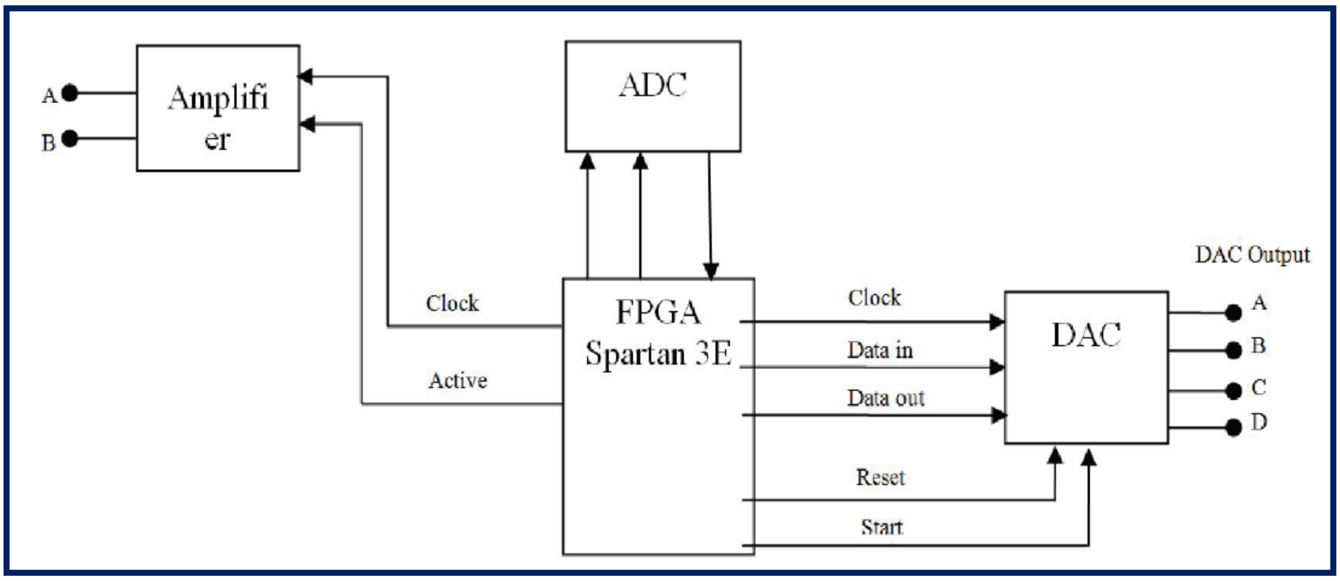


Figure 4 : Block Diagram of pre-amplifier, ADC, DAC and their signals

STATES:

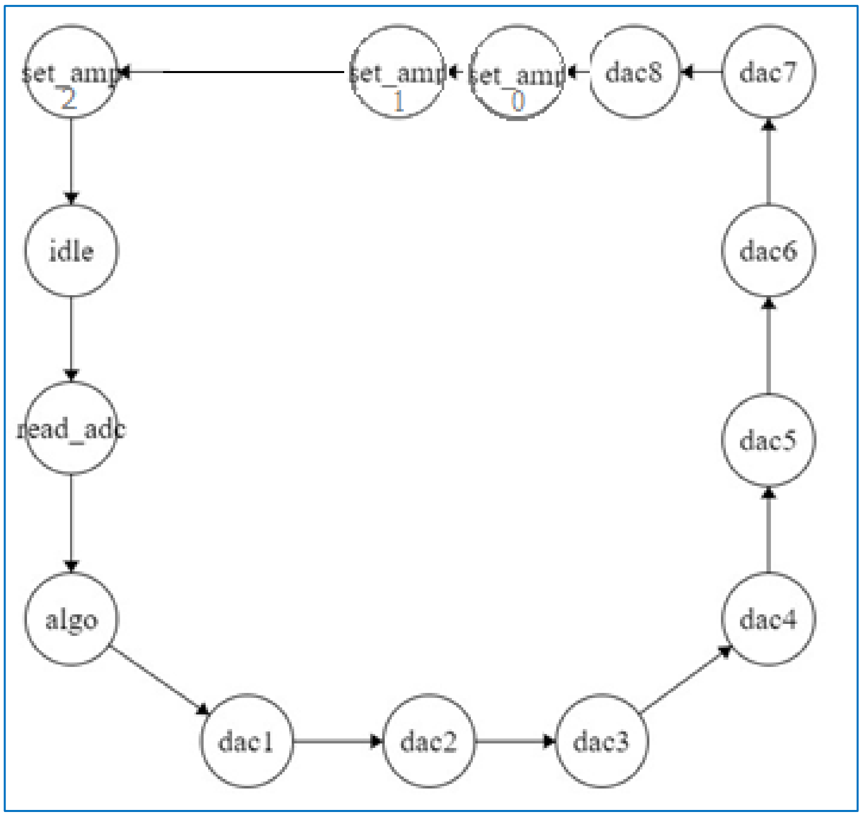


Figure 5 : State Transition Diagram

FOR ADC	set_amp0, set_amp1, set_amp2, idle, read_adc
FOR ALOGRIHTM	algo
FOR DAC	dac1, dac2, dac3, dac4, dac5, dac6, dac7, dac8

set_amp0, set_amp1, set_amp2 -- Gain of the pre-amplifier is set by sending the data word “10001000” on the SPI_MOSI signal since gain1 = gain2 = 0001.

idle -- This state introduces a 13ns delay in the process and resets the ADC.

read_adc -- In this state 32 bit data word consisting of the 13 bit data from channel A and 13 bit data from channel B is compiled. The channel A is being used as input channel here. Thus, the 13 bit data is extracted on the amplitud1_buffer signal. The 12th bit of amplitude1_buffer represents the sign and thus 2’s complement is performed on the data word if it is 1, otherwise no further computation is processed and data is transferred to amplitude1 signal.

algo -- In this state, the CORDIC algorithm is implemented and required phase shifts are performed on the signal received. The phase shifted signal is outputted on the signal ‘data’.

dac1 -- This is the first state of DAC implementation in which DAC is enabled.

dac2 -- In this state command signal “SEND” is formed i.e.
SEND=”0000000000110000” & data &”0000”.

dac3 --In this state conversion of digital signal to analog signal is started.

dac4 --In this state each bit of the command signal is inputted to the DAC.

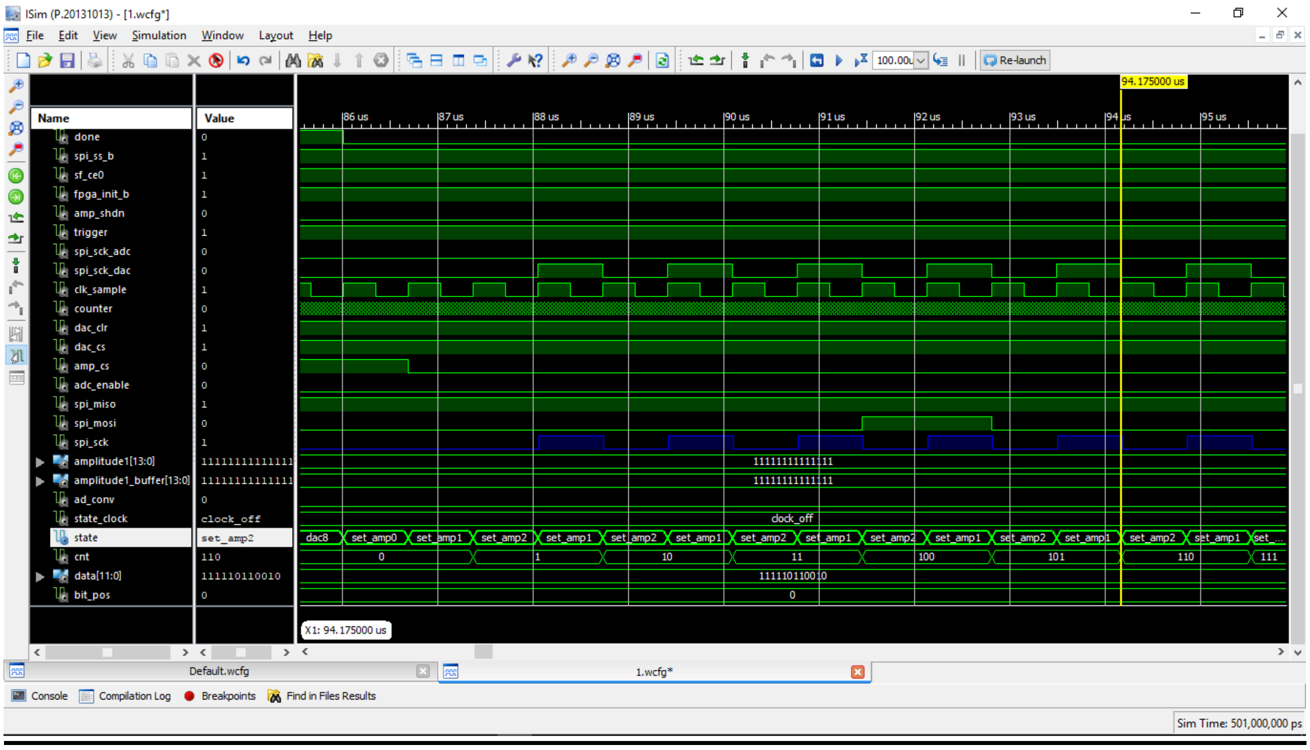
dac5 -- DAC Clock signal is cleared in this state.

dac6 -- In this state the conversion is completed and DAC chip select signal is set high.

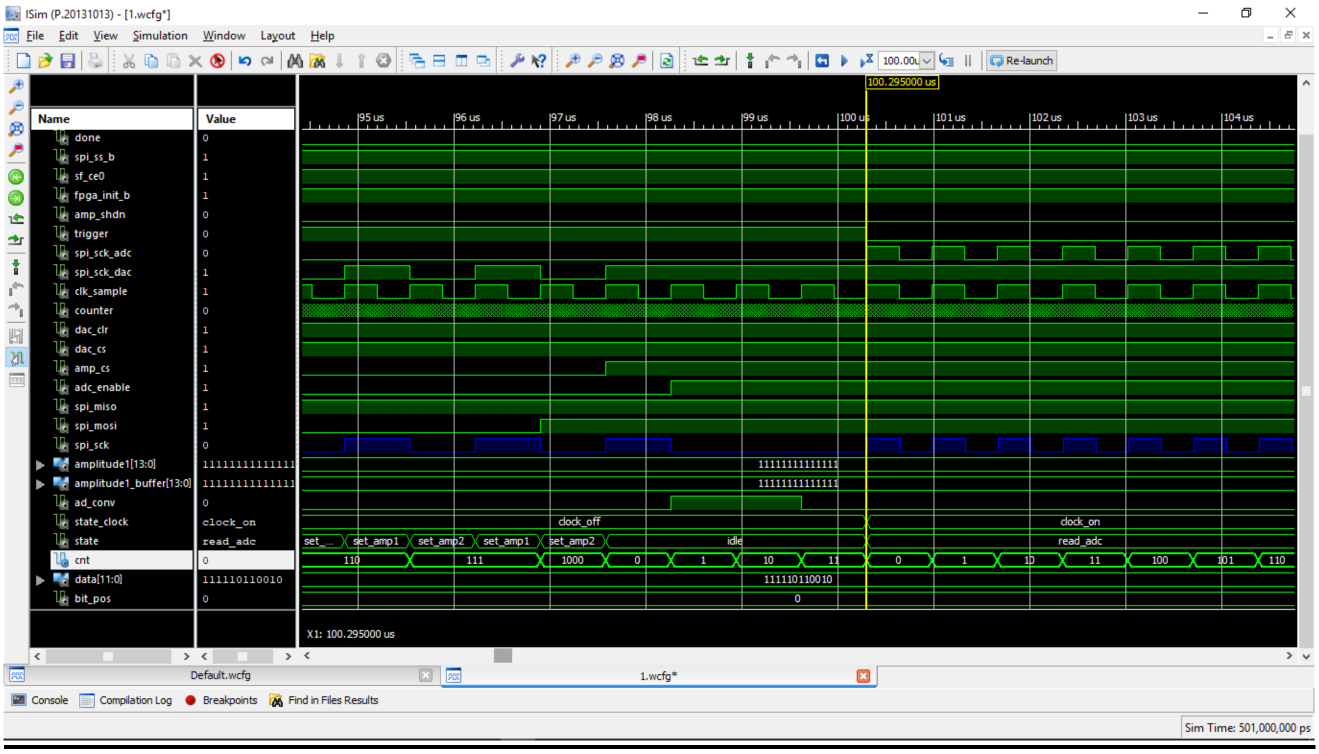
dac7 --Done signal is set in this state.

dac8 -- This is the state in which done signal is cleared and the system goes to SET_AMP state.

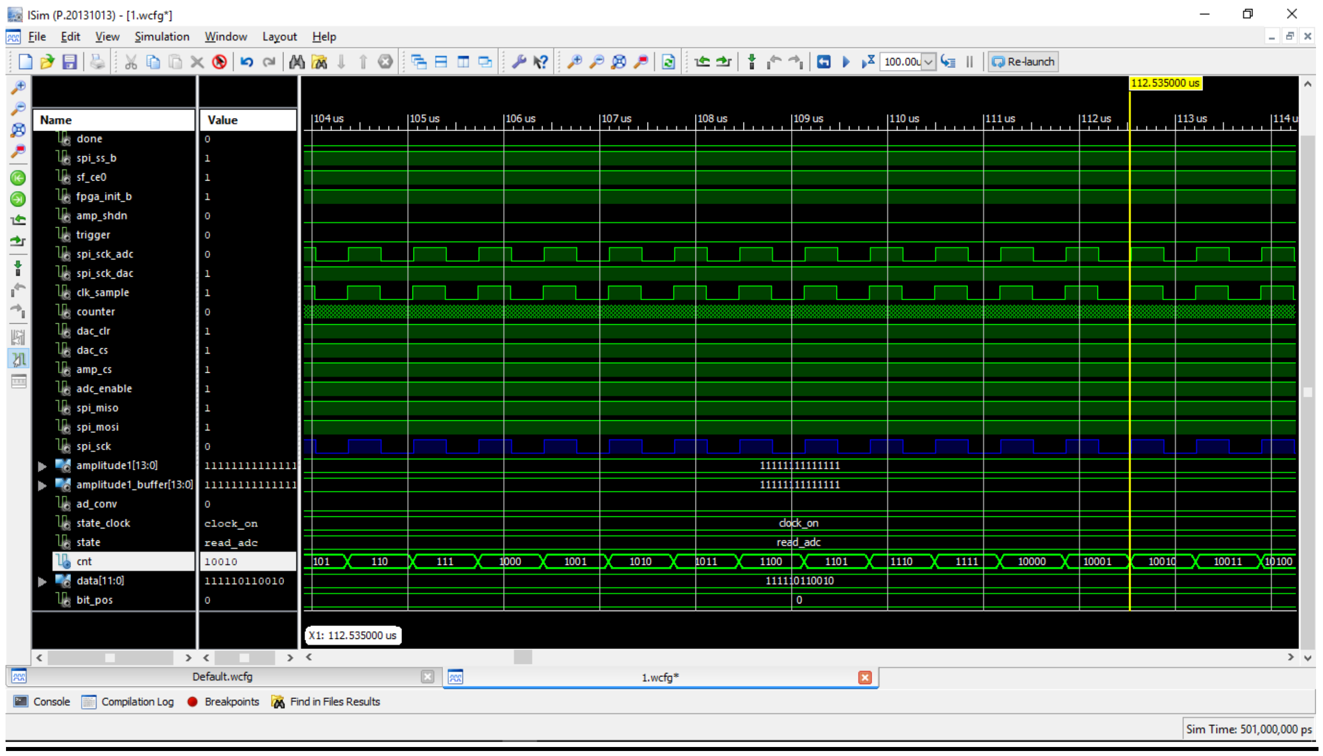
EXPERIMENTALRESULTS



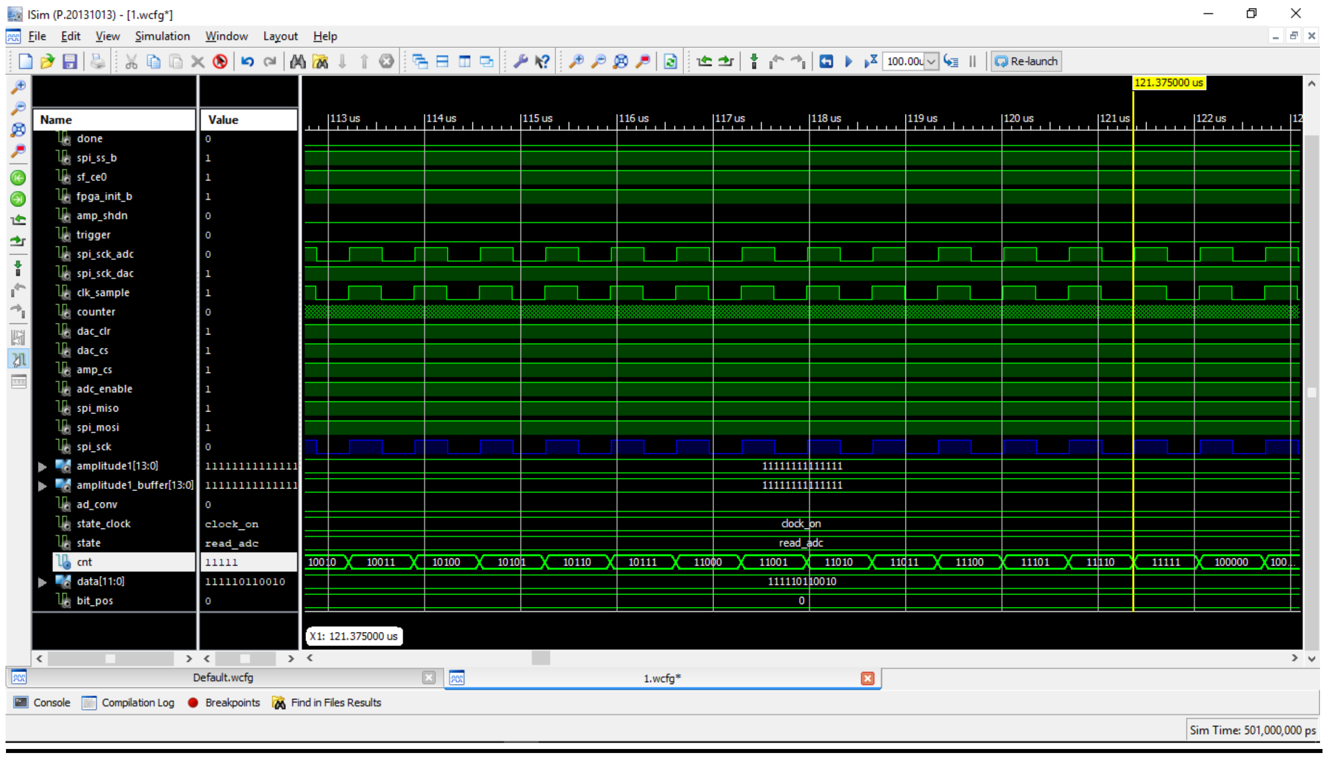
Name	Value
done	0
spi_ss_b	1
sf_ce0	1
fpga_init_b	1
amp_shdn	0
trigger	1
spi_sck_adc	0
spi_sck_dac	0
clk_sample	1
counter	0
dac_clr	1
dac_cs	1
amp_cs	0
adc_enable	0
spi_miso	1
spi_mosi	0
spi_sck	1
amplitude1[13:0]	11111111111111
amplitude1_buffer[13:0]	11111111111111
ad_conv	0
state_clock	clock_off
state	set_amp2
cnt	110
data[11:0]	111110110010
bit_pos	0



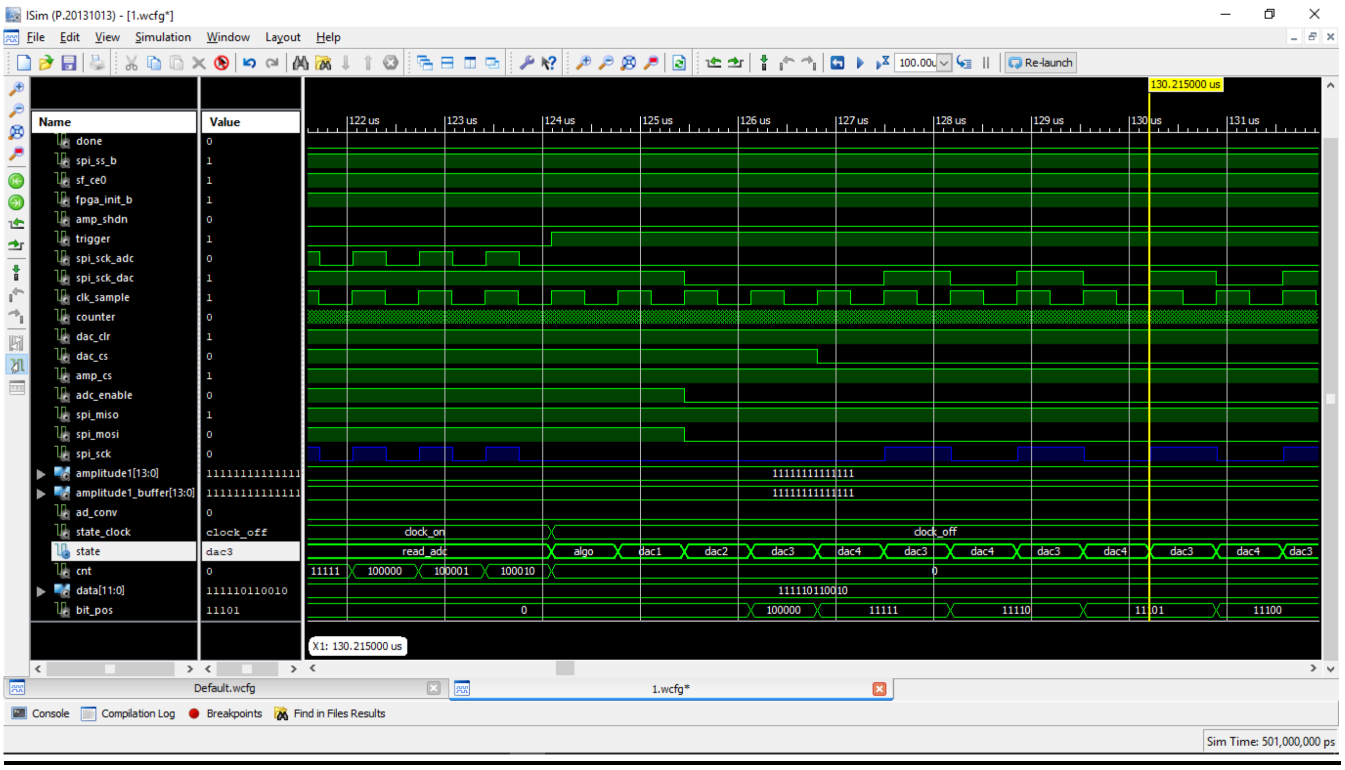
Name	Value
done	0
spi_ss_b	1
sf_ce0	1
fpga_init_b	1
amp_shdn	0
trigger	0
spi_sck_adc	0
spi_sck_dac	1
clk_sample	1
counter	0
dac_clr	1
dac_cs	1
amp_cs	1
adc_enable	1
spi_miso	1
spi_mosi	1
spi_sck	0
amplitude1[13:0]	11111111111111
amplitude1_buffer[13:0]	11111111111111
ad_conv	0
state_clock	clock_on
state	read_adc
cnt	0
data[11:0]	111110110010
bit_pos	0



```
ISim (P.20131013) - [1.wcfg*]

Name                    Value
done                    0
spi_ss_b                1
sf_ce0                  1
fpga_init_b             1
amp_shdn                0
trigger                 0
spi_sck_adc             0
spi_sck_dac             1
clk_sample              1
counter                 0
dac_clr                 1
dac_cs                  1
amp_cs                  1
adc_enable              1
spi_miso                1
spi_mosi                1
spi_sck                 0
amplitude1[13:0]        1111111111111
amplitude1_buffer[13:0] 1111111111111
ad_conv                 0
state_clock             clock_on
state                   read_adc
cnt                     10010
data[11:0]              111110110010
bit_pos                 0

X1: 112.535000 us
Sim Time: 501,000,000 ps
```



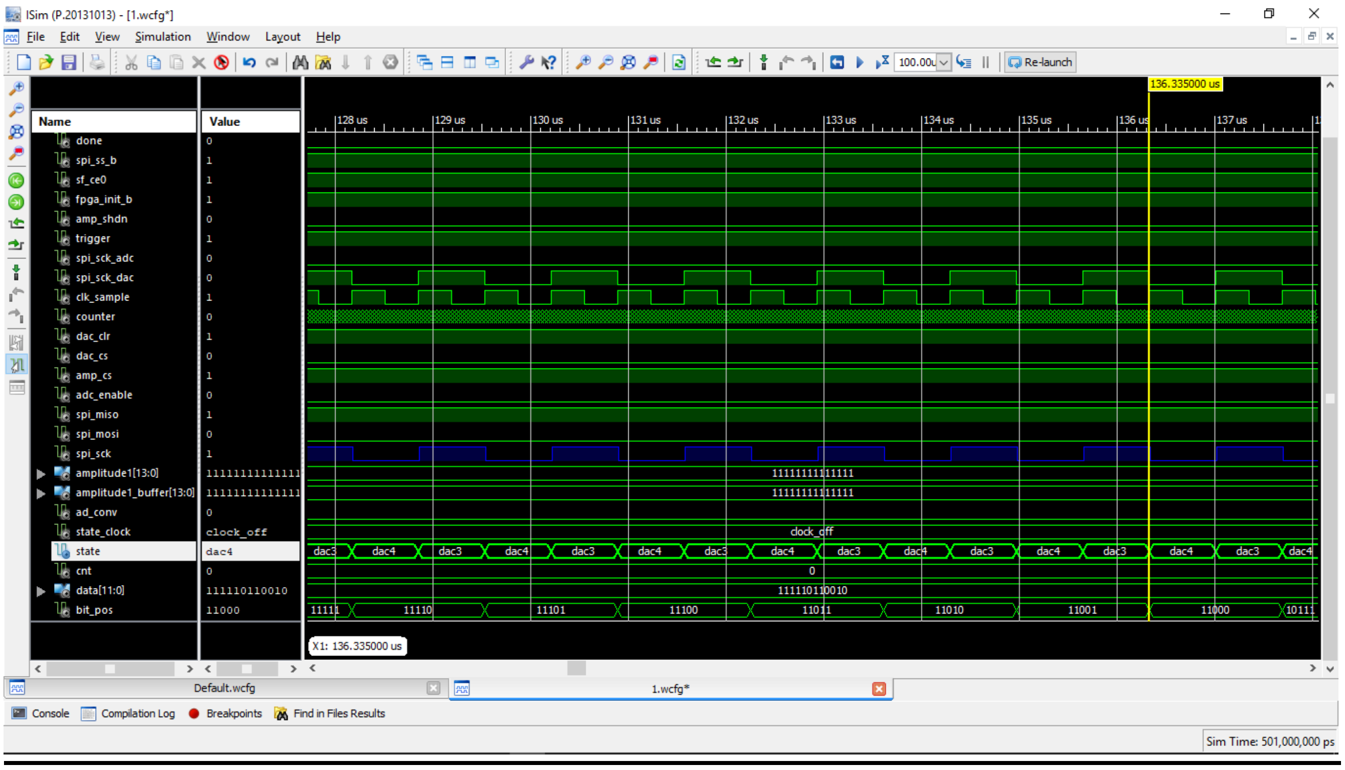
```
ISim (P.20131013) - [1.wcfg*]

Name                    Value
done                    0
spi_ss_b                1
sf_ce0                  1
fpga_init_b             1
amp_shdn                0
trigger                 0
spi_sck_adc             0
spi_sck_dac             1
clk_sample              1
counter                 0
dac_clr                 1
dac_cs                  1
amp_cs                  1
adc_enable              1
spi_miso                1
spi_mosi                1
spi_sck                 0
amplitude1[13:0]        1111111111111
amplitude1_buffer[13:0] 1111111111111
ad_conv                 0
state_clock             clock_on
state                   read_adc
cnt                     11111
data[11:0]              111110110010
bit_pos                 0

X1: 121.375000 us
Sim Time: 501,000,000 ps
```


ISim (P.20131013) - [1.wcfg*]

Name	Value
done	0
spi_ss_b	1
sf_ce0	1
fpga_init_b	1
amp_shdn	0
trigger	1
spi_sck_adc	0
spi_sck_dac	1
clk_sample	1
counter	0
dac_clr	1
dac_cs	0
amp_cs	1
adc_enable	0
spi_miso	1
spi_mosi	0
spi_sck	0
amplitude1[13:0]	11111111111111
amplitude1_buffer[13:0]	11111111111111
ad_conv	0
state_clock	clock_off
state	dac3
cnt	0
data[11:0]	111110110010
bit_pos	11101

X1: 130.215000 us

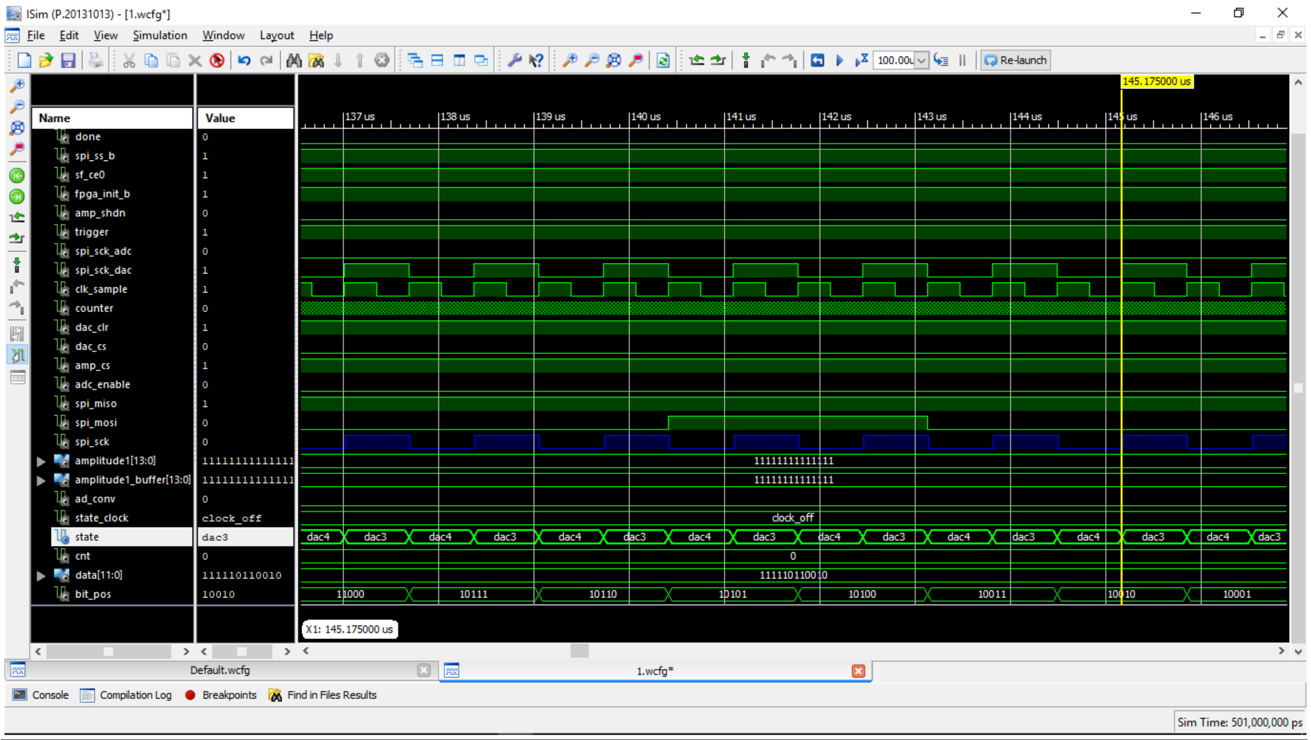
Sim Time: 501,000,000 ps


ISim (P.20131013) - [1.wcfg*]

Name	Value
done	0
spi_ss_b	1
sf_ce0	1
fpga_init_b	1
amp_shdn	0
trigger	1
spi_sck_adc	0
spi_sck_dac	0
clk_sample	1
counter	0
dac_clr	1
dac_cs	0
amp_cs	1
adc_enable	0
spi_miso	1
spi_mosi	0
spi_sck	1
amplitude1[13:0]	11111111111111
amplitude1_buffer[13:0]	11111111111111
ad_conv	0
state_clock	clock_off
state	dac4
cnt	0
data[11:0]	111110110010
bit_pos	11000

X1: 136.335000 us

Sim Time: 501,000,000 ps



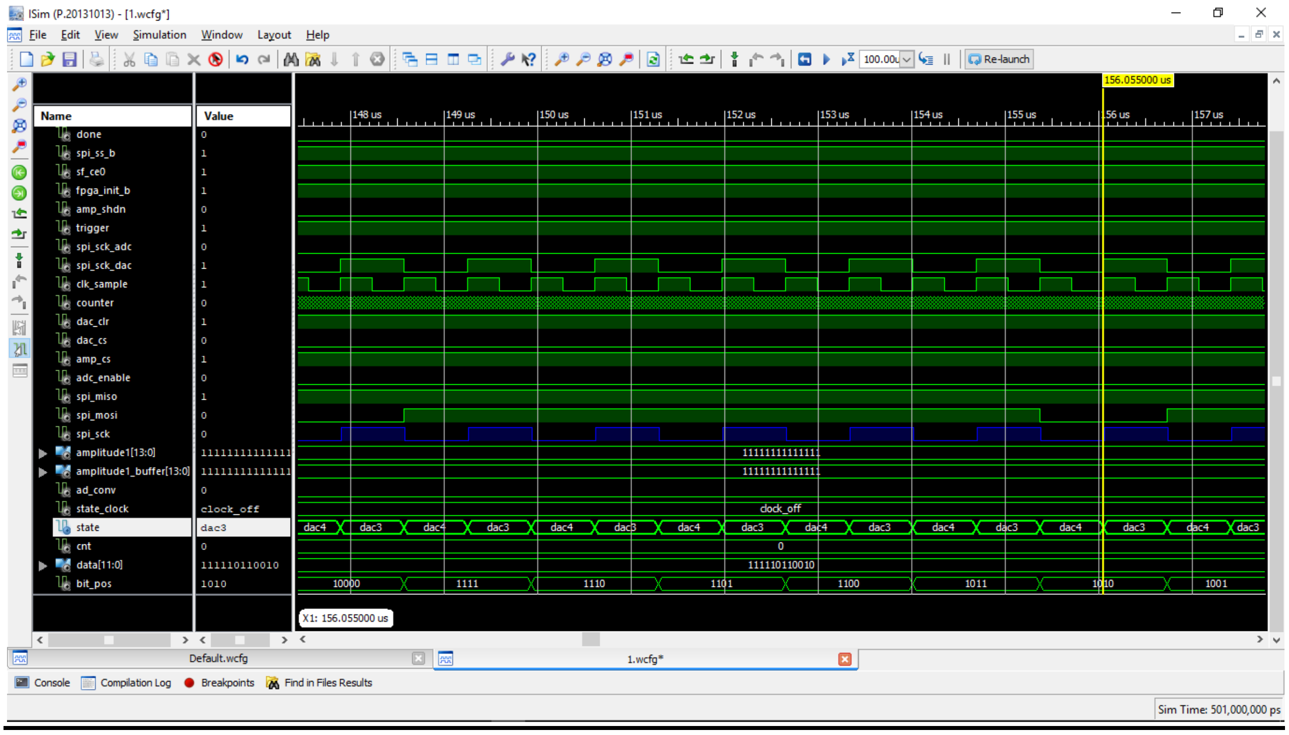
ISim (P.20131013) - [1.wcfg*]

Name	Value
done	0
spi_ss_b	1
sf_ce0	1
fpga_init_b	1
amp_shdn	0
trigger	1
spi_sck_adc	0
spi_sck_dac	1
clk_sample	1
counter	0
dac_clr	1
dac_cs	0
amp_cs	1
adc_enable	0
spi_miso	1
spi_mosi	0
spi_sck	0
amplitude1[13:0]	1111111111111
amplitude1_buffer[13:0]	1111111111111
ad_conv	0
state_clock	clock_off
state	dac3
cnt	0
data[11:0]	111110110010
bit_pos	10010

145.175000 us

X1: 145.175000 us

Sim Time: 501,000,000 ps



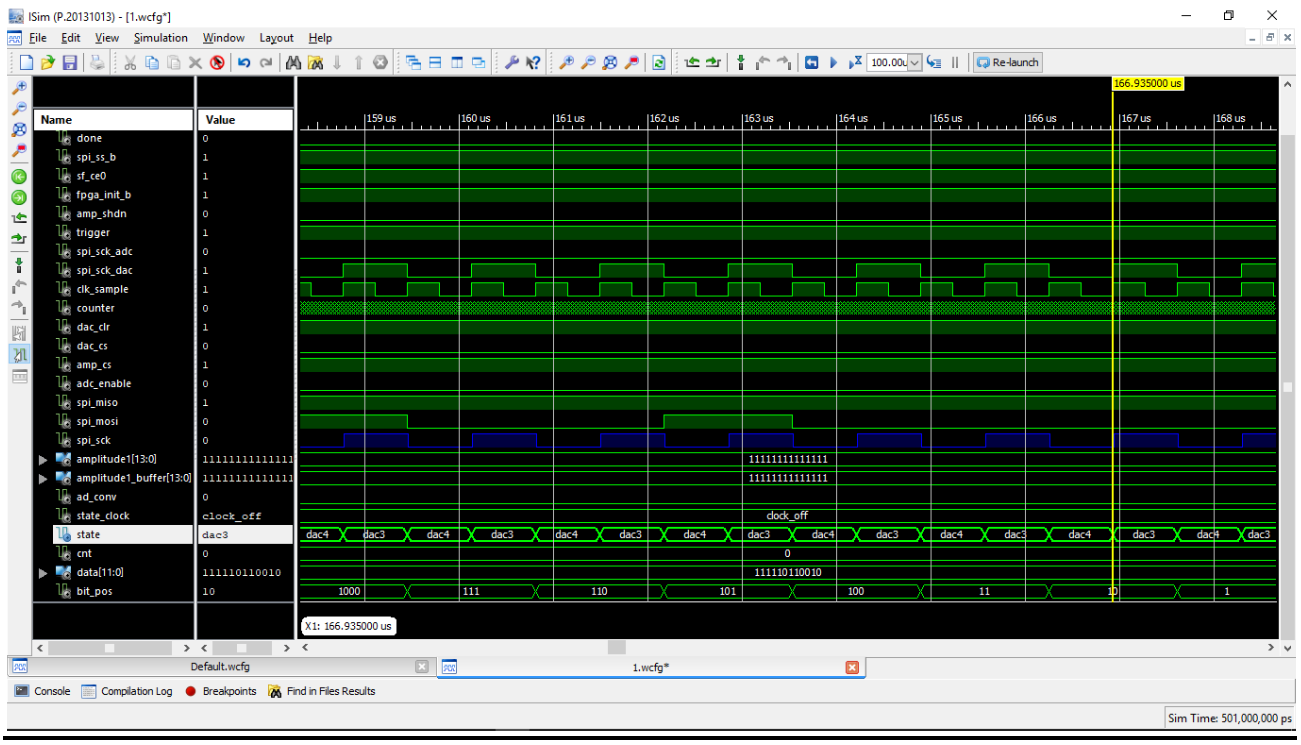
ISim (P.20131013) - [1.wcfg*]

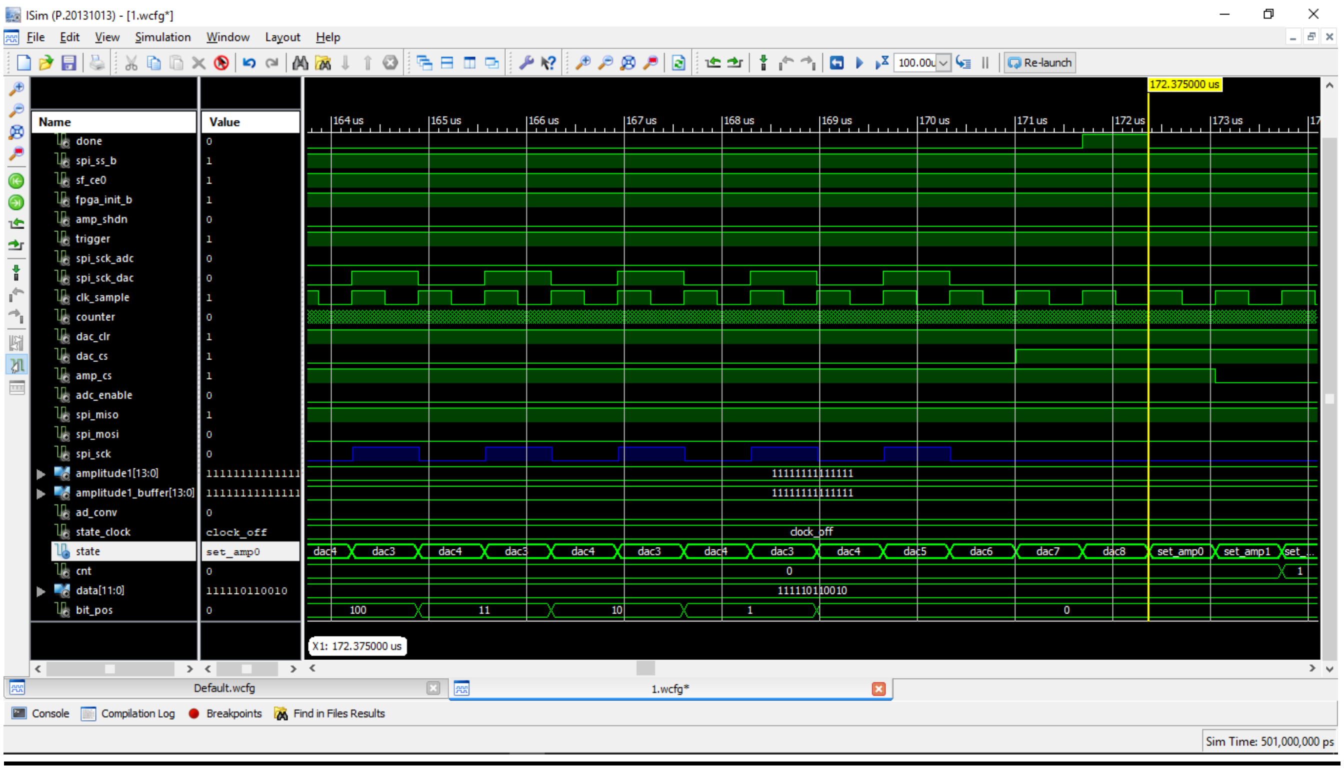
Name	Value
done	0
spi_ss_b	1
sf_ce0	1
fpga_init_b	1
amp_shdn	0
trigger	1
spi_sck_adc	0
spi_sck_dac	1
clk_sample	1
counter	0
dac_clr	1
dac_cs	0
amp_cs	1
adc_enable	0
spi_miso	1
spi_mosi	0
spi_sck	0
amplitude1[13:0]	1111111111111
amplitude1_buffer[13:0]	1111111111111
ad_conv	0
state_clock	clock_off
state	dac3
cnt	0
data[11:0]	111110110010
bit_pos	1010

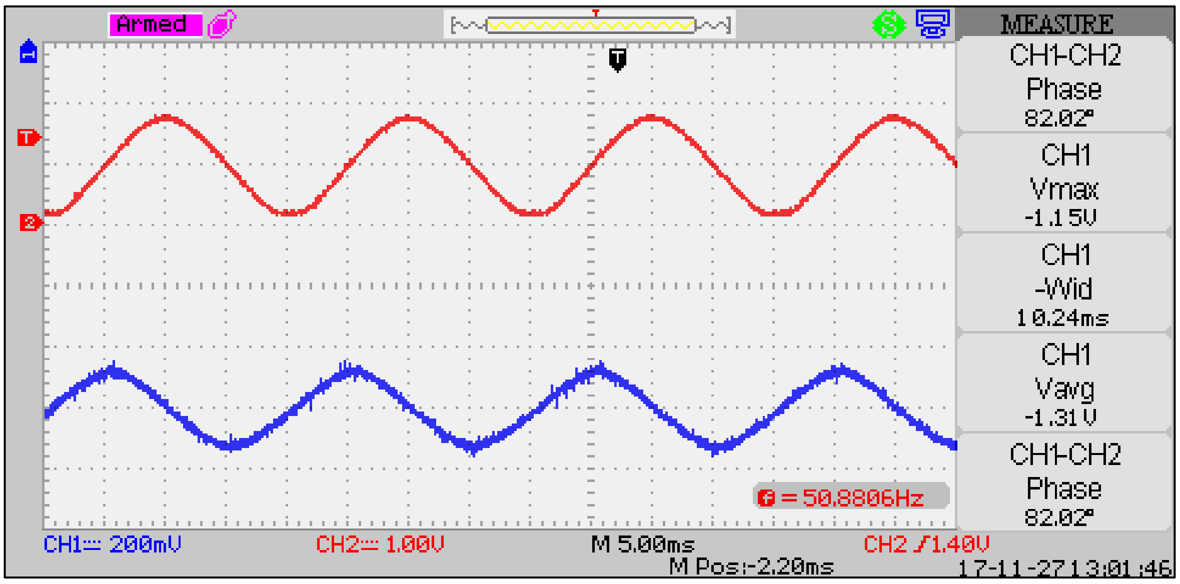
156.055000 us

X1: 156.055000 us

Sim Time: 501,000,000 ps


ISim (P.20131013) - [1.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
166.935000 us
Name
Value
done 0
spi_ss_b 1
sf_ce0 1
fpga_init_b 1
amp_shdn 0
trigger 1
spi_sck_adc 0
spi_sck_dac 1
clk_sample 1
counter 0
dac_clr 1
dac_cs 0
amp_cs 1
adc_enable 0
spi_miso 1
spi_mosi 0
spi_sck 0
amplitude1[13:0] 11111111111111
amplitude1_buffer[13:0] 11111111111111
ad_conv 0
state_clock clock_off
state dac3
cnt 0
data[11:0] 111110110010
bit_pos 10
X1: 166.935000 us
Default.wcfg
1.wcfg*
Console
Compilation Log
Breakpoints
Find in Files Results
Sim Time: 501,000,000 ps

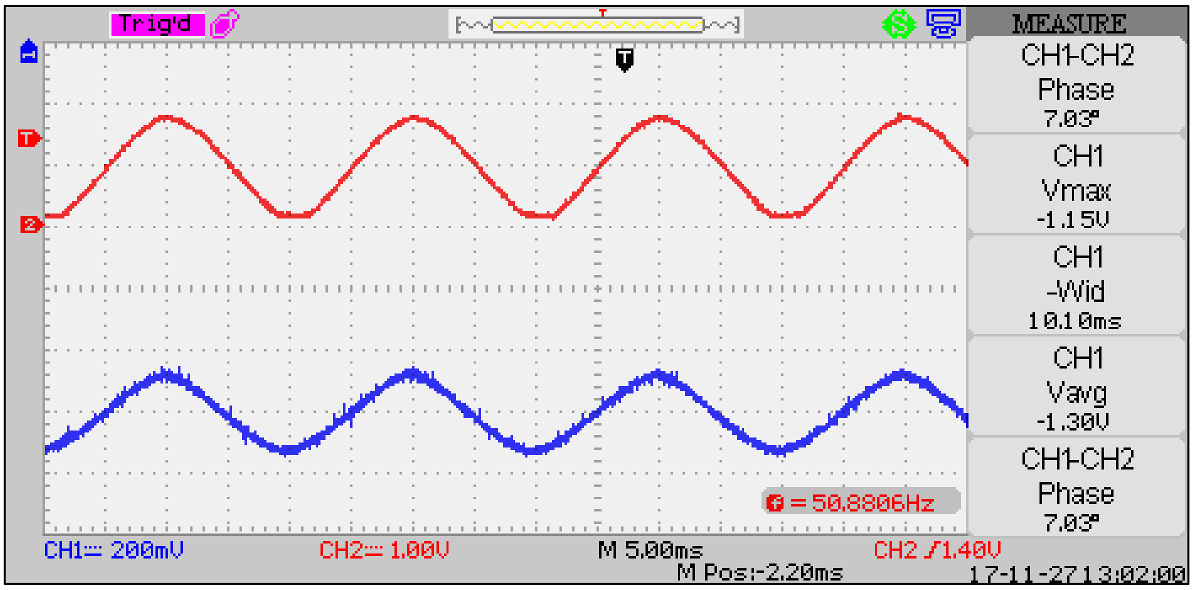

ISim (P.20131013) - [1.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
172.375000 us
Name
Value
done 0
spi_ss_b 1
sf_ce0 1
fpga_init_b 1
amp_shdn 0
trigger 1
spi_sck_adc 0
spi_sck_dac 0
clk_sample 1
counter 0
dac_clr 1
dac_cs 1
amp_cs 1
adc_enable 0
spi_miso 1
spi_mosi 0
spi_sck 0
amplitude1[13:0] 11111111111111
amplitude1_buffer[13:0] 11111111111111
ad_conv 0
state_clock clock_off
state set_amp0
cnt 0
data[11:0] 111110110010
bit_pos 0
X1: 172.375000 us
Default.wcfg
1.wcfg*
Console
Compilation Log
Breakpoints
Find in Files Results
Sim Time: 501,000,000 ps


Armed

MEASURE
CH1-CH2 Phase 82.02°
CH1 Vmax -1.15V
CH1 -Wid 10.24ms
CH1 Vavg -1.31V
CH1-CH2 Phase 82.02°

= 50.8806Hz

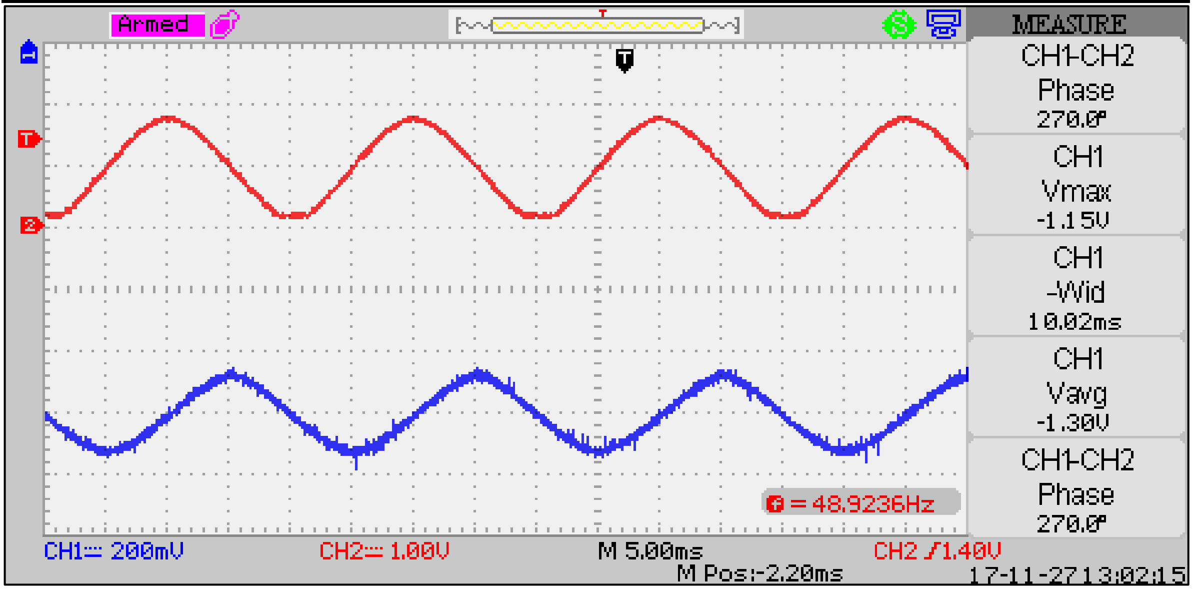
CH1⎓ 200mV CH2⎓ 1.00V M 5.00ms CH2 ⨍1.40V
M Pos:-2.20ms 17-11-27 13:01:46


Trig'd

MEASURE
CH1-CH2 Phase 7.03°
CH1 Vmax -1.15V
CH1 -Wid 10.10ms
CH1 Vavg -1.30V
CH1-CH2 Phase 7.03°

= 50.8806Hz

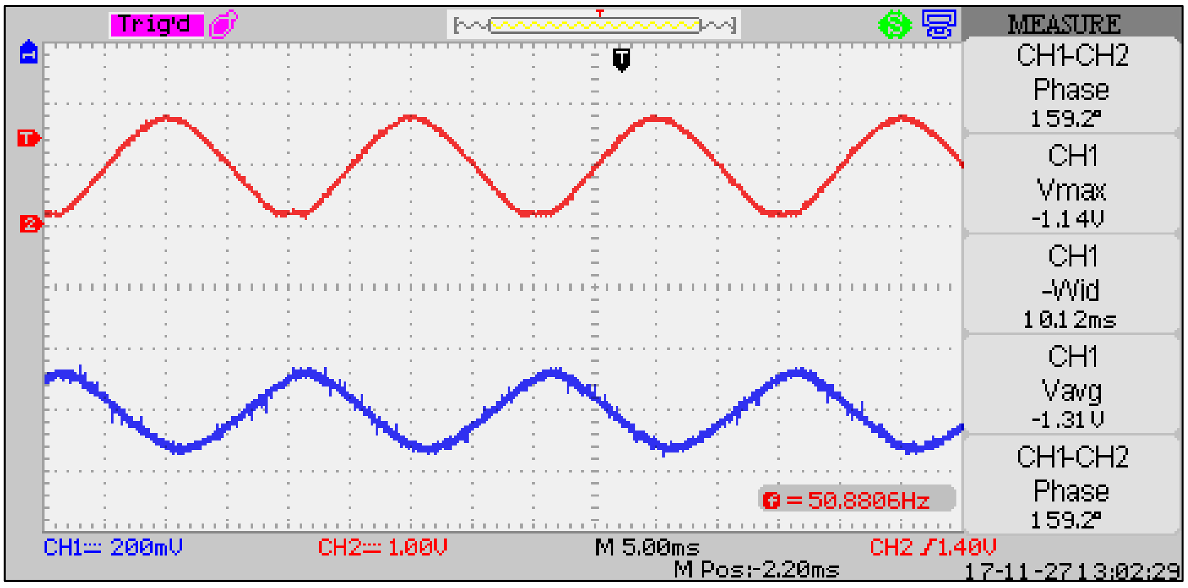
CH1⎓ 200mV CH2⎓ 1.00V M 5.00ms CH2 ⨍1.40V
M Pos:-2.20ms 17-11-27 13:02:00


Armed

MEASURE
CH1-CH2 Phase 270.0°
CH1 Vmax -1.15V
CH1 -Wid 10.02ms
CH1 Vavg -1.30V
CH1-CH2 Phase 270.0°

= 48.9236Hz

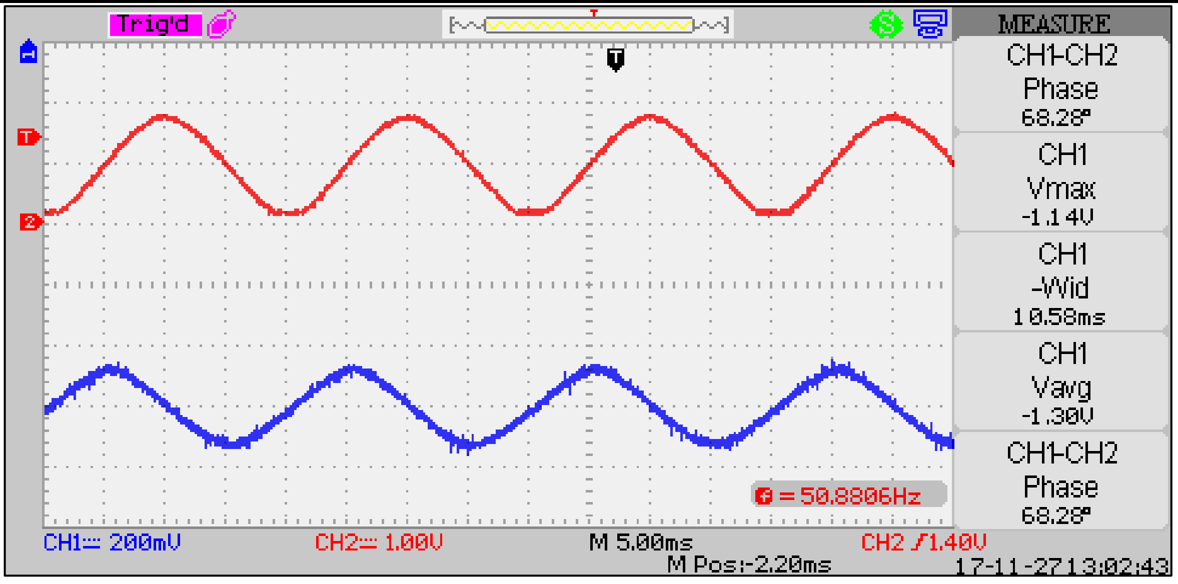
CH1⎓ 200mV CH2⎓ 1.00V M 5.00ms CH2 ⨍1.40V
M Pos:-2.20ms 17-11-27 13:02:15


Trig'd

MEASURE
CH1-CH2 Phase 159.2°
CH1 Vmax -1.14V
CH1 -Wid 10.12ms
CH1 Vavg -1.31V
CH1-CH2 Phase 159.2°

f = 50.8806Hz

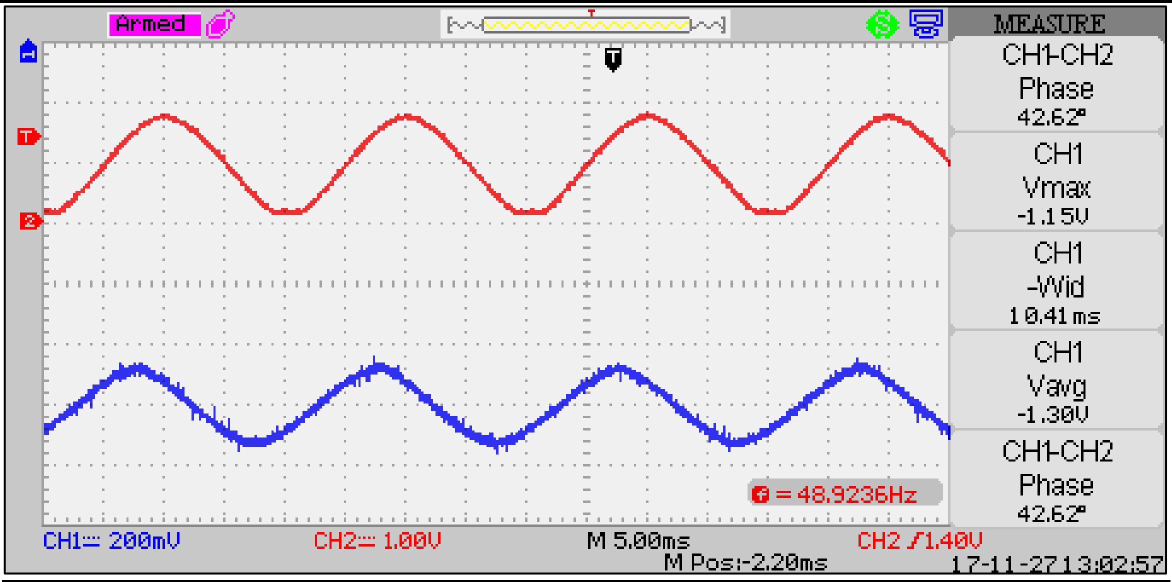
CH1 ⎓ 200mV CH2 ⎓ 1.00V M 5.00ms CH2 ⨍1.40V
M Pos:-2.20ms 17-11-27 13:02:29


Trig'd

MEASURE
CH1-CH2 Phase 68.28°
CH1 Vmax -1.14V
CH1 -Wid 10.58ms
CH1 Vavg -1.30V
CH1-CH2 Phase 68.28°

f = 50.8806Hz

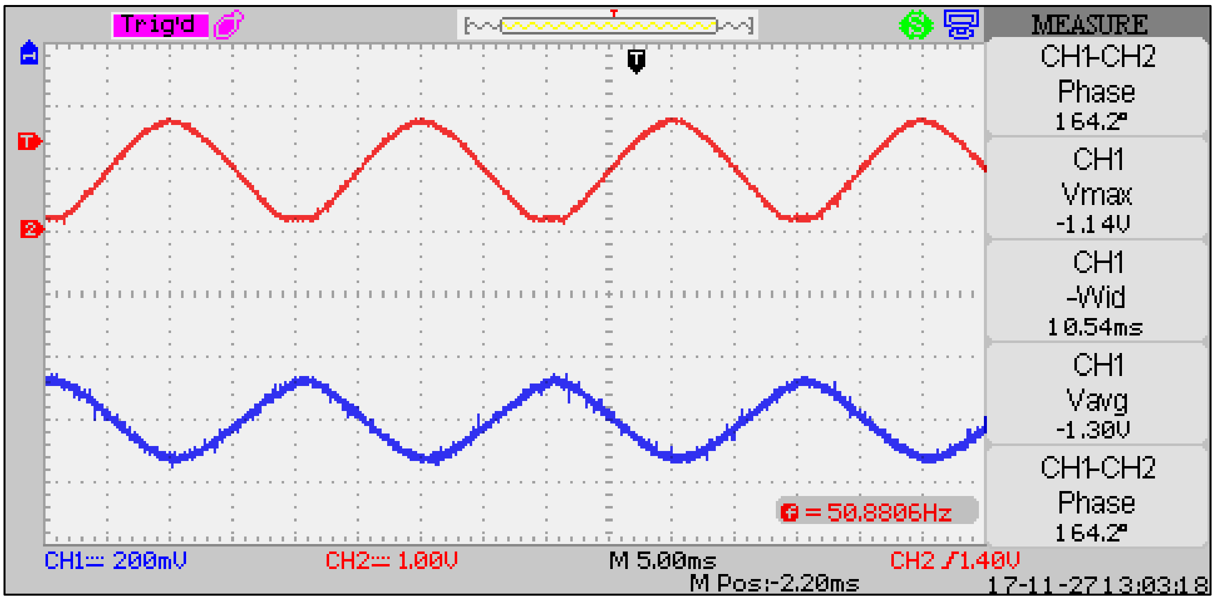
CH1 ⎓ 200mV CH2 ⎓ 1.00V M 5.00ms CH2 ⨍1.40V
M Pos:-2.20ms 17-11-27 13:02:43

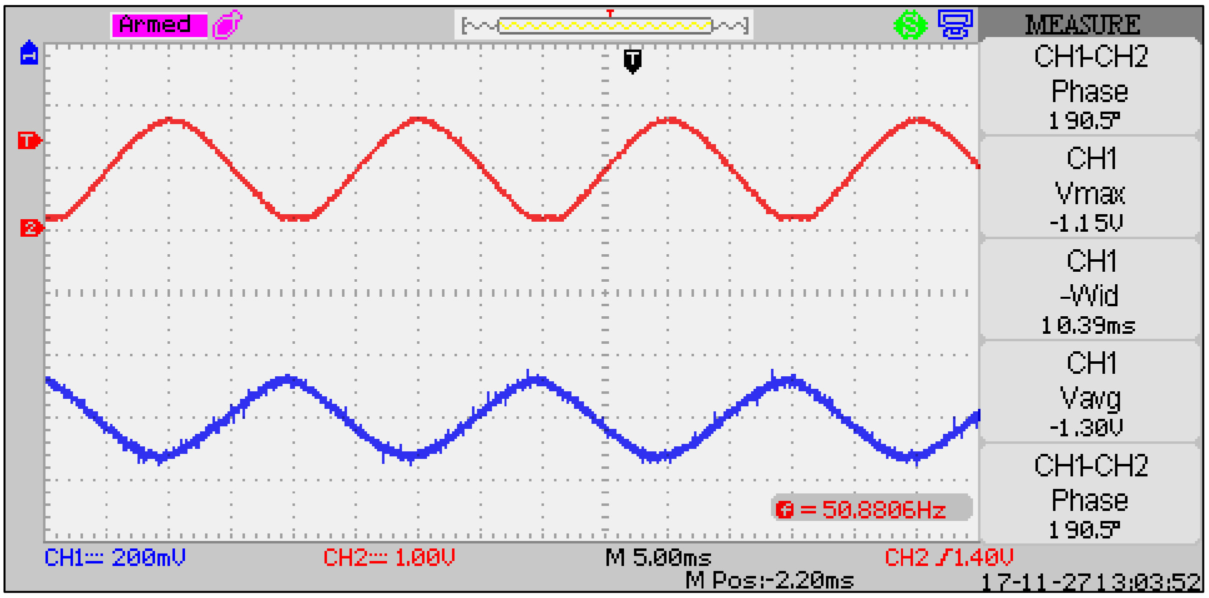

Armed

MEASURE
CH1-CH2 Phase 42.62°
CH1 Vmax -1.15V
CH1 -Wid 10.41ms
CH1 Vavg -1.30V
CH1-CH2 Phase 42.62°

f = 48.9236Hz

CH1 ⎓ 200mV CH2 ⎓ 1.00V M 5.00ms CH2 ⨍1.40V
M Pos:-2.20ms 17-11-27 13:02:57





Desired Phase Shift is given with the help of slide switches present on SPARTAN 3E board. Since, it has only four slide switches, therefore we program input coarsely i.e. on 1 bit change in switch input, phase shifts by about 36°.

CONCLUSION

A frequency independent (variable frequency) sinusoidal wave phase shifter has been designed. On board pre-amplifier (LTC6912-1),ADC(LTC1407A-1) and DAC (LTC2624) of Spartan-3E FPGA board have been utilised. These peripheral devices use a SPI protocol and a common bus.

The desired phase shift is inputted using on-board switches, the vales of which are directly mapped into a binary 9-bit signed word. This input is feed to CORDIC Algorithm along with the processed output of pre-amplifier and ADC set. The algorithm is Frequency independent, which mean that the input wave frequency may vary, however the processing done by the algorithm block remains invariant. After dropping last two bits, the phase shifted wave is feed to the DAC and a corresponding analog wave is seen on the output channel.

The project has been developed on Xilinx ISE Design Suite and the VHDL code was dumped on Spartan-3E starter FPGA board. Despite of various calculations involved, the code has been made synthesisable for FPGA.

REFERENCES

1. Triveni.C, SudhakaraReddy.P , "*Implementation of Phase Shifter using CORDIC on FPGA for RADAR Application*", Volume 5, Issue 6, June 2016, IJARECE.

2. Radi A, Zidan WI, Khedr HI and Mostafa AG. "*Controlling of Analog Capture Circuit and Digital Analog Converter for Spatan-3E FPGA Starter Kit*."*Nat Sci*2013;11(11):177-182.

3. Spartan-3E FPGA Starter Kit Board User Guide, *UG230 (v1.2) January 20, 2011,Xilinx*