LAB REPORT

Digital Systems / VHDL [ET-DTV / ESD1]

Function Generator with DAC

Date of Execution: 29/05/2019 and 05/06/2019

Group: DTVL – 3

Report By:

Harsha Priya Yapamakula Srinivasa Murthy (37091)

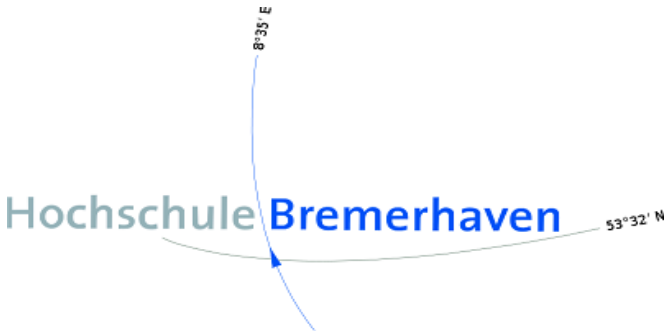
Rama Krishna Reddy Salla(37089)

Under the Guidance of:

Prof. Dr.-Ing. Kai Mueller

Date of Report Submission

13/06/2019



Master of Science in Embedded Systems Design

Hochschule Bremerhaven

June-2019

TABLE OF CONTENTS

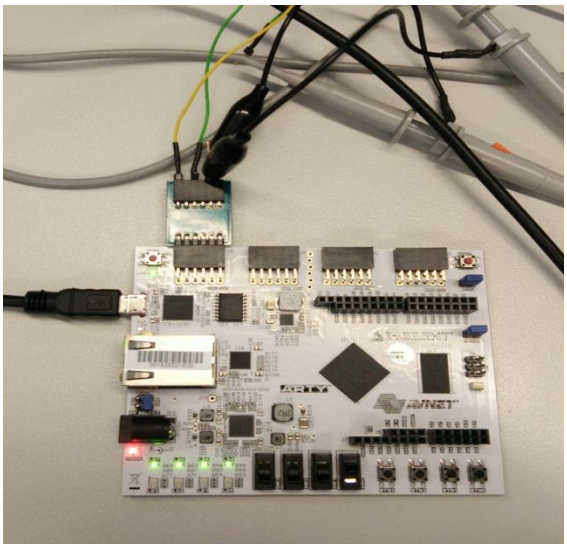
INTRODUCTION

AIM: The aim of this program is to implement SPI communication protocol between a master and slave device. The FPGA acts as a function generator of triangular and sinusoidal waveforms and is the master with DAC acting as Slave. The output of DAC is observed on the CRO.

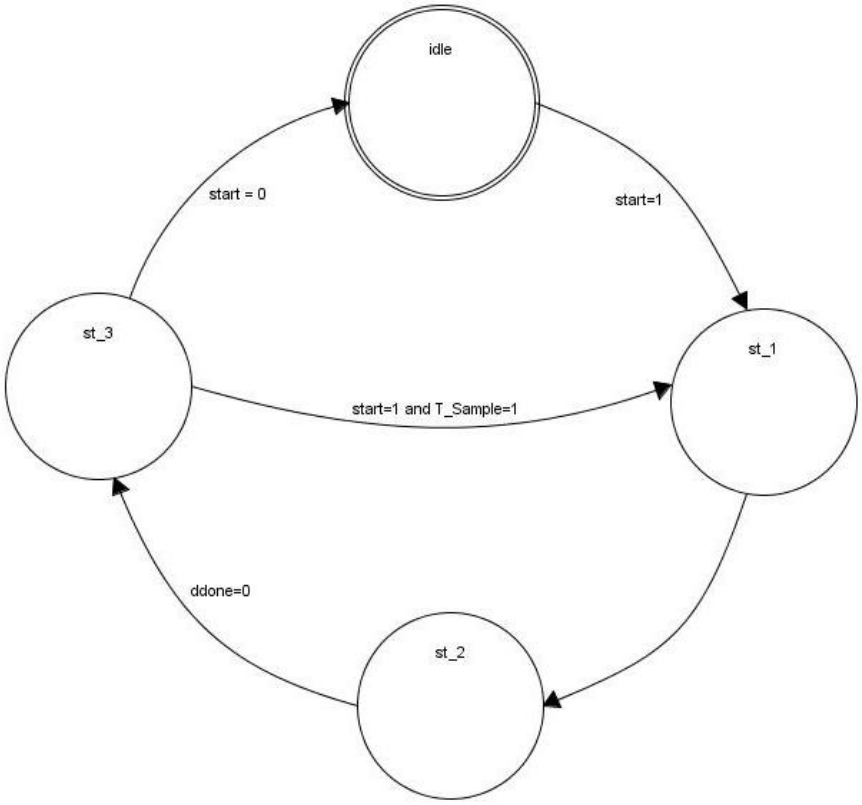
Waveforms observed on CRO

ABSTRACT: A Function generator is developed for triangular and sinusoidal waveforms and their digital values are serially transmitted to the DAC using SPI protocol. The DAC then converts the digital values into analog voltage that is observed on the CRO.

This program is implemented on the **ARTY** (Artix – 7) board. The program is developed and executed on **Xilinx Vivado 2018.3**.



DAC Attached to ARTY



State Diagram of Function Generator

The function generator state machine consists of four states:

1. **idle** : Default state where system is in resetted to
2. **st_1** : State where triangular and sinusoidal waveform sample values are calculated
3. **st_2** : State where the calculated values are transmitted to DAC via SPI
4. **st_3** : Waiting period for the function generator till its time to calculate the next sample.

DAC:

A DAC is a device that converts digital values into corresponding analog voltages. For this implementation, we are using a two input channel DAC each consisting of 12 bits of digital data. Therefore the maximum digital value possible is 2^{12}-1 = 4095. The supply voltage is 3.3V. Therefore the maximum possible output value is

3.3V*4095/4096 =**3.299V**

The DAC used has 6 pins for Supply voltage, Ground, Chip select, Clock and two Data lines.

Function Generator:

A Function generator VHDL code is written to generate two output waveforms. One triangular and one sinusoidal waveform. The frequency of both the waveforms is 10kHz. The sampling rate for each of the waveforms is 160kHz.

Triangular waveform: A triangular waveform is generated by multiplying the values of an up/down counter with some delta value and adding an offset to it.

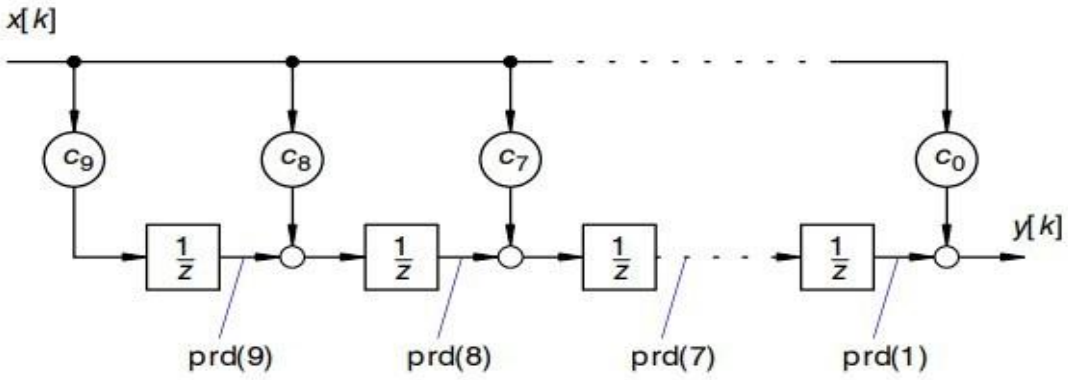
An up-down counter with values 0,1,2,3,4,5,6,7,8,7,6,5,4,3,2,1,0,1,2… is considered. A delta value of 508 is taken to be multiplied with each value of the up down counter. An offset value of 16 is added to each of these values. This will help the triangular wave to expand within the range of maximum DAC value i.e., 4095.

The minimum value is: **(0*508) + 16 = 16**

The maximum value is: **(8*508) + 16 = 4080**

Therefore the triangular wave takes digital values between 16 to 4080 in steps of 508.

Sinusoidal waveform: The sinusoidal waveform is generated by passing the triangular waveform through a FIR low pass filter. A FIR filter of order 9 in transposed form is considered because the maximal combinational path is short and is suitable for calculation in hardware.



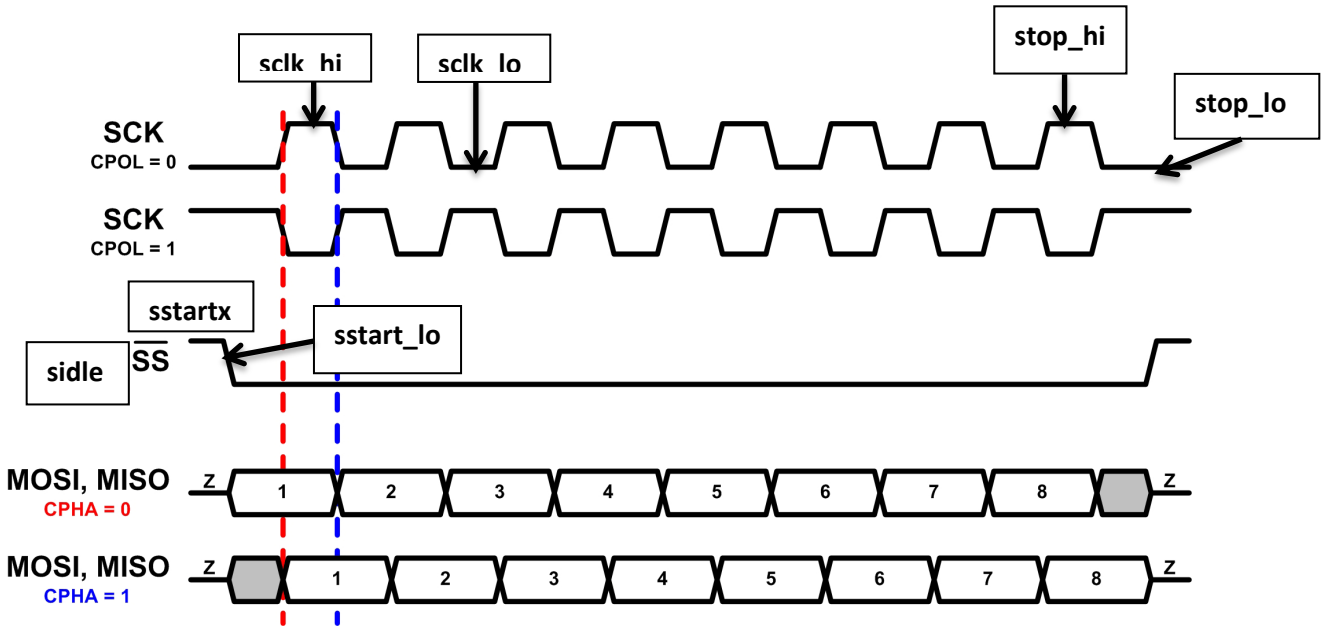
FPGA Suitable FIR Filter

The values of [786, 2136, 5820, 10422, 13604, 13604, 10422, 5820, 2136,786] are taken as filter co-efficients. These are filter co-efficients when expressed in int16.16 i.e., 16 bits of values completely used to express fractional values.

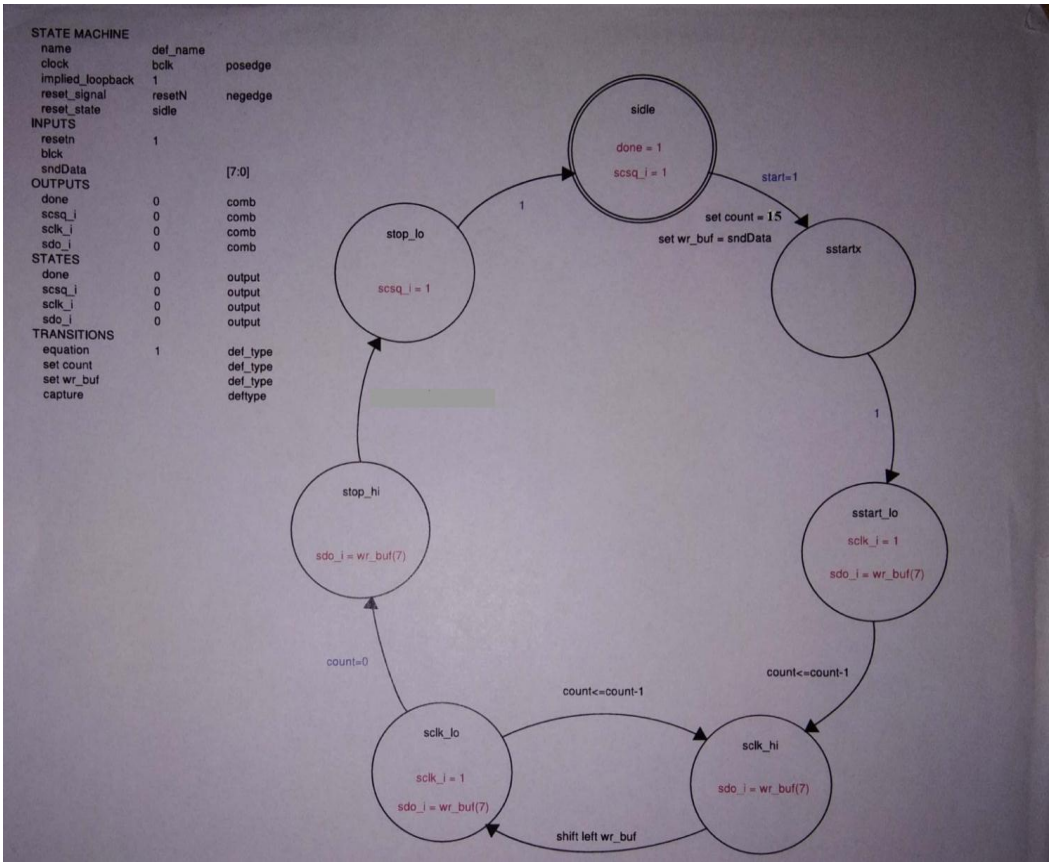
Sreial Peripheral Interface (SPI)

Serial peripheral interface is a technique of two-way communication between two devices. It is fast and simple protocol for the serial transfer of data. In general, SPI consists of four signals Master Out Slave In(MOSI); Master In Slave Out(MISO); clock(clk) and chip select(CS). In this implementation, FPGA is the Master that transmits the two sets of digital data as generated by the function generator to the DAC which is slave.

SPI with CPOL 0 and CPH 1 is used. The below given state diagram is implemented to transmit the data from SPI Master.



Timing Diagram of 8 bit data communication over SPI



State Diagram of SPI Master for data output

VHDL SOURCE CODE

VHDL Source Code for Function Generator (Triangular and Sine wave) – (fctgen.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity fctgen is
   Port ( reset : in STD_LOGIC;
          clk : in STD_LOGIC;
          start : in STD_LOGIC;
          spi_csq : out STD_LOGIC;
          spi_sclk : out STD_LOGIC;
          spi_sd0_0 : out STD_LOGIC;
          spi_sd0_1 : out STD_LOGIC;
          aLED : out STD_LOGIC_VECTOR (3 downto 0));
end fctgen;

architecture Behavioral of fctgen is

-- for simulator
CONSTANT CLK_160KHz : integer := 100;
-- for hardware
--CONSTANT CLK_160KHz : integer := 624;

-- for simulator
CONSTANT SPI_10MHz : integer := 1;
-- for hardware
--CONSTANT SPI_10MHz : integer := 4;
CONSTANT ACNT_MAX : integer := 8;
CONSTANT AMIN : integer := 16;
CONSTANT ADELTA : integer := 508;
SUBTYPE Count_type IS integer RANGE 0 to ACNT_MAX;
SIGNAL cnt : Count_type := 0;
SUBTYPE int_f16 IS integer RANGE -32768 to 32767;
SUBTYPE  int_f32 IS integer RANGE -2147483648 to 2147483647;
SIGNAL sig_u_i : int_f16;
SIGNAL down_dir : std_logic := '0';
CONSTANT FIR_ORDER : integer := 9;
CONSTANT FRACTS : integer := 16;
TYPE  c_array IS ARRAY (0 to FIR_ORDER) of int_f16;
```

```
CONSTANT fcoef : c_array := (786, 2136, 5820, 10422,
            13604, 13604, 10422, 5820, 2136, 786);
TYPE  p_array IS ARRAY (1 to FIR_ORDER) of int_f32;
SIGNAL f_sum : int_f32;
SIGNAL prd : p_array;
SIGNAL y_32 : std_logic_vector(31 downto 0);

TYPE State_type IS (idle, st_1, st_2, st_3);
SIGNAL state : State_type := idle;

CONSTANT SPIDAC_NBITS : integer := 16;

COMPONENT spidacms IS
   GENERIC ( USPI_SIZE : INTEGER := 16 );
   PORT ( resetn : in STD_LOGIC;
        bclk : in STD_LOGIC;
        spi_clkp : in STD_LOGIC;
        dstart : in STD_LOGIC;
        ddone : out STD_LOGIC;
        scsq : out STD_LOGIC;
        sclk : out STD_LOGIC;
        sdo_0 : out STD_LOGIC;
        sdo_1 : out STD_LOGIC;
        sndData_0 : in STD_LOGIC_VECTOR (USPI_SIZE-1 downto 0);
        sndData_1 : in STD_LOGIC_VECTOR (USPI_SIZE-1 downto 0) );
END COMPONENT spidacms;

SIGNAL reset_lo : std_logic;
SIGNAL dstart : std_logic;
SIGNAL ddone : std_logic;
SIGNAL spi_clkp : std_logic;
SIGNAL sig_u : std_logic_vector(15 downto 0);
SIGNAL SndData0_i, SndData1_i : std_logic_vector(SPIDAC_NBITS-1 downto 0);

SIGNAL T_Sample : std_logic;

begin

   aLED <= reset_lo & start & start & ddone;
   reset_lo <= NOT reset;

   --Sample time period for 160kHz
   stp_p : PROCESS(clk)
   variable cnt : integer RANGE 0 to CLK_160KHz := CLK_160KHz;
```

```
BEGIN
   IF rising_edge(clk) THEN
      T_Sample<='0';
      IF cnt=0 THEN
        T_Sample<='1';
        cnt:=CLK_160KHz;
      ELSE
        cnt := cnt -1;
      END IF;
   END IF;
END PROCESS stp_p;

-- 10 MBit/s spi transmission speed
clkp_gen : PROCESS(clk)
VARIABLE cnt : integer RANGE 0 to SPI_10MHz := SPI_10MHz;
BEGIN
   IF rising_edge(clk) THEN
     spi_clkp <= '0';
     IF cnt=0 THEN
        spi_clkp <= '1';
        cnt := SPI_10MHz;
     ELSE
        cnt := cnt - 1;
     END IF;
   END IF;
END PROCESS clkp_gen;

SndData0_i <= std_logic_vector(to_signed(sig_u_i, SndData0_i'length));
y_32 <= std_logic_vector(to_signed(f_sum, y_32'length));
SndData1_i <= y_32(31 downto 16);

sgen_p : PROCESS(clk)
begin
   IF rising_edge(clk) THEN
     dstart <= '0';
     IF reset = '1' THEN
        state<=idle;
        sig_u_i <= AMIN;
        f_sum <= 0;
        FOR k IN 1 to FIR_ORDER LOOP
          prd(k) <= 0;
        END LOOP;
     ELSE
        CASE state IS
```

```
            WHEN idle =>
               IF start='1' THEN
                  state <= st_1;
               END IF;
            WHEN st_1 =>
               --Function Generator
               IF down_dir = '0' THEN
                  IF cnt = ACNT_MAX - 1 THEN
                     down_dir <= '1';
                  END IF;
                  cnt <= cnt+1;
               ELSE
                  cnt <= cnt-1;
                  IF cnt = 1 THEN
                     down_dir <= '0';
                  END IF;
               END IF;
               sig_u_i <= AMIN + cnt*ADELTA ;
               --FIR
               f_sum <= fcoef(0) * sig_u_i + prd(1);
               FOR n IN 1 to FIR_ORDER -1 LOOP
                  prd(n) <= sig_u_i*fcoef(n) + prd(n+1);
               END LOOP;
               prd(FIR_ORDER) <= fcoef(FIR_ORDER) * sig_u_i;
               state<= st_2;
            WHEN st_2 =>
               --Sending to DACs
               dstart <= '1';
               IF ddone ='0' THEN
                  state <= st_3;
               END IF;
            WHEN st_3 =>
               --Wait for sample time period
               IF start='0' THEN
                  state <= idle;
               ELSIF T_Sample = '1' THEN
                  state <= st_1;
               END IF;
         END CASE;
      END IF;
   END IF;
END PROCESS sgen_p;

reset_lo <= NOT reset;
```

```
    pmoddac : spidacms
    GENERIC MAP( USPI_SIZE => SPIDAC_NBITS)
    PORT MAP( resetn => reset_lo,
              bclk => clk,
           spi_clkp => spi_clkp,
            dstart => dstart,
             ddone => ddone,
              scsq => spi_csq,
              sclk => spi_sclk,
             sdo_0 => spi_sd0_0,
             sdo_1 => spi_sd0_1,
          sndData_0 => sndData0_i,
          sndData_1 => sndData1_i);
end Behavioral;
```

VHDL Source Code for SPI Master for DAC – (spidacms.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY spidacms IS
   GENERIC ( USPI_SIZE : INTEGER := 16 );
   PORT ( resetn : in STD_LOGIC;
          bclk : in STD_LOGIC;
          spi_clkp : in STD_LOGIC;
          dstart : in STD_LOGIC;
          ddone : out STD_LOGIC;
          scsq : out STD_LOGIC;
          sclk : out STD_LOGIC;
          sdo_0 : out STD_LOGIC;
          sdo_1 : out STD_LOGIC;
          sndData_0 : in STD_LOGIC_VECTOR (USPI_SIZE-1 downto 0);
          sndData_1 : in STD_LOGIC_VECTOR (USPI_SIZE-1 downto 0) );
END spidacms;

architecture Behavioral of spidacms is

TYPE State_type IS (sidle, sstartx, start_lo, sclk_hi, sclk_lo,  stop_hi, stop_lo );
SIGNAL state, next_state :  State_type;
SIGNAL scsq_i, sclk_i, sdo0_i, sdo1_i : std_logic;
SIGNAL wr_buf0, wr_buf1 : STD_LOGIC_VECTOR (USPI_SIZE-1 downto 0);

SUBTYPE Count_type IS integer RANGE 0 to USPI_SIZE-1;
SIGNAL count : Count_type;

begin

   seq_p :PROCESS(bclk, resetn, next_state, scsq_i, sclk_i, sdo0_i, sdo1_i, count)
   BEGIN
      IF rising_edge(bclk) THEN
         IF resetn = '0' THEN
            state <= sidle;
         elsif spi_clkp='1' then
            IF next_state = sstartx THEN
               count  <= USPI_SIZE-1;
               wr_buf0 <= sndData_0;
               wr_buf1 <= sndData_1;
            ELSIF next_state = sclk_hi THEN
```

```
            count <= count -1;
         ELSIF next_state = sclk_lo THEN
            wr_buf0 <= wr_buf0(USPI_SIZE-2 downto 0) & '-';
            wr_buf1 <= wr_buf1(USPI_SIZE-2 downto 0) & '-';
         ELSE
         END IF;
         state <= next_state;
         scsq <= scsq_i;
         sclk <= sclk_i;
         sdo_0 <= sdo0_i;
         sdo_1 <= sdo1_i;
      END IF;
   END IF;
END PROCESS seq_p;

cmb_p :PROCESS(state, dstart, wr_buf0, wr_buf1,count)
BEGIN
   next_state <= state;
   scsq_i  <= '0';
   sclk_i  <= '0';
   sdo0_i   <= '0';
   sdo1_i   <= '0';
   ddone    <= '0';

   CASE state IS
      WHEN sidle =>
         ddone   <= '1';
         scsq_i <= '1';
         IF dstart = '1' THEN
            next_state <= sstartx;
         END IF;
      WHEN sstartx =>
         next_state <= start_lo;
      WHEN start_lo =>
         next_state <= sclk_hi;
         sclk_i <= '1';
         sdo0_i <= wr_buf0(USPI_SIZE-1);
         sdo1_i <= wr_buf1(USPI_SIZE-1);
      WHEN sclk_hi =>
         next_state <= sclk_lo;
         sdo0_i <= wr_buf0(USPI_SIZE-1);
         sdo1_i <= wr_buf1(USPI_SIZE-1);
      WHEN sclk_lo =>
         IF count = 0 THEN
```

```
            next_state <= stop_hi;
          ELSE
            next_state <= sclk_hi;
          END IF;
          sclk_i <= '1';
          sdo0_i <= wr_buf0(USPI_SIZE-1);
          sdo1_i <= wr_buf1(USPI_SIZE-1);
        WHEN stop_hi =>
          next_state <= stop_lo;
          sdo0_i <= wr_buf0(USPI_SIZE-1);
          sdo1_i <= wr_buf1(USPI_SIZE-1);
        WHEN stop_lo =>
          next_state <= sidle;
          scsq_i <= '1';
      END CASE;
    END PROCESS cmb_p;

end Behavioral;
```

EXPLANATION OF SOURCE CODE

fctgen.vhd

- Library **IEEE** and package **STD_LOGIC_1164** and **NUMERIC_STD** has been added to access data types and functions in the package.
- Define an entity **fctgen** that has 3 inputs and 5 outputs.
- The inputs are :
    1. **reset** : active high reset of type **STD_LOGIC**
    2. **clk** : System Clock of type **STD_LOGIC**
    3. **start** : variable of type **STD_LOGIC** to start the operation
- The outputs are :
    1. **spi_csq** : chip select signal from SPI Master of type **STD_LOGIC**
    2. **spi_sclk** : clock signal from SPI Master of type **STD_LOGIC**
    3. **spi_sd0_0** : serial data output from SPI Master for triangular waveform of type **STD_LOGIC**
    4. **spi_sd0_1** : serial data output from SPI Master for sinusoidal waveform of type **STD_LOGIC**
    5. **aLED** : Used for status indication. It is of type **STD_LOGIC_VECTOR** of size **4**.
- A Signal named **sig_u_i** of type **int_f16 (signed integer from range -32768 to 32767)** is used to store the values associated with the triangular waveform.
- A Signal named **f_sum** of type **int_f32 (signed integer from range - 2147483648 to 2147483647)** is used to store the values associated with the sinusoidal waveform.
- A Signal named **y_32** of type **STD_LOGIC_VECTOR** of size **32** is used to store the values associated with the sinusoidal waveform. Only the upper 16 bits of this signal is used for the values of final sinusoidal output waveform.
- A Signal **state** of Type **State_type** is defined and it consists of states **idle, st_1, st_2** and **st_3**.
- A Signal **T_Sample** is used and it is set to 1 when it is the next sampling interval.
- A **COMPONENT spidacms(**SPI Master) is instantiated with name **pmoddac** that primarily has two serial outputs**.**
- Three Processes are used in the code:
    1. **stp_p:** This process is used to set the value of **T_Sample** to **1** after the completion of every 625 clock cycles. This equates to the sampling frequency of 160kHz as the system clock signal is of 100MHz frequency.
    2. **clkp_gen:** This process is used to generate a clk signal for spi communication. A Signal **spi_clkp** is set to 1 after every 5 system clock cycles. This generates a clock signal of 20MHz. This in turn makes the transmission speed of SPI **10Mbit/s**.
    3. **sgen_p :** In this process, several actions under the four states take place.
        - **idle:** This is the default state at the beginning. If the start button is pushed to 1, then the state changes to **st_1**.

- **st_1 :** In this state, the values of triangular and sinusoidal waveform are generated for that particular sampling interval. Then the state is changed to **st_2.**
    - **Triangular waveform:** An up-down counter is used with values between 0 and 8. Each of these values is multiplied by a delta value of 508 and an offset of 16 is added.

$$u_{tr} = A_{min} + kA_{delta}$$

    - **Sinusoidal waveform:** This waveform is generated by passing the triangular waveform through a FIR Low pass filter of order 9. A for loop is used to calculate products 1 to 8. Product 9 is calculated separately as it doesn't require a sum. The final product 1 is added with the product of input signal and 0^{th} coefficient of filter to obtain the value of the sinusoidal waveform.
- **st_2:** In this state, the values calculated in **st_1** are transmitted via SPI to DAC. Therefore the variable **dstart** signalling the start of transmission is set to **1**. At the end of transmission of the value state is changed to **st_3.**
- **st_3:** In this state, we wait for the next sampling interval. When **T_Sample** is set to 1 we switch the state to **st_1.**

spidacms.vhd

- Library **IEEE** and package **STD_LOGIC_1164** and **NUMERIC_STD** has been added to access data types and functions in the package.
- Define an entity **spidacms** that has 6 inputs and 5 outputs.
- The inputs are :
    1. **resetn** : active high reset of type **STD_LOGIC**
    2. **bclk** : System clock of type **STD_LOGIC**
    3. **spi_clkp** : Clock of type **STD_LOGIC** that is given to dac
    4. **start** : variable of type **STD_LOGIC** to start the operation
    5. **sndData_0** : variable of type **STD_LOGIC_VECTOR** of size 16
    6. **sndData_1** : variable of type **STD_LOGIC_VECTOR** of size 16
- The outputs are :
    1. **ddone** : used to indicate transmission completion and of type **STD_LOGIC**
    2. **scsq** : chip select signal from SPI Master of type **STD_LOGIC**
    3. **sclk** : clock signal from SPI Master of type **STD_LOGIC**
    4. **sdo_0** : serial data output from SPI Master for triangular waveform of type **STD_LOGIC**
    5. **sdo_1** : serial data output from SPI Master for sinusoidal waveform of type **STD_LOGIC**
- Signals **state** and **next_state** of Type **State_type** are used which consists of states **sidle, sstartx, start_lo, sclk_hi, sclk_lo, stop_hi, stop_lo**
- Two processes are used in the code:
    1. **seq_p:** This process is used to perform the sequential operations. Transition operations take place when **spi_clkp** is '1'. The following are performed:
        - If **next_state** is **sstartx**, count is initialised to 15 to perform serial write of the final 15 bits of data in subsequent states.
        - If **next_state** is **sclk_hi,** count is decremented.
        - If **next_state** is **sclk_lo,** the data is left shifted.
    2. **cmb_p:** This process is used to perform combinational operations. A Case statement is used to specify operations in different states.
        - **sidle :** done and chip select are set to 1 and if **dstart** is set to high, state transition starts and **sstartx** becomes the **state** in next clock cycle.
        - **sstartx : start_lo** is assigned to next_state.
        - **start_lo : sclk_hi** is assigned to next_state and MSB data of **sndData_0** and **sndData_1** is written onto **sdo_0** and **sdo_1**.
        - **sclk_hi : sclk_lo** is assigned to **next_state** and MSB data of **sndData_0** and **sndData_1** is written onto **sdo_0** and **sdo_1**.
        - **sclk_lo :** If **count** is non-zero, next state is **sclk_hi** otherwise it is **stop_hi**.The left shifted data is written onto **sdo_0** and **sdo_1**.
        - **stop_hi : stop_lo** is assigned to next state and data in MSB of **sndData_0** and **sndData_1** is written onto **sdo_0** and **sdo_1**.
        - **stop_lo :** chip select line is set to 1 and **sidle** is assigned to next state.

USER CONSTRAINTS FILE

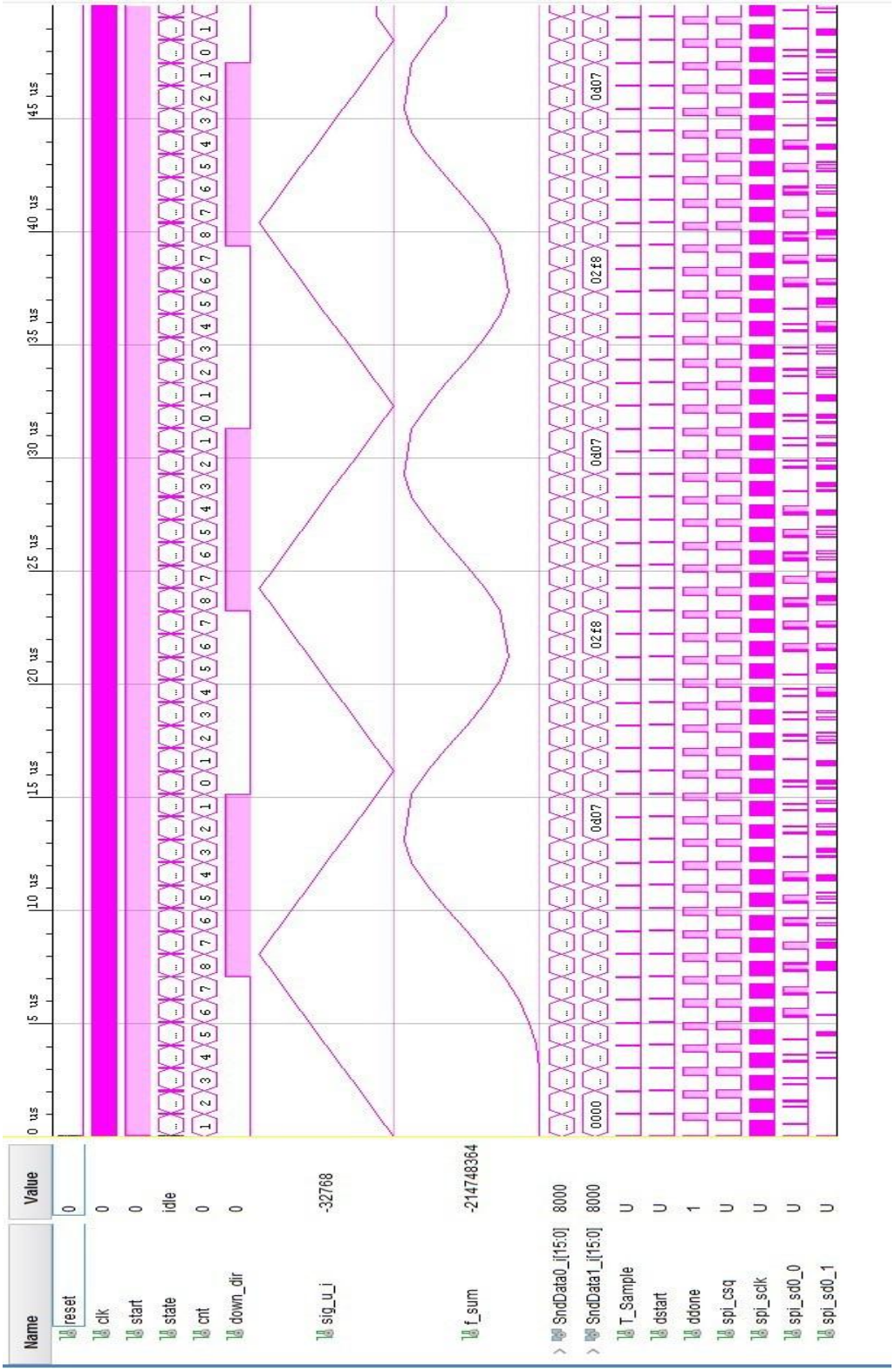
```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L12P_T1_MRCC_35 Sch=gclk[100]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];

## Switches
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { start }];
#IO_L12N_T1_MRCC_16 Sch=sw[0]

## Buttons
set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L6N_T0_VREF_16 Sch=btn[0]

##LEDs
set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports { aLED[0] }]; #IO_L24N_T3_35 Sch=led[4]
set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 } [get_ports { aLED[1] }]; #IO_25_35 Sch=led[5]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { aLED[2] }];
#IO_L24P_T3_A01_D17_14 Sch=led[6]
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { aLED[3] }];
#IO_L24N_T3_A00_D16_14 Sch=led[7]

##Pmod Header JA
set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { spi_csq }]; #IO_0_15 Sch=ja[1]
set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { spi_sd0_0 }]; #IO_L4P_T0_15 Sch=ja[2]
set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33 } [get_ports { spi_sd0_1 }]; #IO_L4N_T0_15 Sch=ja[3]
set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33 } [get_ports { spi_sclk }]; #IO_L6P_T0_15 Sch=ja[4]

## Voltage config
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
```

VHDL TESTBENCH

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fctgen_tb is
end fctgen_tb;

architecture Behavioral of fctgen_tb is

COMPONENT fctgen is
   Port ( reset : in STD_LOGIC;
          clk : in STD_LOGIC;
          start : in STD_LOGIC;
          spi_csq : out STD_LOGIC;
          spi_sclk : out STD_LOGIC;
          spi_sd0_0 : out STD_LOGIC;
          spi_sd0_1 : out STD_LOGIC;
          aLED : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT fctgen;

SIGNAL reset :  STD_LOGIC := '0';
SIGNAL clk :  STD_LOGIC:= '0';
SIGNAL start :  STD_LOGIC:= '0';
SIGNAL spi_csq :  STD_LOGIC:= '0';
SIGNAL spi_sclk :  STD_LOGIC:= '0';
SIGNAL spi_sd0_0 :  STD_LOGIC:= '0';
SIGNAL  spi_sd0_1 :  STD_LOGIC:= '0';
SIGNAL  aLED : STD_LOGIC_VECTOR(3 DOWNTO 0) := x"0";

CONSTANT clk_period :time := 10ns;

begin

   uut : fctgen
   PORT MAP ( reset =>reset,
          clk  =>clk,
          start  => start,
          spi_csq => spi_csq,
          spi_sclk  => spi_sclk,
          spi_sd0_0  => spi_sd0_0,
          spi_sd0_1  =>spi_sd0_1,
          aLED  =>aLED);
```

```
    clk_p : PROCESS
    BEGIN
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    END PROCESS clk_p;

    stim_p : PROCESS
    BEGIN
        wait for clk_period;
        reset <= '1';
        wait for clk_period;
        reset <= '0';
        wait for clk_period;
        start <= '1';
        wait;
    END PROCESS stim_p;
end Behavioral;
```

SIMULATION WAVEFORMS



SYNTHESIS REPORT

Start Writing Synthesis Report

Report BlackBoxes:

```
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+
```

Report Cell Usage:

```
+------+----------+------+
|      |Cell      |Count |
+------+----------+------+
|1     |BUFG      |     1|
|2     |CARRY4    |     5|
|3     |DSP48E1_1 |     8|
|4     |DSP48E1_2 |     1|
|5     |DSP48E1_3 |     1|
|6     |LUT1      |     5|
|7     |LUT2      |     8|
|8     |LUT3      |    17|
|9     |LUT4      |    13|
|10    |LUT5      |    35|
|11    |LUT6      |     9|
|12    |FDRE      |    94|
|13    |FDSE      |     1|
|14    |IBUF      |     3|
|15    |OBUF      |     8|
+------+----------+------+
```

Report Instance Areas:

```
+------+----------+---------+------+
|      |Instance  |Module   |Cells |
+------+----------+---------+------+
|1     |top       |         |   209|
|2     |  pmoddac |spidacms |    90|
+------+----------+---------+------+
```

Finished Writing Synthesis Report : Time (s): cpu = 00:01:47 ; elapsed = 00:01:55 . Memory (MB): peak = 813.355 ; gain = 537.934

Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:01:12 ; elapsed = 00:01:34 . Memory (MB): peak = 813.355 ; gain = 214.508
Synthesis Optimization Complete : Time (s): cpu = 00:01:48 ; elapsed = 00:01:56 . Memory (MB): peak = 813.355 ; gain = 537.934
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 15 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 1 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.001 . Memory (MB): peak = 818.875 ; gain = 0.000
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

INFO: [Common 17-83] Releasing license: Synthesis
38 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:01:54 ; elapsed = 00:02:01 . Memory (MB): peak = 818.875 ; gain = 555.492
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 818.875 ; gain = 0.000
WARNING: [Constraints 18-5210] No constraints selected for write.
Resolution: This message can indicate that there are no constraints for the design, or it can indicate that the used_in flags are set such that the constraints are ignored. This later case is used when running synth_design to not write synthesis constraints to the resulting checkpoint. Instead, project constraints are read when the synthesized design is opened.
INFO: [Common 17-1381] The checkpoint 'D:/ESD/Sem1/VHDL/Lab/Code/DTVL3/fctgen/fctgen.runs/synth_1/fctgen.dcp' has been generated.
INFO: [runtcl-4] Executing : report_utilization -file fctgen_utilization_synth.rpt -pb fctgen_utilization_synth.pb
INFO: [Common 17-206] Exiting Vivado at Sat Jun 8 16:26:36 2019...

CONCLUSION

In this lab, we implemented a function generator on **ARTY** board that generates a triangular and sinusoidal waveform. The 12 bit digital values of each of the waveforms are transmitted on two serial data output lines from FPGA to DAC via the SPI communication protocol. Here the FPGA acts as a master and DAC acts as a slave device. The output of DAC is then given onto a CRO on which we observe the analog waveforms.