School of Engineering

Department of Electrical and Electronic Engineering

Microelectronics: Systems and Devices: MSc

EEE8088: Reconfigurable Hardware Design

Final Report

X.Liu

May 2020

EEE8088 Reconfigurable Hardware Design Report

Xinjie Liu
School of Engineering
Newcastle University, NE1 7RU
United Kingdom

X.Liu81@newcastle.ac.uk

I. Aims and objectives

FPGA is one of common use semiconductor devices in industries, which have a variety of advantages including can be reprogrammed to implement different logic functions and enable system performance simulation while developing systems. In this coursework, we studied VHDL and developed a Digital Audio Filter system on FPGA. This system includes three blocks, Codec initialization, serial to parallel & parallel to serial, FIR filter. All the works depended on Quartus II platform.

II. Background

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Altera, Xilinx and Achronix are the only three manufacturers on FPGA market. In this coursework, Altera Cyclone V FPGA is used, which is carry by Altera DE1 board which have many features. Figure 1 shows Altera DE1 board and figure 2 shows the block diagram of DE1 board. This board carrying Cyclone V SoC 5CSEMA5F31C6 Device with Dual-core ARM Cortex-A9 processor, which is suitable for developing audio digital filter system. Line in, Audio codec, and the main processor is used in this board.

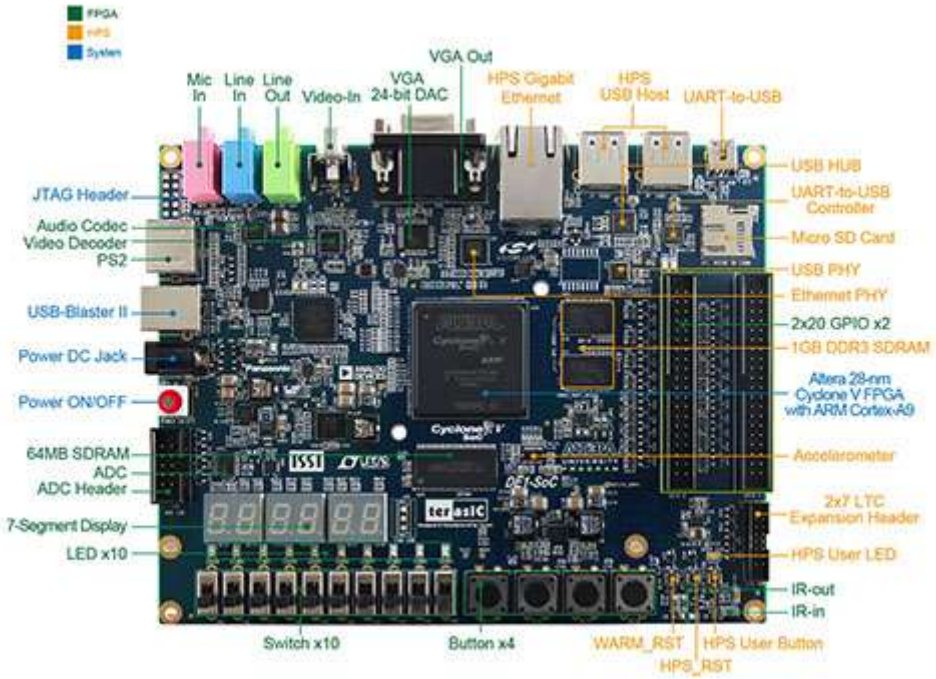


Figure 1: Altera DE1 PCB

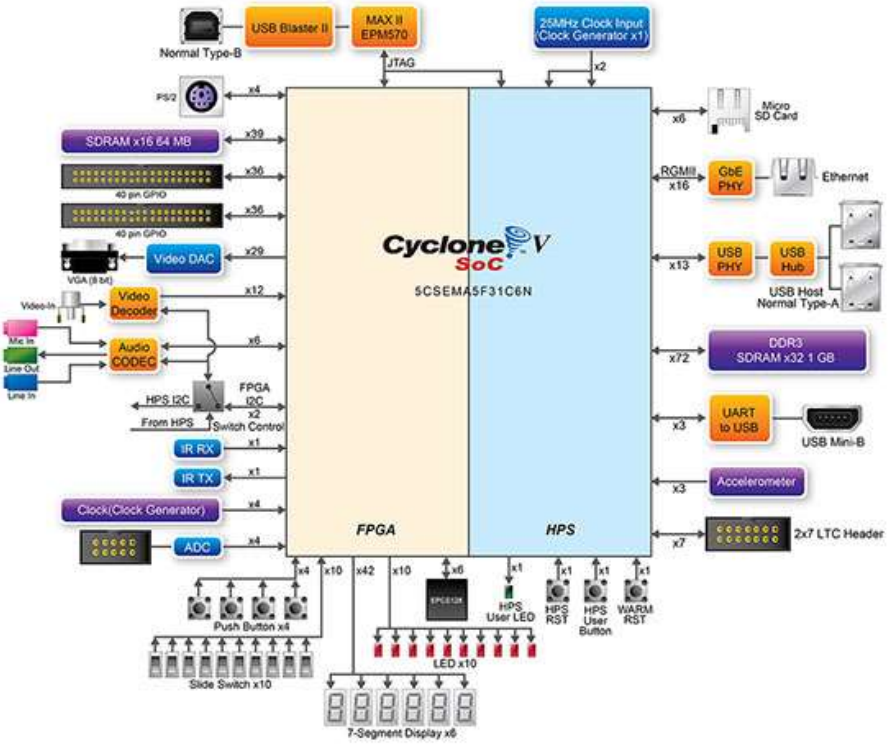


Figure 2: Block Diagram of the DE1-SOC Board

III. Detailed Objectives

In this coursework, 3 digital blocks were described with VHDL in this project, shown in figure 3-6, including Codec initialization, serial to parallel & parallel to serial, FIR filter. The input port is set as blue and out is green in those schematics. Figure 3 demonstrate the system diagram of this coursework. The audio signal come from Line in converted by WM8731 to Digital serial signal, codec_init block giving a protocol configuration to WM8731 DSP. The configuration parameter is provided by coursework requirement. The s2p_adaptor can convert serial to parallel signal, it also covert parallel signal to serial signal for sending to DAC, as well as undertake the ready and strobe signal. The FIR filter is the most important part in this system, which can reject the unwanted parts of signal like noise, crosstalk or unwanted frequency data. That means only certain frequency signal can pass the filter. In this coursework, from the figure 7, with the frequency increase, gain drop down. so, we can determine the FIR filter is a low pass filter.

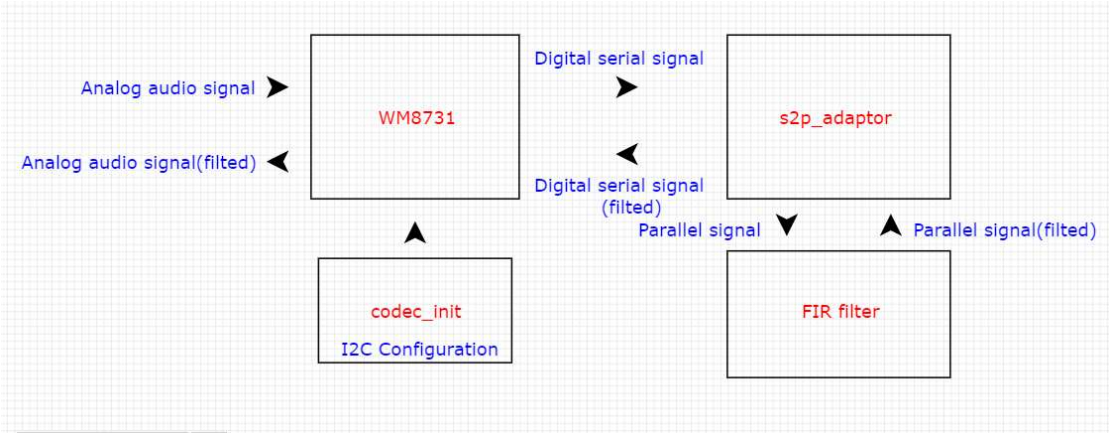


Figure 3: system diagram

III. 1 Codec initialization

Figure 4 giving a codec block diagram. The feature of codec initialization block is giving a protocol configuration to DSP. The control interface used in this FPGA experiment is 2-wire serial interface I2C. With WM8731 DSP, the maximum clock is up to 500kHz. And the digital audio interface is I2S, which carrying the Input and output digital audio streams. The speed of it is 64x44.1 kHz, that means the codec sample rate is 44100 Hz. The DSP provide left and right channel. Only left channel selected in this design.

Figure 5 shows the I2C protocol and figure 6 shows the output and internal signal waveform of this block. This protocol is to determine the Start and Stop state and sending 3 ACK (acknowledge) signal when data has been received by next block. SDIN include 3 bytes of information which is totally 24 bits. In these 24 bits, the first 7 bits ([15:9]) is Control Address bits, and 1bit of read or write, 7bits of register map address, 9 bits of register data [1].

In codec_init block, there are 3 counters: frequency divider, bit counter and word counter. By giving functions to these counters, the SCLK and SCIN can successfully output.

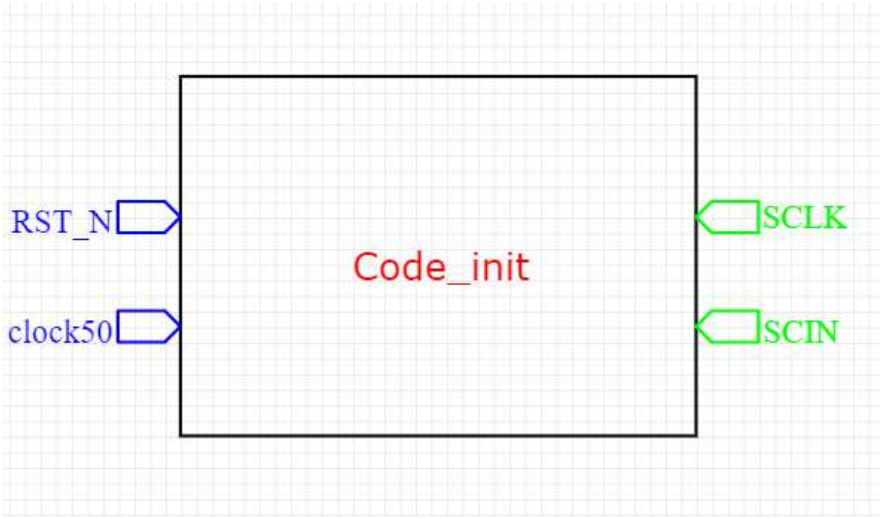


Figure 4: Codec block diagram

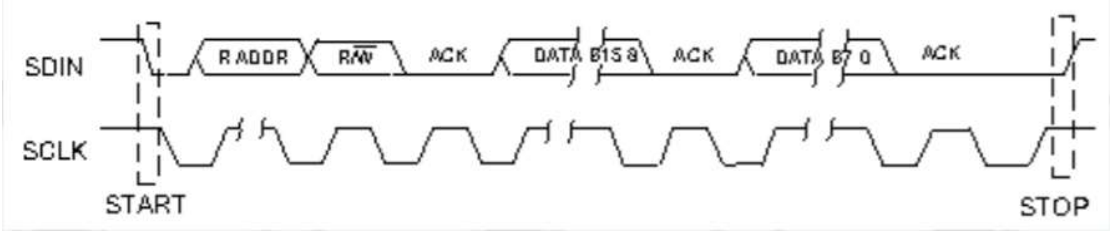


Figure 5: I2C Bus Format [1]

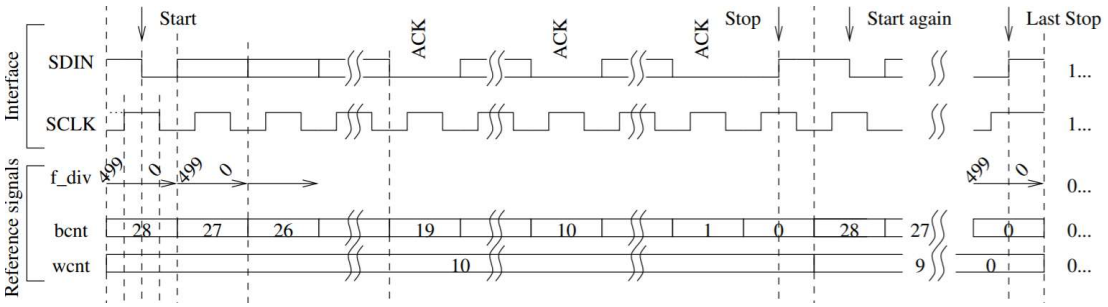


Figure 6: I2C protocol output& internal signal waveform

III. 2 Serial to Parallel & Parallel to Serial adaptor

Figure 7 present a s2p_adaptor block diagram. This block can covert the serial signal sent from DSP to Parallel signal. Because only parallel signal can be processed by FIR filter. This block run on 50MHz and need to detect the rising edge from DSP clock and Strobe, ready signal to enable output parallel signal, which is DACDAT, Shown in figure 8. Figure 9 demonstrate the parallel to serial part, when Ready, rising edge and strobe come in, the audio output channel produces serial output signal. To detect whether the clock is on rising edge or positive edge, a register can be used to storage pervious clock state and compare with current clock state to determine rising edge.

According to Ready and Strobe signal, Strobe signal indicate whether this block is busy or idle to receive

the new data. Ready signal is generated by next block and having same feature as strobe. In this coursework, when the parallel is ready to send to filter, the Strobe will be set as 1. And when the ready is set as 1 which mean filter is idle, the data can transfer from s2p_adaptor to FIR filter. Same from filter back to s2p_adaptor. The only difference between Analog to Digital route and Digital to Analog route in this block is the port and internal signal is different.

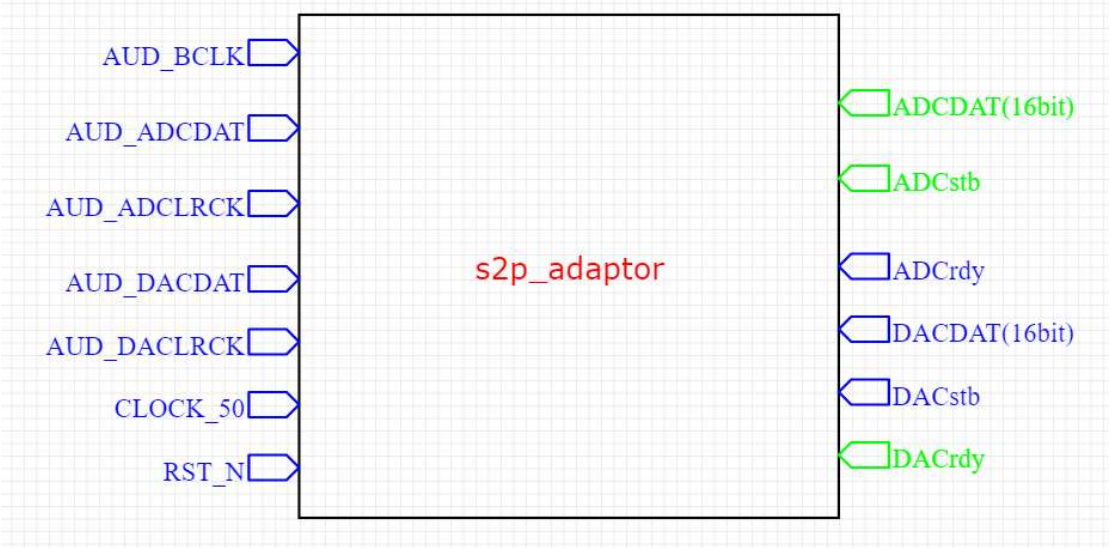
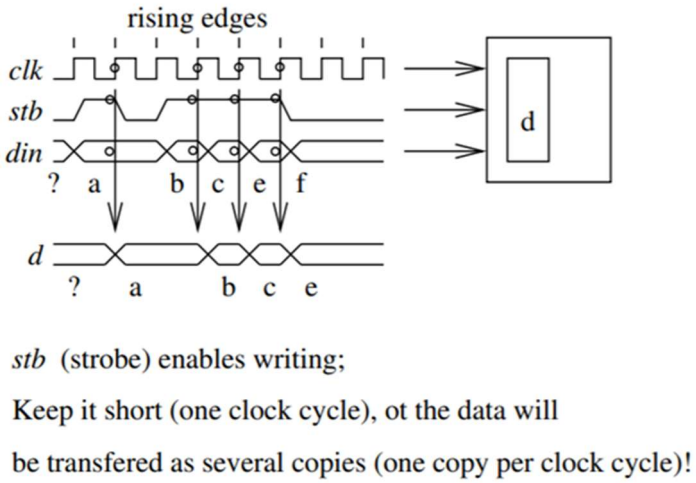


Figure 7: s2p_adaptor block diagram



stb (strobe) enables writing;

Keep it short (one clock cycle), ot the data will

be transfered as several copies (one copy per clock cycle)!

Figure 8: Parallel interface with strobe signal

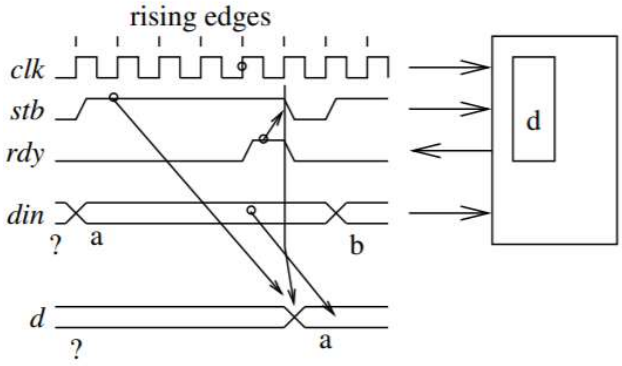


Figure 9: Parallel interface with ready signal

III. 3 FIR filter

FIR (Finite Impulse Response) filter is a non-recursive filter in digital design, which means the output settles to zero in finite time. [2] Figure 10-11 demonstrate FIR filter block diagram and its taps. This filter relies on large number of coefficients and taps. Figure 11 demonstrate the frequency respond of this filter. Equation 1 shows the frequency response of FIR filter:

$$y(n) = \sum_{k=0}^{M} x(n-k)b(k)$$

Equation 1: frequency response of FIR filter

In equation 1, $b(k)$ is the coefficients of the filter, which presented in figure 12.

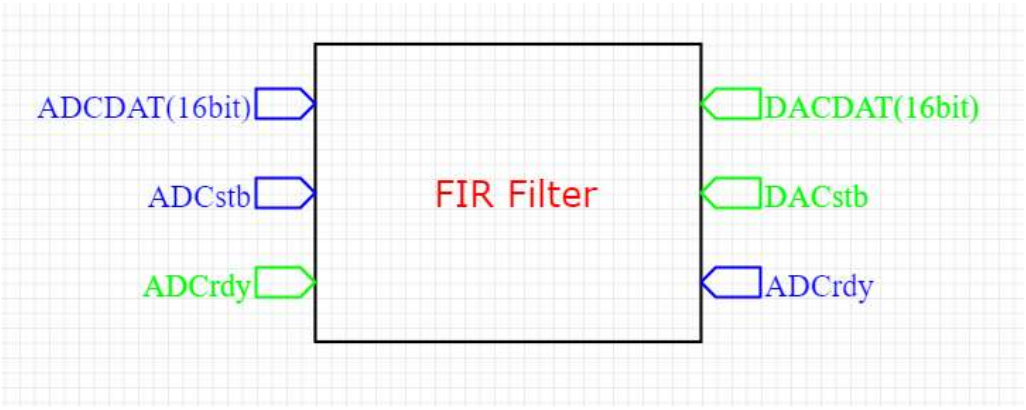


Figure 10: FIR filter block diagram

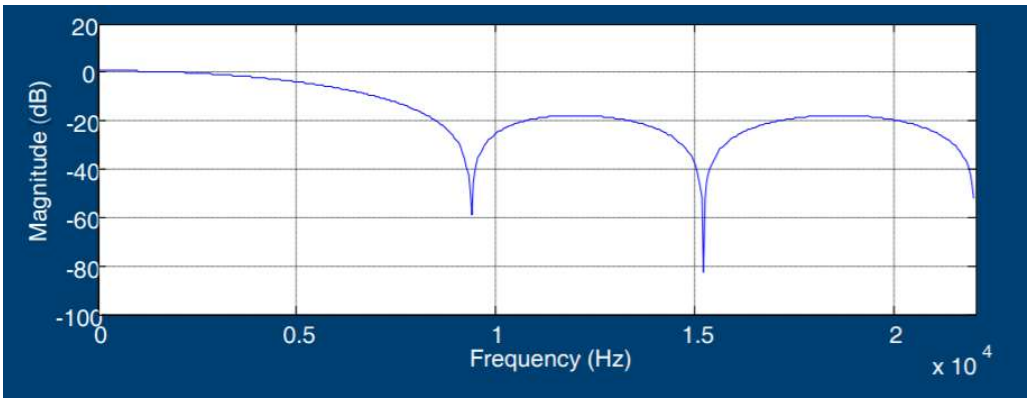


Figure 11: Frequency response given from slides

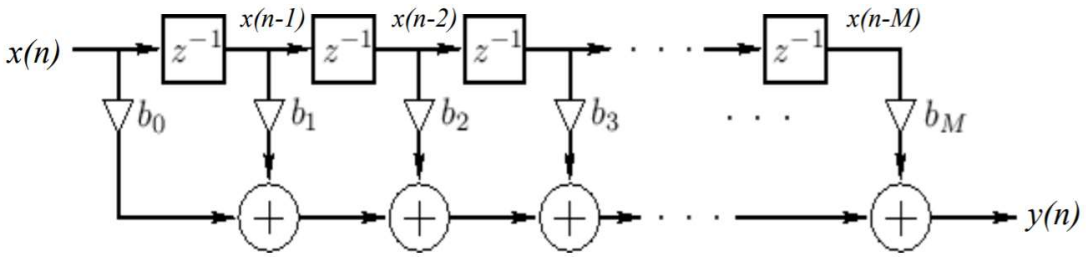


Figure 12: FIR filter taps

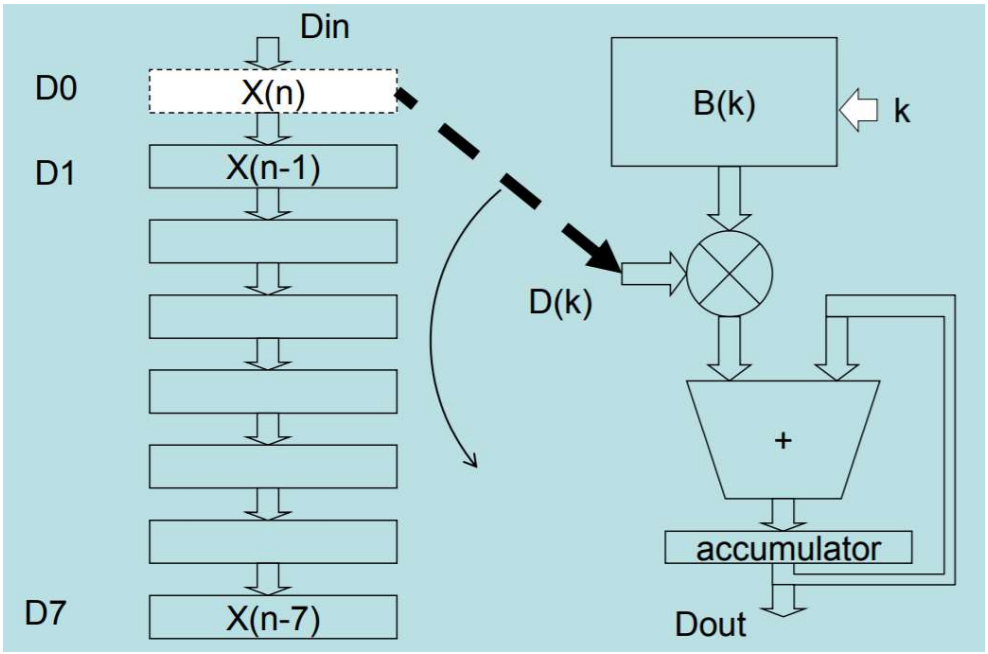


Figure 13: realization of FIR filter

This FIR filter includes 8 taps, the coefficients are -1260, 7827, 12471, 16384, 16384,12471, 7827, -1260. To build this filter, the easiest way is use multipliers to multiply the input data and coefficients. The register of multiplier must be 32 bits to contain two 16 bits number multiply. Finally, the first 16 bits data of multiplier will be transfer to output. At the same time change the strobe signal.

IV. Result and Discussion

IV.1 Codec initialization

In testbench, a 50MHz clock and a reset 1 signal is given to this circuit.

Figure 14: codec_init simulation result

Figure 14 demonstrate the codec_init simulation result. Frequency divider start from 499 to 0, bit counter start from 28 to 0 and word counter start from 10 to 0. When frequency divider counts down to 375, the SCLK set to 1. Continue counts down to 125, SCLK set to 0. Start from 'wcnt1' 27, the waveform is "00110100 ACK 000111", which is same as the sdin_load given by template.

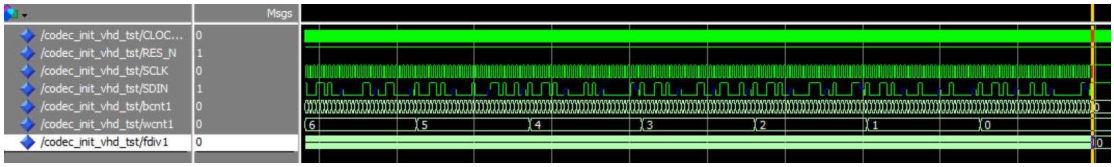


Figure 15: codec_init simulation result 2

Figure 15 giving the I2C protocol simulation result. In every word, there are 3 Z signal which are acknowledgement signal. As well as 1 start signal and 1 stop signal. The other 19 bits is data. The output stops when word counter count to 0, which is "deadlock". If this block need to active again, reset signal must be gave to the circuit to reset all the counter.

IV.2 Serial to Parallel & Parallel to Serial adaptor

50MHz clock, 44.1kHz DSP clock, 22.05kHz audio input signal, ready and reset signal is given to this circuit in simulation.

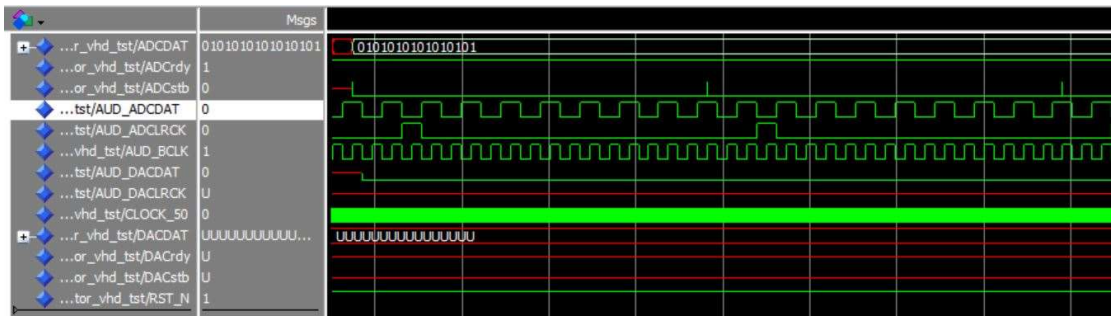


Figure 16: Serial to Parallel part simulation result

Figure 16 shows the when AUD_ADCLRCK raise to 1, the circuit start to capture AUD_ADCDAT signal to register. After the counter full of 16 bits data, the ADCstb will give a strobe and the ADCDAT give a 16 bits parallel data.

In this simulation, ADCDAT is connected to DACDAT. Other signal is same as serial to parallel part.

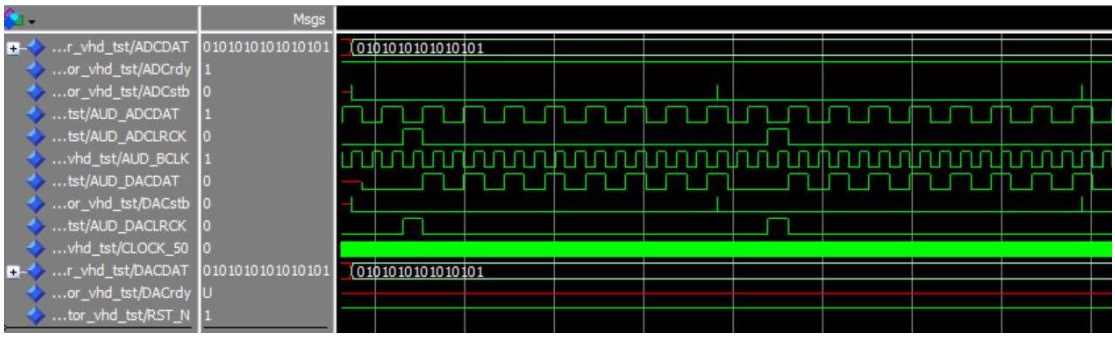


Figure 17: Parallel to Serial part simulation result

Figure 17 demonstrate that AUD_ADCLRCK rising to 1, the circuit start to capture the parallel signal DACDAT and convert to serial signal every 22.68μs. After every 16 bits parallel signal converted, a DACrdy giving a strobe to show the parallel to serial part is ready. In figure 17 the DACrdy is "U", because this signal usually transfer from filter block. This simulation only includes s2p_adaptor block.

IV.3 FIR filter

In this experiment, the approximate sine waveform generator is used. 500Hz, 1kHz, 2kHz and 100kHz sin wave signal is given to circuit and the result is in figure 18-21. Even the amplitude cannot be detected with simulation, but from the binary signal given, this circuit can be determined as a low-pass filter. When frequency goes up, the difference of input and output binary number is become huger.

There are many ways to build a sine wave generator in VHDL testbench. As far as I concern, there are 3 different way to build it. First one is using a Sine function to generate the data and multiply the amplitude. The second one is building a ROM a store the sine table in the testbench [4], which is the easier than first one, but needs to use frequency divider to change the frequency. The last one is CORDIC [5], that is like the first method but more commonly used in FPGA, there is an example [6].

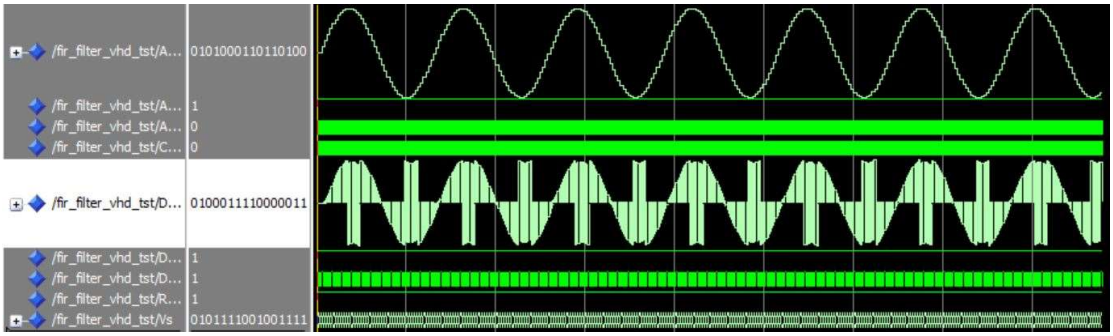


Figure 18: 500Hz FIR filter simulation result

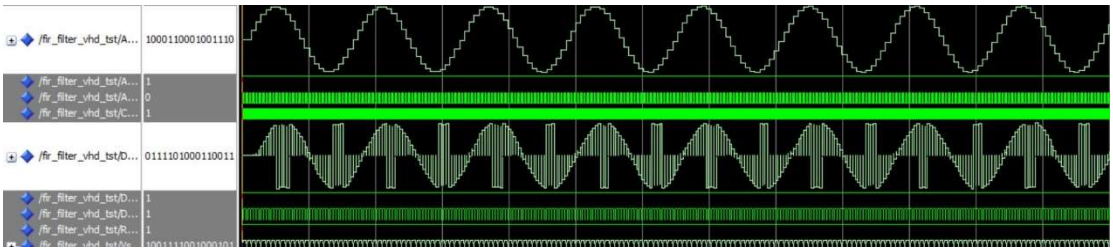


Figure 19: 1kHz FIR filter simulation result

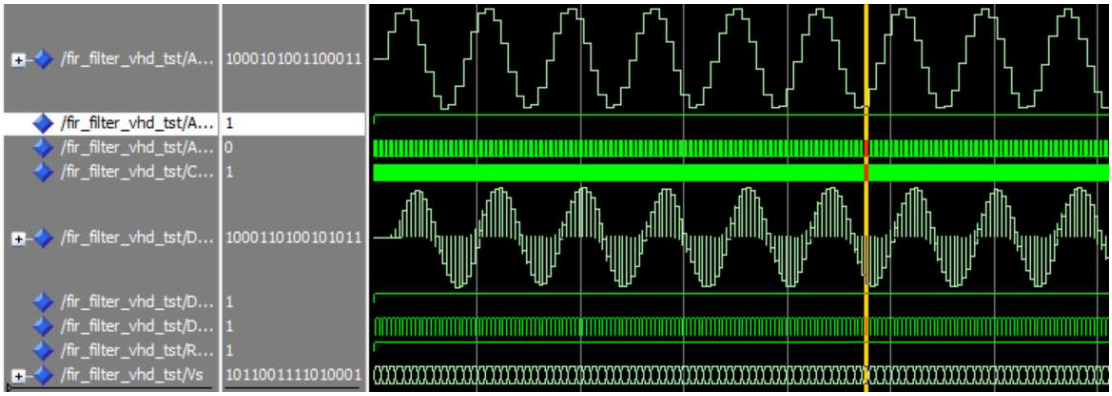


Figure 20: 2kHz FIR filter simulation result

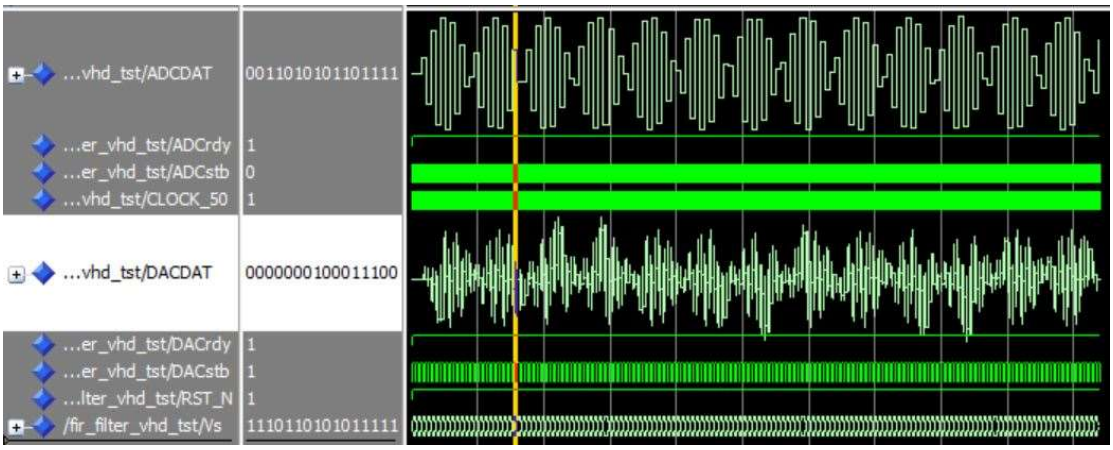


Figure 21: 100kHz FIR filter simulation result

V. Conclusion

This coursework giving an overview on audio filter system on FPGA. The result shows codec initialization, serial to parallel & parallel to serial and FIR filter implementation is successful. Codec initialization block utilize configuration with DSP for using I2C protocol to control the transmit audio signal and receive filtered data from filter. This block sends 11 words signal, which include the Start, Stop, Acknowledgement signal and configuration data.

Serial to parallel part of s2p_adaptor covert left channel serial audio signal sends from DSP and covert to 16 bits parallel signal. The parallel part covert the parallel signal processed from filter with the clock given by DSP to serial signal. In addition, the s2p_adaptor transmit the Strobe and Ready signal to next block to inform whether this block and next block is processing data or not.

FIR filter is the most important block in this coursework, which is consists of shifter and accumulator. This filter has 8 taps, each tap is a shifter and the final tap accumulate those shifter result and giving a 16 bits parallel data to output, which send back to s2p_adaptor for further process.

Finally, the sine wave generator is used in testing FIR filter, the principle of 3 different sine wave generator is discussed. However, this design cannot deploy to Altera DE1 FPGA, so the cut off frequency cannot be scan out by oscilloscope to verify.

VI. Reference

[1] WM8741 datasheet: https://www.alldatasheet.com/datasheet-pdf/pdf/211089/WOLFSON/WM8741.html

[2] FIR wiki https://en.wikipedia.org/wiki/Finite_impulse_response

[3] Jassim, Manal H., and Asaad Hameed Sahar. "High-Pass Digital Filter Implementation Using FPGA." IRAQI JOURNAL OF COMPUTERS, COMMUNICATION AND CONTROL & SYSTEMS ENGINEERING 13.3 (2013): 41-50.

[4] ROM sine wave generator: https://surf-vhdl.com/how-to-generate-sine-samples-in-vhdl/

[5] CORDIC https://en.wikipedia.org/wiki/CORDIC

[6] Using a CORDIC to calculate sines and cosines in an FPGA https://zipcpu.com/dsp/2017/08/30/cordic.html

VII. Appendix

Codec_init

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3.
4. entity codec_init is
5.   port
6.   (
7.          CLOCK_50        : in  std_logic;
8.          RES_N           : in  std_logic;
9.          SCLK            : out std_logic;
10.                 SDIN            : out std_logic;
11.                 bcnt1    : out integer;
12.                 wcnt1    : out integer;
13.                 fdiv1    : out integer
14.         );
15.
16.  end entity;
17.
18.  architecture rtl of codec_init is
19.
20.  constant sdin_load : std_logic_vector (11*24-1 downto 0) :=
21.  b"0011010_0_0001111_000000000"&
22.  b"0011010_0_0000000_000011111"&
23.  b"0011010_0_0000001_000110111"&
24.  b"0011010_0_0000010_001111001"&
25.  b"0011010_0_0000011_000110000"&
26.  b"0011010_0_0000100_011010010"&
27.  b"0011010_0_0000101_000000001"&
28.  b"0011010_0_0000110_001100010"&
29.  b"0011010_0_0000111_001000011"&
30.  b"0011010_0_0001000_000100000"&
31.  b"0011010_0_0001001_000000001";
32.  -- 11 words, the first is reset (R15), the others are
   registers R0-9.
33.  -- each word is 24 bit constructed as
```

```
34.  -- chip address, r/w bit, reg address, reg data
35.  -- these words do not include start, stop and ack bits, see
     packet format below
36.
37.
38.  -- Packet format:                                (bit number)
39.
40.  ---- start bit                                   28
41.  -- 7 bits chip address,
42.  -- 1 r/w bit,
43.  --** ack                                         19
44.  -- 8 high bits of reg. data,
45.  --** ack                                         10
46.  -- 8 low bits of reg. data,
47.  --** ack                                         1
48.  ---- stop bit                                   0
49.
50.
51.
52.  -- reg. data = 7 bit address + 9 bit config data, 16 bits
     total,
53.  -- split as 8+8 bits in the packet, MSB go first.
54.
55.
56.  -- declare a shift register
57.  signal sht_r : std_logic_vector (11*24-1 downto 0);
58.  -- declare an internal signal to be copied into SIDN
59.
60.  -- declare the bit counter;  -- bit counter, runs at 100kHz,
61.  signal bcnt: integer range 28 downto 0:=28;
62.  -- bits 28, 19, 10, 1 and 0 are special
63.  -- declare the word counter; -- word counter, runs at about
     5kHz
64.  signal wcnt: integer range 10 downto 0:=10;
65.
66.  -- declare the counter for the bit length; -- frequency
     divider counter,
67.  -- runs at 50MHz
```

```
68.
69.  signal f_div: integer range 499 downto 0:=499;
70.
71.
72.
73.  begin
74.
75.          process (CLOCK_50)
76.          begin
77.           if (rising_edge(CLOCK_50)) then
78.                  ----------------------------
79.                  -- reset actions
80.                          if (RES_N = '0') then
81.                          -- reset the counters to an
   appropriate state
82.                              f_div <=499;        -- load the
   frequency divider,
83.                                                  -- 50MHz/500=100kHz bus
   speed
84.                           sht_r <= sdin_load;    -- load the
   shift register
85.                              bcnt<=28;    -- load the bit
   counter,
86.                                                  -- 29 bits in the word
   protocol
87.                              wcnt<=10;    -- load the word
   counter, 11 words
88.                              -- reset the outputs to an
   appropriate state
89.                                   SDIN<='1';
90.                                   SCLK<='0';
91.
92.                  -- deadlock in the end
93.
94.                          elsif (wcnt=0 and bcnt=0 and f_div=0)
   then
95.                               SCLK <= '1';
96.                          -- do nothing, wait for the next reset
```

```
97.
98.                     -- modify reference counters
99.                   -- for frequency divider, bits and words
100.
101.                        -- modify reference counters
102.                    -- for frequency divider, bits and words
103.                        elsif (f_div=0) then -- at the end of
    each bit
104.                                    f_div<=499; -- reload the
    frequency divider counter
105.
106.                                if (bcnt = 0) then -- at the
    end of each word
107.                                        bcnt<=28;      -- reset
    the bit counter
108.                                        wcnt<= wcnt-1; --modify
    the word counter
109.                                else    -- the bit is not the
    end of a word
110.                                        bcnt<= bcnt-1; --modify
    the bit counter
111.                                end if;
112.
113.                    else -- if not the end of the bit
114.                                f_div<= f_div-1; -- modify the
    frequency divider
115.                end if;
116.
117.         -- generating SCLK, it is going up and then down
    inside each bit
118.
119.                    if (f_div= 375) then -- condition when SCLK
    goes up
120.                                 SCLK <= '1';
121.                        elsif (f_div = 125) then       --
    condition when SCLK goes down
122.                                 SCLK <= '0';
123.                        end if;
```

```
124.
125.                 --generating---- serial--- data output
126.
127.                   -- start transition condition
128.                if (bcnt=28) then
129.                        if(f_div=499) then
130.                         SDIN<='1';
131.                        elsif(f_div=250) then
132.                                        SDIN<='0';
133.                        end if;
134.                        ----ack ---bit--- condition-----------
    --
135.                elsif (bcnt=19 or bcnt=10 or bcnt=1) then
136.                                       SDIN<='Z';
137.                        ---- stop ---transition ---condition--
    ---
138.                elsif (bcnt=0) then
139.                                if (wcnt/=0)then
140.                                if(f_div<=250)then
141.                                        SDIN<='1';
142.                                else
143.                                        SDIN<='0';
144.                                end if;
145.
146.                else
147.                        if(f_div=499) then
148.                          SDIN<='0';
149.                        elsif (f_div=250) then
150.                                SDIN<='1';
151.                   end if;
152.
153.     end if;
154.                        -- condition for the non-special bits
155.                        elsif(bcnt/=28 or bcnt/=19 or bcnt/=10
    or bcnt/=1 or bcnt/=0)and (f_div=499) then
156.
157.                                sht_r (11*24-1 downto 1)<=sht_r
    (11*24-2 downto 0); -- shifting
```

```
158.                          SDIN<= sht_r (11*24-1);
159.              end if;
160. end if;
161. bcnt1 <= bcnt;
162. wcnt1 <= wcnt;
163. fdiv1 <= f_div;
164.
165. end process;
166. end rtl;
```

s2p_adaptor

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. use ieee.numeric_std.all;
4.
5. entity s2p_adaptor is
6. port(
7. --        Core Side - two parallel interfaces for input and
   output
8.   ADCDAT:         out     std_logic_vector(15 downto 0);
9.   DACDAT:         in      std_logic_vector(15 downto 0);
10.          DACrdy:         out     std_logic;
11.          ADCrdy:         in      std_logic;
12.          DACstb:         in      std_logic;
13.          ADCstb:         out     std_logic;
14.  --      Audio Side in MASTER mode
15.          AUD_DACDAT:     out     std_logic; -- serial data out
16.          AUD_ADCDAT:     in      std_logic; -- serial data in
17.          AUD_ADCLRCK:    in      std_logic; -- strobe for input
18.          AUD_DACLRCK:    in      std_logic; -- strobe for output
19.          AUD_BCLK:       in      std_logic; -- serial interface
   "clock"
20.  --      Control Signals
21.          CLOCK_50:       in      std_logic               ;
22.          RST_N:          in      std_logic
23.  );
24.  end entity;
```

```
25.
26.  architecture rtl of s2p_adaptor is
27.  --      Internal Signals
28.  --input s2p
29.  signal counter1: integer range 15 downto -1 ;
30.  signal counter2: integer range 15 downto -1 ;
31.  signal old_BCLK :std_logic;
32.  signal old_AUD_DACLRCK:std_logic;
33.  signal shr:std_logic_vector(15 downto 0);
34.  signal ADCstb2:std_logic;
35.  signal DACrdy2:std_logic;
36.  begin
37.
38.          process (CLOCK_50)
39.          variable bit_ADC: integer;
40.          begin
41.          if (rising_edge(CLOCK_50)) then
42.                  -------------begin sync design----------------
43.
44.                  -- reset actions (synchronous)
45.                  if (RST_N = '0') then
46.                                  old_BCLK<='0';
47.                                  counter1<=15;
48.                                  counter2<=15;
49.                                  ADCstb2<='0';
50.                          else
51.                                  old_BCLK <= AUD_BCLK; -- needed
   for change detection on BCLK input
52.
53.                  old_AUD_DACLRCK <=AUD_DACLRCK;
54.
55.
56.
57.                  -- input channel
58.                    if (old_BCLK='0' and AUD_BCLK='1') then --
   rising edge of AUD_BCLK
59.
```

```
60.                                 if (AUD_ADCLRCK='1') then --
   condition for the start of the protocol
61.                                                 counter1<=14; -
   - load the bit counter
62.
     ADCDAT(15)<=AUD_ADCDAT; -- read the first bit of the packet
63.
64.                                 elsif (counter1>=0) then --
   condition for the data bits of the left channel
65.
     ADCDAT(counter1)<=AUD_ADCDAT;  -- input one bit
66.                                             counter1<=counter1-
   1;        -- advance the bit counter
67.                             if (counter1=0) then   -- condition
   for the strobe of ADC parallel interface
68.                                 ADCstb2<='1';
69.                                 end if;
70.                             end if;
71.
72.                     end if;
73.
74.                     if (ADCstb2='1') then-- condition to drop
   the ADC strobe
75.                             ADCstb2<='0';
76.                     end if;
77.
78.
79.
80.
81.
82.          if (old_AUD_DACLRCK = '0' and AUD_DACLRCK = '1' )
   then
83.          counter2<=14;
84.      AUD_DACDAT <= shr(15);
85.
86.          elsif (old_BCLK='1' and AUD_BCLK='0' and counter2 >=
   0) then
87.          counter2<=counter2-1;
```

```
88.          AUD_DACDAT <= shr(counter2);
89.
90.          elsif (old_BCLK='1' and AUD_BCLK='0' and counter2 <0)
   then
91.          AUD_DACDAT <= '0';
92.      shr <= DACDAT;
93.          end if;
94.          if (DACstb = '1' and counter2 = 0) then
95.    shr <= DACDAT;
96.          end if;
97.
98.
99.                  -- output channel
100.                          if(AUD_DACLRCK<='1') then -- start
   condition
101.
102.           counter2<=14;

103.                         AUD_DACDAT <=shr(15);-- the MSB will
   be o/p first
104.
105.                                 elsif (old_BCLK='1' and
   AUD_BCLK='0') then -- each following falling edge
106.
     AUD_DACDAT<=shr(counter2); -- produce DAC serial data bit
107.                                         counter2<=counter2-1;
108.                                 end if;
109.
110.                                 if (counter2=0) then --
   condition for loading DAC parallel register
111.                                         DACrdy2<='1';-- ready
   to receive the data from FIR filter
112.                                 end if;
113.                 if(DACrdy2='1') then
114.                         DACrdy2<='0';
115.                                 end if;
116.
117.                         end if;
```

```
118.
119.                 -------------end sync design----------------
120.         end if;
121. end process;
122.
123.         ADCstb<=ADCstb2;
124.         DACrdy<=DACrdy2;
125.
126. end rtl;
127.
```

FIR filter

```
1. library ieee;
2. use ieee.std_logic_1164.all;
3. use ieee.numeric_std.all;
4.
5. entity FIR_filter is
6.
7.   generic
8.   (
9.           NUM_STAGES : natural := 8      --Number of taps
10.          );
11.
12.          port(
13.  --      Core Side - two parallel interfaces for input and
   output
14.          ADCDAT: in      std_logic_vector(15 downto 0); --Q1.15
15.          DACDAT: out     std_logic_vector(15 downto 0); --Q3.13
16.          DACrdy:         in      std_logic;
17.          ADCrdy:         out     std_logic;
18.          DACstb:         out     std_logic;
19.          ADCstb:         in      std_logic;
20.
21.  --      Control Signals
22.          CLOCK_50:       in      std_logic;
23.          RST_N:          in      std_logic
24.          );
```

```
25.  end entity;
26.
27.
28.  architecture rtl of FIR_filter is
29.
30.  -- Build a 2-D array type for the shift register
31.          subtype sr_width is signed(15 downto 0);
32.          type sr_length is array ((NUM_STAGES-1) downto 0) of
   sr_width;
33.          -- Declare the input shift register signal
34.          signal SR_in: sr_length;
35.
36.  -- Build a 2-D array type for the shift register
37.          subtype SR_out_width is signed(34 downto 0);
38.          type SR_out_length is array ((NUM_STAGES-1) downto 0)
   of SR_out_width;
39.          -- Declare the output shift register signal
40.          signal SR_out: SR_out_length;
41.
42.  --FIR coefficient
43.          subtype Coeff_width is integer;
44.          type Coeff_length is array((NUM_STAGES-1) downto 0)
   of Coeff_width;
45.          constant Coeff : Coeff_length := (-1260, 7827, 12471,
   16384, 16384, 12471, 7827, -1260);
46.
47.  --      Internal Signals
48.          signal old_ADCstb : std_logic;
49.
50.          begin
51.                  process(CLOCK_50)
52.
53.                  variable n,i: integer;
54.                  variable signed_Coeff : signed(15 downto 0);
55.                  variable Y : signed(34 downto 0);      --Q5.30
56.                  variable one_exist : std_logic := '0';--for
   checking the exist of '1'
```

```
57.                 constant round_up:
   signed :="0000000000000001";
58.
59.                 begin
60.                         if (rising_edge(CLOCK_50)) then
61.
62.                                 -- reset actions
63.             if (RST_N = '0') then
64.                                         ADCrdy <= '0';
65.                                         DACstb <= '0';
66.                                         n := 0;
67.                                         SR_in <=
   ("0000000000000000",
68.
   "0000000000000000",
69.
   "0000000000000000",
70.
   "0000000000000000",
71.
   "0000000000000000",
72.
   "0000000000000000",
73.
   "0000000000000000",
74.
   "0000000000000000");
75.
76.                                 else
77.                                         old_ADCstb <= ADCstb;
78.

79.                                         if (old_ADCstb ='0' and
   ADCstb='1') then --rising edge of ADCstb
80.
     SR_in((NUM_STAGES-1) downto 1) <= SR_in((NUM_STAGES-2) downto
   0);       -- Shift data by one stage; data from last stage is
   lost
```

```
81.                                                 SR_in(0) <=
   signed(ADCDAT);  -- Load new data into the first stage
82.
83.                                                 ADCrdy <= '1';
84.                                                 n :=
   NUM_STAGES;
85.
86.                                             else
87.
88.                                                 if (n > 0) then
89.
     SR_out((NUM_STAGES-1) downto 1) <= SR_out((NUM_STAGES-2)
   downto 0);
90.
     signed_Coeff := to_signed(Coeff(8-n),16);
91.
     SR_out(0) <= resize(SR_in(n-1)* signed_Coeff,35);
92.                                                             --Y := Y
   + SR_out(0);
93.                                                             n := n-
   1;        --decrease the counter
94.
95.                                                 else
96.                                                             Y :=
   SR_out(7) + SR_out(6) + SR_out(5) + SR_out(4) + SR_out(3) +
   SR_out(2) +SR_out(1) + SR_out(0);
97.
98.                                                             for i in
   0 to 15 loop
99.
     one_exist := one_exist or Y(i);
100.                                                            end
   loop;
101.
102.                                                            if
   (Y(34) = '0' and Y(17) = '1') then
```

```
103.
    DACDAT <= std_logic_vector(Y(34) & Y(32 downto 18)+round_up);
104.
    DACstb <= '1';
105.                                                    elsif
   (Y(34) = '1' and Y(16) = '1' and one_exist = '1') then
106.
    DACDAT <= std_logic_vector(Y(34) & Y(32 downto 18)+round_up);

107.
    DACstb <= '1';
108.
    else
109.
    DACDAT <= std_logic_vector(Y(34) & Y(32 downto 18));
110.
    DACstb <= '1';
111.                                                    end if;
112.

113.                                                end if;
114.                                                ADCrdy <= '0';
115.

116.                                        end if;
117.
118.                                        if (DACrdy='1') then
    -- condition to drop the DAC strobe
119.                                                DACstb <= '0';
120.                                        end if;
121.
122.                                end if;
123.
124.                        end if;
125.                ------------end sync design---------------
126.        end process;
127.
128. end rtl;
```

Testbench

Codec_init

```
1. LIBRARY ieee;
2. USE ieee.std_logic_1164.all;
3.
4. ENTITY codec_init_vhd_tst IS
5. END codec_init_vhd_tst;
6. ARCHITECTURE codec_init_arch OF codec_init_vhd_tst IS
7. -- constants
8. -- signals
9. SIGNAL CLOCK_50 : STD_LOGIC;
10.  SIGNAL RES_N : STD_LOGIC;
11.  SIGNAL SCLK : STD_LOGIC;
12.  SIGNAL SDIN : STD_LOGIC;
13.  SIGNAL bcnt1 : integer;    --bcnt output
14.  SIGNAL wcnt1 :  integer;    --wcnt output
15.  SIGNAL fdiv1 :  integer;              --fdiv output
16.  COMPONENT codec_init
17.          PORT (
18.          CLOCK_50 : IN STD_LOGIC;
19.          RES_N : IN STD_LOGIC;
20.          SCLK : OUT STD_LOGIC;
21.          SDIN : OUT STD_LOGIC;
22.          bcnt1    : out integer;    --bcnt output
23.          wcnt1    : out integer;    --wcnt output
24.          fdiv1    : out integer          --fdiv output
25.          );
26.  END COMPONENT;
27.  BEGIN
28.          i1 : codec_init
29.          PORT MAP (
30.  -- list connections between master ports and signals
31.          CLOCK_50 => CLOCK_50,
32.          RES_N => RES_N,
33.          SCLK => SCLK,
34.          SDIN => SDIN,
```

```
35.          bcnt1 => bcnt1,
36.          wcnt1 => wcnt1,
37.          fdiv1 => fdiv1
38.          );
39.  clock : PROCESS
40.  -- variable declarations
41.  variable i : integer;-- variable declarations
42.  BEGIN
43.     for i in 1 to 500000 loop -- specify here the length of the
   simulation run
44.       CLOCK_50 <= '0';
45.       wait for 10 ns;
46.       CLOCK_50 <= '1';
47.       wait for 10 ns;
48.       end loop;     -- code that executes only once
49.  WAIT;
50.  END PROCESS clock;        -- code that executes only once
51.
52.  reset : PROCESS
53.
54.  BEGIN
55.        RES_N <= '0';
56.                  wait for 2700 ns;
57.                  RES_N <= '1';  -- code executes for every
   event on sensitivity list
58.  WAIT;
59.  END PROCESS reset;
60.
61.  END codec_init_arch;
```

s2p_adaptor

```
1. LIBRARY ieee;
2. USE ieee.std_logic_1164.all;
3. use ieee.numeric_std.all;
4. use ieee.math_real.all;
```

```
5.
6. ENTITY s2p_adaptor_vhd_tst IS
7. END s2p_adaptor_vhd_tst;
8. ARCHITECTURE s2p_adaptor_arch OF s2p_adaptor_vhd_tst IS
9. -- constants
10.  -- signals
11.  SIGNAL ADCDAT : STD_LOGIC_VECTOR(15 DOWNTO 0);
12.  SIGNAL ADCrdy : STD_LOGIC;
13.  SIGNAL ADCstb : STD_LOGIC;
14.  SIGNAL AUD_ADCDAT : STD_LOGIC;
15.  SIGNAL AUD_ADCLRCK : STD_LOGIC;
16.  SIGNAL AUD_BCLK : STD_LOGIC;
17.  SIGNAL AUD_DACDAT : STD_LOGIC;
18.  SIGNAL AUD_DACLRCK : STD_LOGIC;
19.  SIGNAL CLOCK_50 : STD_LOGIC;
20.  SIGNAL DACDAT : STD_LOGIC_VECTOR(15 DOWNTO 0);
21.  SIGNAL DACrdy : STD_LOGIC;
22.  SIGNAL DACstb : STD_LOGIC;
23.  SIGNAL RST_N : STD_LOGIC;
24.  COMPONENT s2p_adaptor
25.          PORT (
26.          ADCDAT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
27.          ADCrdy : IN STD_LOGIC;
28.          ADCstb : OUT STD_LOGIC;
29.          AUD_ADCDAT : IN STD_LOGIC;
30.          AUD_ADCLRCK : IN STD_LOGIC;
31.          AUD_BCLK : IN STD_LOGIC;
32.          AUD_DACDAT : OUT STD_LOGIC;
33.          AUD_DACLRCK : IN STD_LOGIC;
34.          CLOCK_50 : IN STD_LOGIC;
35.          DACDAT : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
36.          DACrdy : OUT STD_LOGIC;
37.          DACstb : IN STD_LOGIC;
38.          RST_N : IN STD_LOGIC
39.          );
40.  END COMPONENT;
41.  BEGIN
42.          i1 : s2p_adaptor
```

```
43.          PORT MAP (
44.  -- list connections between master ports and signals
45.          ADCDAT => ADCDAT,
46.          ADCrdy => ADCrdy,
47.          ADCstb => ADCstb,
48.          AUD_ADCDAT => AUD_ADCDAT,
49.          AUD_ADCLRCK => AUD_ADCLRCK,
50.          AUD_BCLK => AUD_BCLK,
51.          AUD_DACDAT => AUD_DACDAT,
52.          AUD_DACLRCK => AUD_DACLRCK,
53.          CLOCK_50 => CLOCK_50,
54.          DACDAT => DACDAT,
55.          DACrdy => DACrdy,
56.          DACstb => DACstb,
57.          RST_N => RST_N
58.          );
59.
60.  clock : PROCESS
61.  -- variable declarations
62.  variable i : integer;-- variable declarations
63.  BEGIN
64.     for i in 1 to 5000000 loop -- specify here the length of
   the simulation run
65.       CLOCK_50 <= '0';
66.       wait for 10 ns;
67.       CLOCK_50 <= '1';
68.       wait for 10 ns;
69.       end loop;     -- code that executes only once
70.  WAIT;
71.  END PROCESS clock;         -- code that executes only once
72.
73.  rst :process
74.  begin
75.          RST_N <= '1';
76.  --       ADCrdy <= '1';
77.  --       DACrdy <= '1';
78.  wait;
79.  end process rst;
```

```
80.
81.  BCLK :process
82.  variable i1 :integer;
83.  begin
84.          AUD_BCLK <='0';
85.          wait for 11.34 us;
86.          for i1 in 1 to 2000 loop
87.          AUD_BCLK <='1';
88.          wait for 11.34 us;
89.          AUD_BCLK <='0';
90.          wait for 11.34 us;
91.          end loop;
92.  wait;
93.  end process BCLK;
94.
95.  ADCLRCK :process
96.  variable i2 :integer;
97.  begin
98.          AUD_ADCLRCK <= '0';
99.          AUD_DACLRCK <= '0';
100.         wait for 22.68 us;
101.         for i2 in 1 to 200 loop
102.         AUD_ADCLRCK <= '1';
103.         AUD_DACLRCK <= '1';
104.         wait for 22.68 us;
105.         AUD_ADCLRCK <= '0';
106.         AUD_DACLRCK <= '0';
107.         wait for 385.56 us;
108.         end loop;
109. wait;
110. end process ADCLRCK;
111.
112. --AudioIn :process
113. --variable i3 :integer;
114. --begin
115. --       for i3 in 1 to 2000 loop
116. --       AUD_ADCDAT <='1';
117. --       wait for 22.68 us;
```

```
118. --      AUD_ADCDAT <='0';
119. --      wait for 22.68 us;
120. --      end loop;
121. --wait;
122. --end process AudioIn;
123.
124. signal_samples_proc:
125. PROCESS
126.        VARIABLE v_sin : real; -- genereated sine value
127.        VARIABLE i : integer; -- sample number
128.        CONSTANT Ts : real := 1.0/44100; -- sampling period
129.        CONSTANT f : real := 1000.0; -- frequency of the sine
    wave
130.        CONSTANT A : integer := 32000; -- amplitude
131.        CONSTANT Ns : integer := 200; -- number of samples to
    simulate
132. BEGIN
133.        FOR i IN 0 TO Ns LOOP
134.              v_sin := sin(2 * math_2_pi * f * Ts * i);
135.              AUD_ADCDAT := to_signed(integer(v_sin), 16);
136.              WAIT FOR Ts*64.0;
137.        END LOOP;
138.        WAIT;
139. END PROCESS;
140.
141. DACDAT <= ADCDAT;
142. DACstb <= ADCstb;
143.
144. END s2p_adaptor_arch;
```

FIR filter

```
1. LIBRARY ieee;
2. USE ieee.std_logic_1164.all;
3. use ieee.numeric_std.all;
4. use ieee.math_real.all;
```

```
5.
6.
7.
8. ENTITY fir_filter_vhd_tst IS
9. END fir_filter_vhd_tst;
10.  ARCHITECTURE fir_filter_arch OF fir_filter_vhd_tst IS
11.
12.  SIGNAL ADCDAT : STD_LOGIC_VECTOR(15 DOWNTO 0);
13.  SIGNAL ADCrdy : STD_LOGIC;
14.  SIGNAL ADCstb : STD_LOGIC;
15.  SIGNAL CLOCK_50 : STD_LOGIC;
16.  SIGNAL DACDAT : STD_LOGIC_VECTOR(15 DOWNTO 0);
17.  SIGNAL DACrdy : STD_LOGIC;
18.  SIGNAL DACstb : STD_LOGIC;
19.  SIGNAL RST_N : STD_LOGIC;
20.  SIGNAL Vs : signed (15 downto 0);
21.
22.
23.  COMPONENT fir_filter
24.          PORT (
25.          ADCDAT : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
26.          ADCrdy : OUT STD_LOGIC;
27.          ADCstb : IN STD_LOGIC;
28.          CLOCK_50 : IN STD_LOGIC;
29.          DACDAT : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
30.          DACrdy : IN STD_LOGIC;
31.          DACstb : OUT STD_LOGIC;
32.          RST_N : IN STD_LOGIC
33.          );
34.  END COMPONENT;
35.  BEGIN
36.          i1 : fir_filter
37.          PORT MAP (
38.  -- list connections between master ports and signals
39.          ADCDAT => ADCDAT,
40.          ADCrdy => ADCrdy,
41.          ADCstb => ADCstb,
42.          CLOCK_50 => CLOCK_50,
```

```
43.          DACDAT => DACDAT,
44.          DACrdy => DACrdy,
45.          DACstb => DACstb,
46.          RST_N => RST_N
47.          );
48.
49.
50.  clock : PROCESS
51.  -- variable declarations
52.  variable i : integer;-- variable declarations
53.  BEGIN
54.     for i in 1 to 500000000 loop -- specify here the length of
   the simulation run
55.       CLOCK_50 <= '0';
56.       wait for 10 ns;
57.       CLOCK_50 <= '1';
58.       wait for 10 ns;
59.       end loop;     -- code that executes only once
60.  WAIT;
61.  END PROCESS clock;         -- code that executes only once
62.
63.  reset : PROCESS
64.
65.  BEGIN
66.                 RST_N <= '1';  -- code executes for every
   event on sensitivity list
67.
68.                 DACrdy <= '1';
69.
70.
71.  WAIT;
72.  END PROCESS reset;
73.
74.  DAstrobe : process
75.  variable ia :integer;
76.  begin
77.                 for ia in 1 to 5000 loop
78.                 ADCstb <= '1';
```

```
79.                 wait for 10 ns;
80.                 ADCstb <= '0';
81.                 wait for 385.56 us;
82.                 end loop;
83.  wait;
84.  end process DAstrobe;
85.
86.  signal_samples_proc:
87.  PROCESS
88.         VARIABLE v_sin : real; -- genereated sine value
89.         VARIABLE ib : integer; -- sample number
90.         CONSTANT Ts : real := 1.0/44100; -- sampling period
91.         CONSTANT f : real := 2000.0; -- frequency of the sine
    wave
92.         CONSTANT A : integer := 32000; -- amplitude
93.         CONSTANT Ns : integer := 2000; -- number of samples to
    simulate
94.
95.
96.  BEGIN
97.         FOR ib IN 0 TO Ns LOOP
98.
99.                 v_sin := real(A) * sin(real(ib) * Ts * f *
    real(2) * math_2_pi);
100.                Vs <= to_signed(integer(v_sin), 16);
101.                                ADCDAT <= std_logic_vector(Vs);
102.                WAIT FOR Ts * 64.0 sec ;
103.        END LOOP;
104.        WAIT;
105. END PROCESS;
106.
107. END fir_filter_arch;
```