Bert Zhao
Shangrong Li
EE371 Lab 5 Report

Abstract

This lab report is based on the material of the lab 5 “Digital Signal Processing”. We will be discussing the background, procedure, analysis, result of this lab following by a brief conclusion through this report. This lab is also completed by a team of two students.

Introduction

The objective of the this lab is experiment with the CODEC in the DE1_SoC board and use the data coming out of the CODEC to perform signal processing. Noise is usually the main reason that reduces the quality of the recorded sound. We can still hear noises when we make phone calls nowadays. In order to clear the noises, we will build filters that are intended to clear noises that are caused by a variety of reasons.

Procedure

After physically connecting the ports for the microphone and the speaker, the first task in this lab is to pass the input signals from the microphone to the speaker. By reading the lab spec, we found that read and write only happens when the read_ready or write_ready signal is given respectively. Since we want to pass whatever picked up by the microphone to the speaker, we need to pass the read_data to write_data. In task 2, we need to create a filter with eight DFFs. We created a module that takes read_data as input and outputs the filtered data to the write_data to the top level module. Based on the block diagram from the spec, we have eight DFFs in series and pass the input value along the line of DFFs at each clock cycle. The input was also divided by eight since there are eight DFFs before it was passed into the first DFF. The outputs of each DFFs then summed up to be the output of the module. In order to account for negative values for the input data, we also had to declare the input as signed value in order to keep the sign bit properly when we perform division. On the other hand, we were also given the code for a noise generator. We can use this to add noise to the output data we can listen to the noisy version of the output. Then we can filter the noisy version and better perceive the effectiveness of the filter. In task 3, we were supposed to use a FIFO to create a filter in task 2 while having the option to increase the number of DFFs used in the filter. Knowing there is a FIFO file in the starter_kits file, we can add this file to our project without building another FIFO. Illustrated by the block diagram from the lab spec, we would still pass the input data into the FIFO after dividing it by the number of words in the FIFO. Then we would sum up the divided data with the accumulator and subtract the output from the FIFO. The accumulator is made with a value that keeps track of the output every clock cycle. Since we are only allowed to read from a FIFO when it’s full, we were supposed to read the FIFO when words_used from the FIFO equals to the depth of the FIFO. However, our FIFO happened to only eject the oldest value one clock cycle after the read signal was sent. We would have to adjust this by having the reading happen when words_used is equal to the depth of the FIFO - 1. This strategy was able to keep us away from the case of writing to a full FIFO.

Analysis & Result

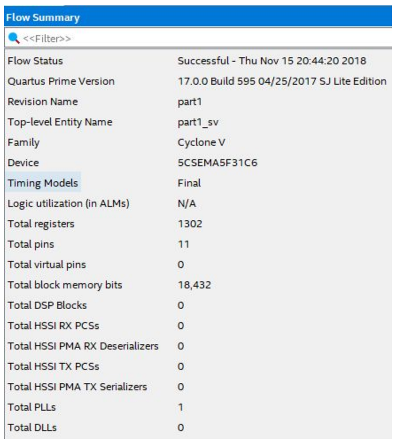
The first task of the lab did not include a filter, and therefore sounded the loudest and included the most noise. This was especially apparent with the extra noise module. The eight sample averaging filter from task two improved the quality slightly from part one. However, it was not enough to prevent feedback when the volume of the speaker was turned up too high. The n-sample averaging filter of task three produced the highest quality of sound playback at a lower volume than the others. The volume is decreased as more the number of samples is increased, which is the expected behavior of filters without an amplifier. In testing our filters, we found that playing music through the microphone gave the most consistent comparisons between the quality of the playback sound. Both task 2 and task 3 were tested using ModelSim. Additionally, data passed through the FIFO in task 3 was probed using signal tap. Screenshots of all related results are included in the appendix.

Conclusion

The primary objective of this lab was to introduce and work with sound inputs and outputs on the DE1-SoC. We worked worked with filters which is an introductory level task in regards to sound processing. More specifically, we worked with averaging filters to reduce the noise in our audio signals. We can now use the skills we learned in this lab and apply them to our final project. This lab took approximately 15 hours to complete.

Appendix

Flow Summary:

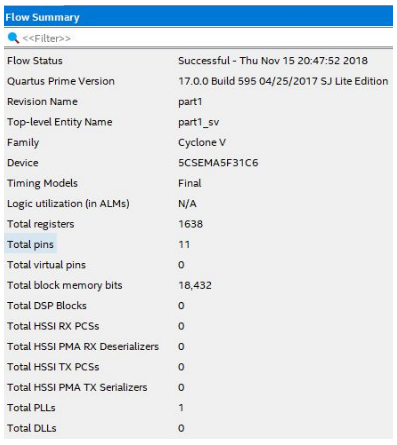


Flow Summary

<<Filter>>

Flow Status	Successful - Thu Nov 15 20:44:20 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	part1
Top-level Entity Name	part1_sv
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	1302
Total pins	11
Total virtual pins	0
Total block memory bits	18,432
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1
Total DLLs	0

Figure 1: Task 1 flow summary

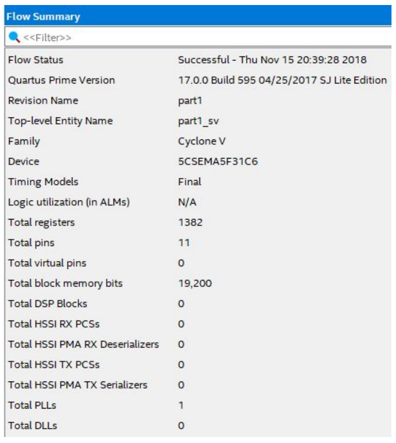


Flow Summary

<<Filter>>

Flow Status	Successful - Thu Nov 15 20:47:52 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	part1
Top-level Entity Name	part1_sv
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	1638
Total pins	11
Total virtual pins	0
Total block memory bits	18,432
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1
Total DLLs	0

Figure 2: Task 2 flow summary



Flow Summary

<<Filter>>

Flow Status	Successful - Thu Nov 15 20:39:28 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	part1
Top-level Entity Name	part1_sv
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	1382
Total pins	11
Total virtual pins	0
Total block memory bits	19,200
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1
Total DLLs	0

Figure 3: Task 3 flow summary

Simulations:

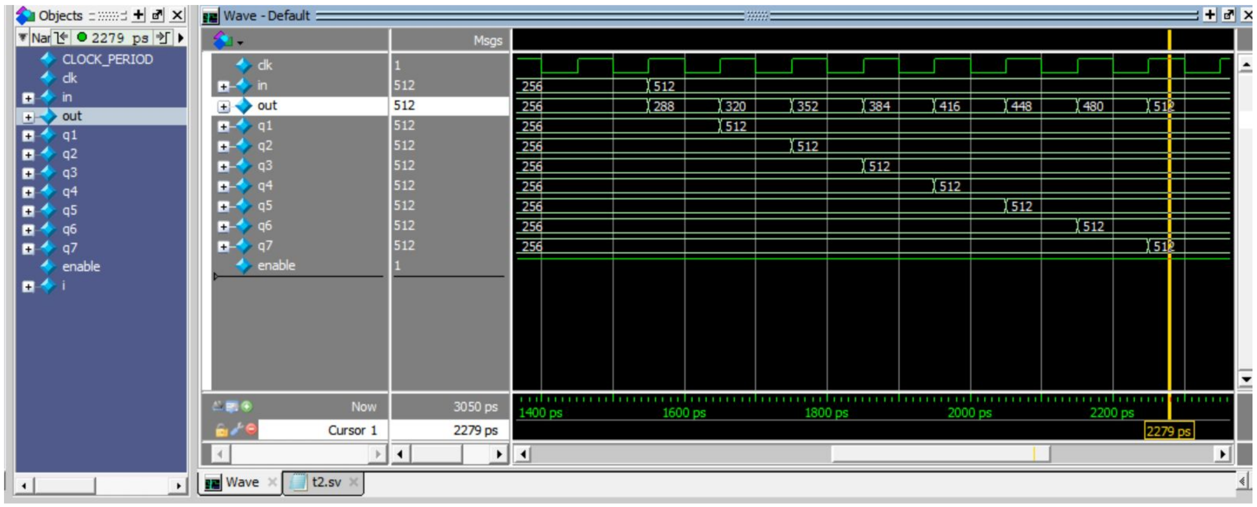


Figure 4: Task 2 ModelSim

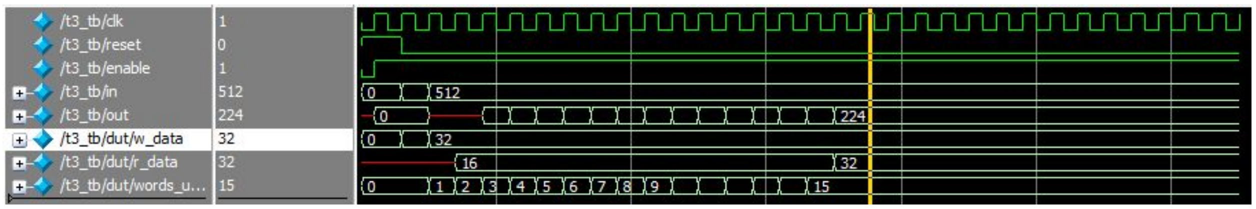
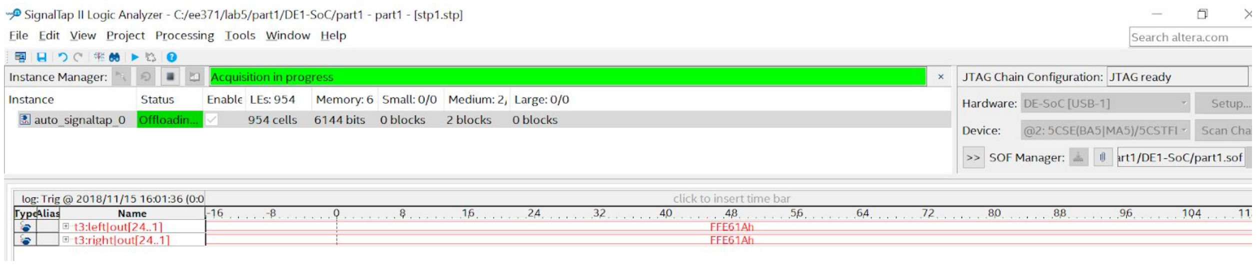


Figure 5: Task 3 ModelSim

SignalTap:


SignalTap II Logic Analyzer - C:/ee371/lab5/part1/DE1-SoC/part1 - part1 - [stp1.stp]

Instance Manager: Acquisition in progress

JTAG Chain Configuration: JTAG ready

Hardware: DE-SoC [USB-1]

Device: @2: 5CSE(BA5|MA5)/5CSTFI

SOF Manager: rt1/DE1-SoC/part1.sof

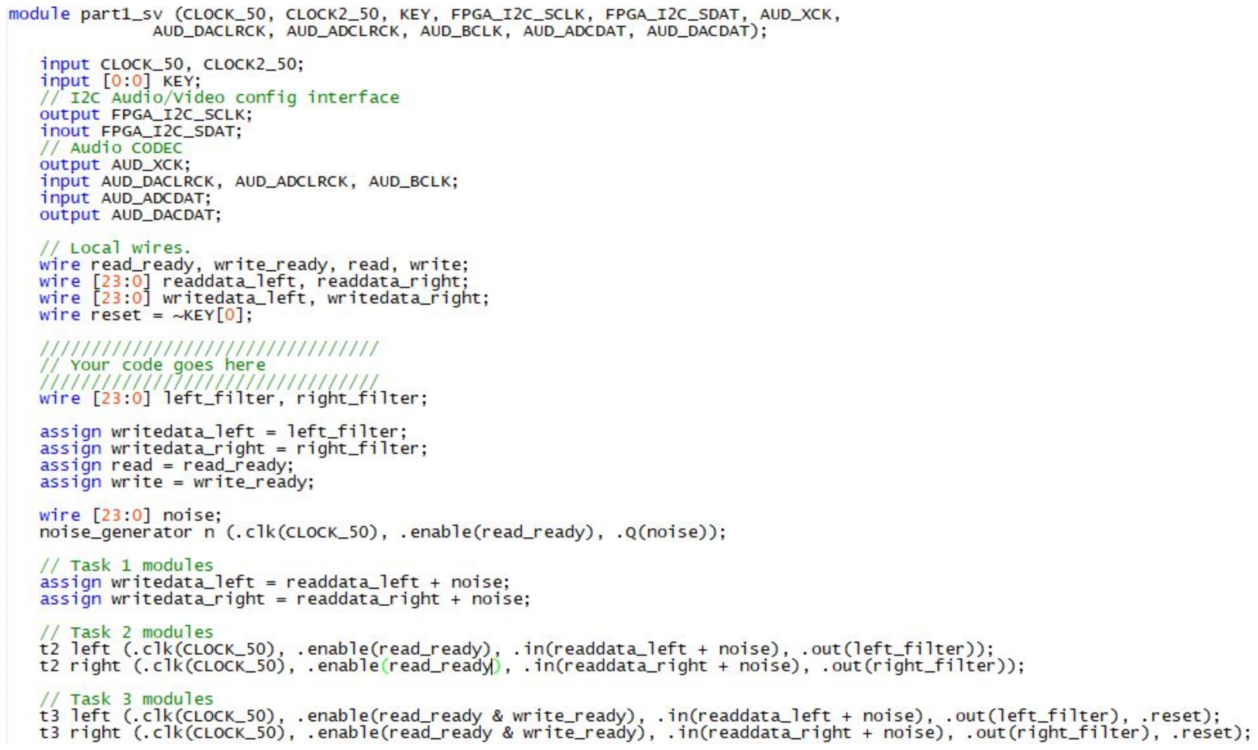
Instance	Status	Enable	LEs: 954	Memory: 6	Small: 0/0	Medium: 2,	Large: 0/0
auto_signaltap_0	Offloadin...	✓	954 cells	6144 bits	0 blocks	2 blocks	0 blocks

log: Trig @ 2018/11/15 16:01:36 (0:0

Type	Alias	Name	Value
		t3:left|out[24..1]	FFE61Ah
		t3:right|out[24..1]	FFE61Ah

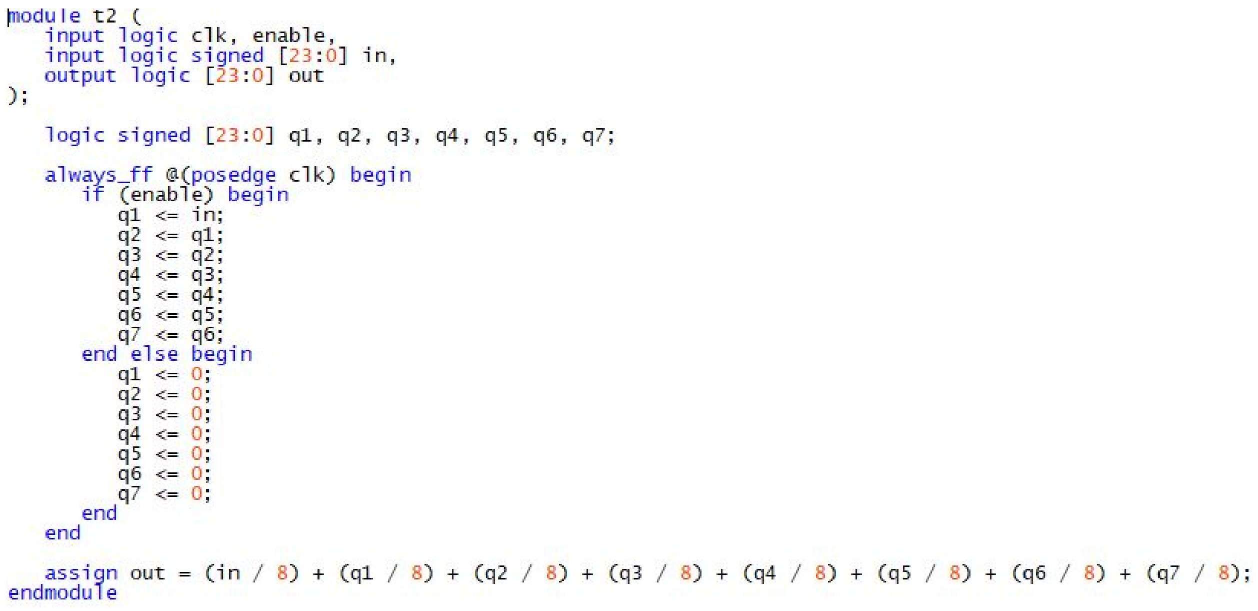
Figure 6: Task 3 ModelSim

Code:



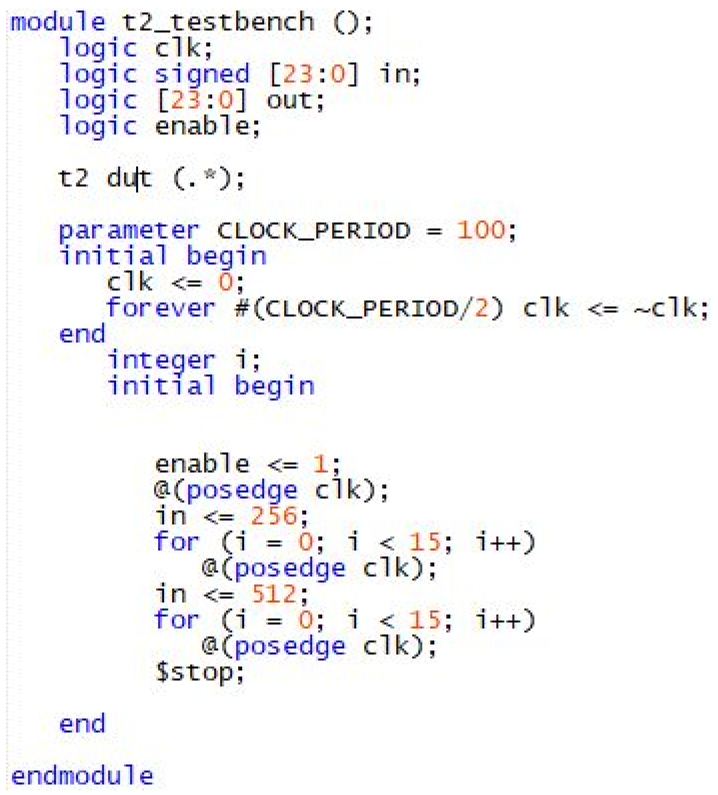
```
module part1_sv (CLOCK_50, CLOCK2_50, KEY, FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK,
                 AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);

   input CLOCK_50, CLOCK2_50;
   input [0:0] KEY;
   // I2C Audio/Video config interface
   output FPGA_I2C_SCLK;
   inout FPGA_I2C_SDAT;
   // Audio CODEC
   output AUD_XCK;
   input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
   input AUD_ADCDAT;
   output AUD_DACDAT;

   // Local wires.
   wire read_ready, write_ready, read, write;
   wire [23:0] readdata_left, readdata_right;
   wire [23:0] writedata_left, writedata_right;
   wire reset = ~KEY[0];

   /////////////////////////////////
   // Your code goes here
   /////////////////////////////////
   wire [23:0] left_filter, right_filter;

   assign writedata_left = left_filter;
   assign writedata_right = right_filter;
   assign read = read_ready;
   assign write = write_ready;

   wire [23:0] noise;
   noise_generator n (.clk(CLOCK_50), .enable(read_ready), .Q(noise));

   // Task 1 modules
   assign writedata_left = readdata_left + noise;
   assign writedata_right = readdata_right + noise;

   // Task 2 modules
   t2 left (.clk(CLOCK_50), .enable(read_ready), .in(readdata_left + noise), .out(left_filter));
   t2 right (.clk(CLOCK_50), .enable(read_ready), .in(readdata_right + noise), .out(right_filter));

   // Task 3 modules
   t3 left (.clk(CLOCK_50), .enable(read_ready & write_ready), .in(readdata_left + noise), .out(left_filter), .reset);
   t3 right (.clk(CLOCK_50), .enable(read_ready & write_ready), .in(readdata_right + noise), .out(right_filter), .reset);
```

Code 1: top level part1.sv



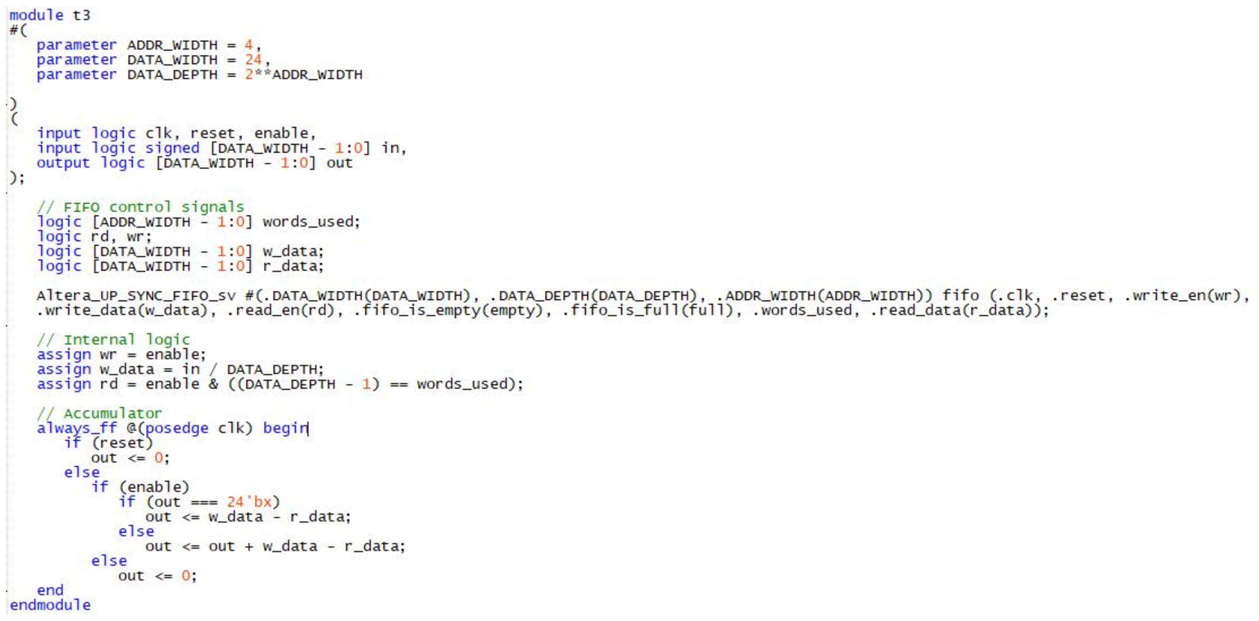
```
module t2 (
   input logic clk, enable,
   input logic signed [23:0] in,
   output logic [23:0] out
);

   logic signed [23:0] q1, q2, q3, q4, q5, q6, q7;

   always_ff @(posedge clk) begin
      if (enable) begin
         q1 <= in;
         q2 <= q1;
         q3 <= q2;
         q4 <= q3;
         q5 <= q4;
         q6 <= q5;
         q7 <= q6;
      end else begin
         q1 <= 0;
         q2 <= 0;
         q3 <= 0;
         q4 <= 0;
         q5 <= 0;
         q6 <= 0;
         q7 <= 0;
      end
   end

   assign out = (in / 8) + (q1 / 8) + (q2 / 8) + (q3 / 8) + (q4 / 8) + (q5 / 8) + (q6 / 8) + (q7 / 8);
endmodule
```

Code 2: Task 2 t2.sv



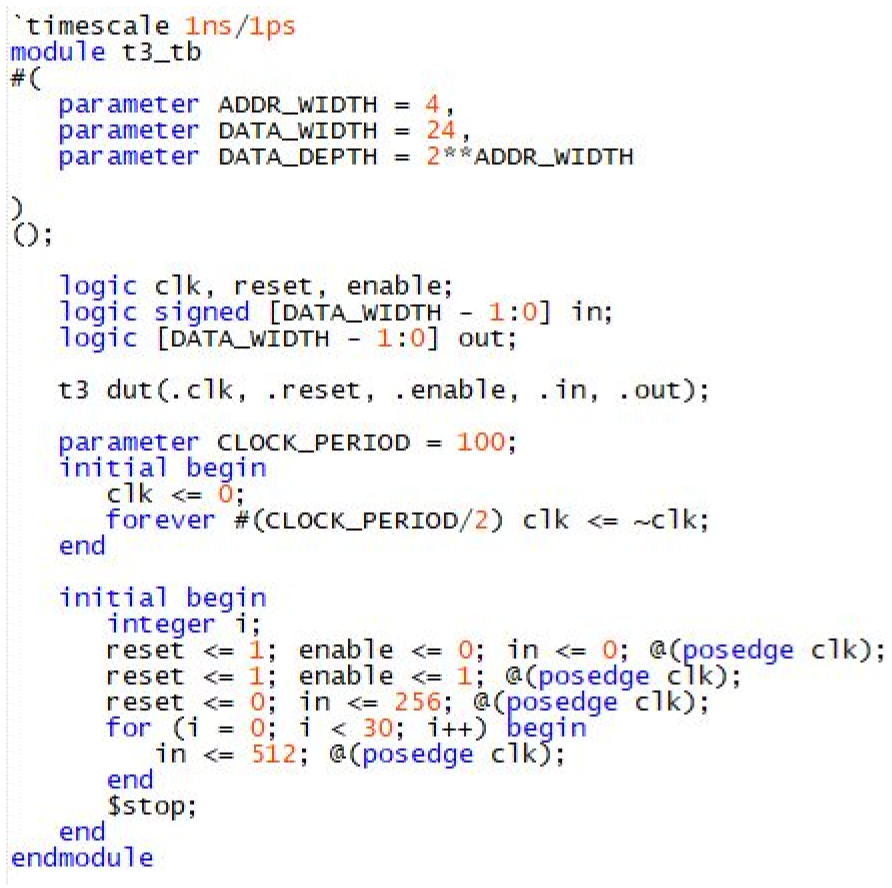
```
module t2_testbench ();
   logic clk;
   logic signed [23:0] in;
   logic [23:0] out;
   logic enable;

   t2 dut (.*);

   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end
      integer i;
      initial begin


         enable <= 1;
         @(posedge clk);
         in <= 256;
         for (i = 0; i < 15; i++)
            @(posedge clk);
         in <= 512;
         for (i = 0; i < 15; i++)
            @(posedge clk);
         $stop;

   end

endmodule
```

Code 3: Task 2 t2_tb.sv



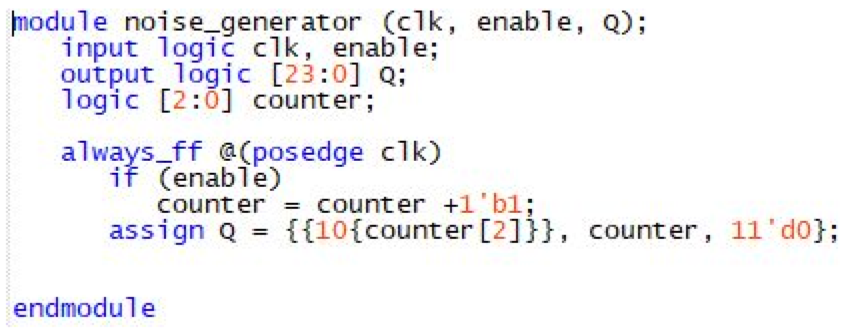
```
module t3
#(
   parameter ADDR_WIDTH = 4,
   parameter DATA_WIDTH = 24,
   parameter DATA_DEPTH = 2**ADDR_WIDTH

)
(
   input logic clk, reset, enable,
   input logic signed [DATA_WIDTH - 1:0] in,
   output logic [DATA_WIDTH - 1:0] out
);

   // FIFO control signals
   logic [ADDR_WIDTH - 1:0] words_used;
   logic rd, wr;
   logic [DATA_WIDTH - 1:0] w_data;
   logic [DATA_WIDTH - 1:0] r_data;

   Altera_UP_SYNC_FIFO_sv #(.DATA_WIDTH(DATA_WIDTH), .DATA_DEPTH(DATA_DEPTH), .ADDR_WIDTH(ADDR_WIDTH)) fifo (.clk, .reset, .write_en(wr),
   .write_data(w_data), .read_en(rd), .fifo_is_empty(empty), .fifo_is_full(full), .words_used, .read_data(r_data));

   // Internal logic
   assign wr = enable;
   assign w_data = in / DATA_DEPTH;
   assign rd = enable & ((DATA_DEPTH - 1) == words_used);

   // Accumulator
   always_ff @(posedge clk) begin
      if (reset)
         out <= 0;
      else
         if (enable)
            if (out === 24'bx)
               out <= w_data - r_data;
            else
               out <= out + w_data - r_data;
         else
            out <= 0;
   end
endmodule
```

Code 4: Task 3 t3.sv



```
`timescale 1ns/1ps
module t3_tb
#(
   parameter ADDR_WIDTH = 4,
   parameter DATA_WIDTH = 24,
   parameter DATA_DEPTH = 2**ADDR_WIDTH

)
();

   logic clk, reset, enable;
   logic signed [DATA_WIDTH - 1:0] in;
   logic [DATA_WIDTH - 1:0] out;

   t3 dut(.clk, .reset, .enable, .in, .out);

   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   initial begin
      integer i;
      reset <= 1; enable <= 0; in <= 0; @(posedge clk);
      reset <= 1; enable <= 1; @(posedge clk);
      reset <= 0; in <= 256; @(posedge clk);
      for (i = 0; i < 30; i++) begin
         in <= 512; @(posedge clk);
      end
      $stop;
   end
endmodule
```

Code 5: Task 3 t3_tb.sv



```
module noise_generator (clk, enable, Q);
   input logic clk, enable;
   output logic [23:0] Q;
   logic [2:0] counter;

   always_ff @(posedge clk)
      if (enable)
         counter = counter +1'b1;
      assign Q = {{10{counter[2]}}, counter, 11'd0};


endmodule
```

Code 6: noise_generator.sv