Mechtron 3TB4: Embedded Systems Design II
Assignment 2 Sample Solution

Due: In class on Wednesday February 28, 2018.

Q.1 [3+4+3=10]

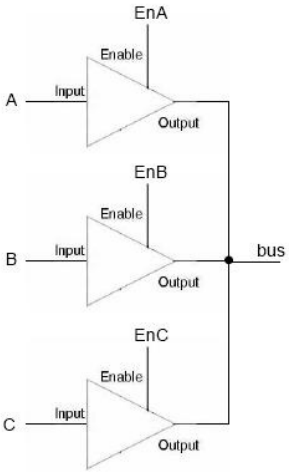
a) Consider the following code written in Verilog HDL:

```
module test(a, result);
output [15:0] a;  // 16-bit output
output [31:0] result; // 32-bit output

assign a= {4{4'b0101}};
assign result = {{16{a[15]}}, {a[15:0]}};
endmodule
```

What are the values assigned to: result[25] and result[2]?
Ans:
result[25] = 0, result[2] = 1

b) The schematic below shows three devices attached to a bus through tri-state buffers. Given the table with input signals, specify the state that you expect to see on the bus. You can enter one of the following values: 0, 1, Z (high impedance), or SC (for short circuit).
ANS: The signals on BUS from top to bottom in the table: SC, 1, Z, 1



A	EnA	B	EnB	C	EnC	bus
1	1	0	1	1	0	
1	1	1	1	0	0	
1	0	1	0	0	0	
1	1	0	0	1	0	

Figure 1:

c) An analog signal of frequency 7kHz is sampled with a sampling rate of 4kHz. What aliasing frequency will result in the sampled data because of this sampling scheme? Show your work.

Ans:
$f_s = 4kHz, f = 7kHz$
$n = 4/7 = 1(nearest\ integer)$
$f_a = |f_s * n - f| = 3kHz$

Q.2 [4+6]

1. Given a digital filter's output in the figure 2, and sampling frequency of 40kHz, calculate (approximately) the filter's output $y(t)$ for the following input signal [4]

$$x(t) = 0.5 * sin(2 * \pi * 2000 * t) + 0.3 * cos(2 * \pi * 3000 * t)$$

ANS:

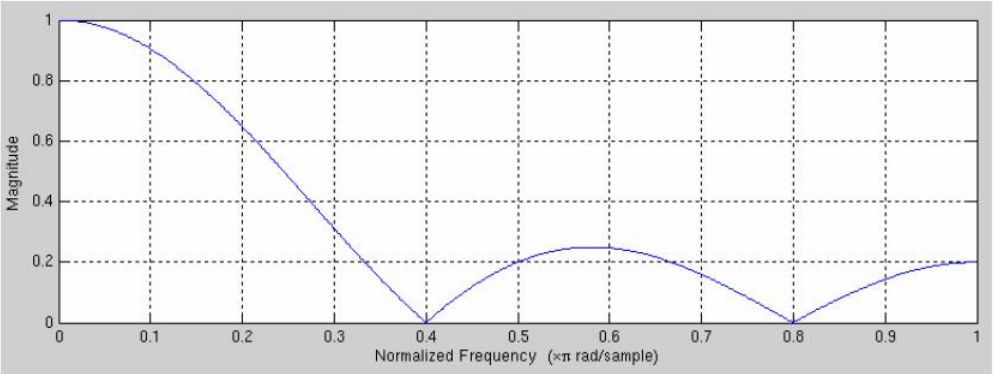
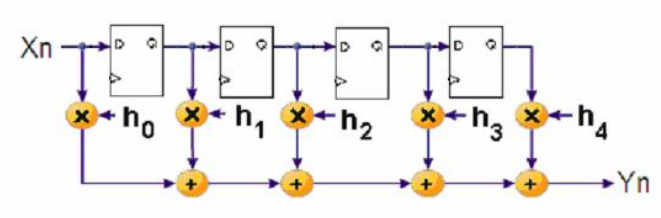


Figure 2:

$$y(t) = 0.5 * 0.9 * sin(2 * \pi * 2000 * t) + 0.3 * 0.8 * cos(2 * \pi * 3000 * t)$$

2. You are given the FIR filter shown in figure 3, and set of coefficients $h0 = 1, h1 = 0, h2 = 0.25, h3 = 0.5$, and $h4 = 0.75$.
   If a sequence of input signals Xn, given in the table that is part of figure 3 is used, calculate the response of the filter Yn. [6]
   Hint: Assume that all registers initially contain zero. Also Xn is fed from left to right i. e. 10 first then 20 and so on.

ANS: Yn from left to right: 10, 20, 12.5, 10, 50, 60



Xn	10	20	10	0	30	40
Yn						

Figure 3:

Q3 [10] Write verilog code to implement a simple ALU (arithmatic logic unit) that takes two 8-bit operands a and b and three control signals: add_sub, set_low and set_high. The output is an 8-bit data called result. The operation is described below:
If set_low is asserted, the ALU replaces the 4-least significant bits of operand a with 4-least significant bits of operand b and outputs the modified value.
If set_high is asserted, the ALU replaces the 4-most significant bits of operand a with 4-least significant bits of operand b and outputs the modified value.
If both of the above controls are not asserted then the output depends on the control signal add_sub. If it is high, b is subtracted from a and if it is low, a and b are added.

ANS:

```
module alu (input add_sub, set_low, set_high, input [7:0] operanda,
                    operandb, output reg [7:0] result);

always @(*)
begin
     if (set_low)
          result={operanda[7:4], operandb[3:0]};
     else if (set_high)
          result={operandb[3:0], operanda[3:0]};
     else
             begin
                   if(add_sub) // if 1 then subtraction,   0---addition
                          result=operanda-operandb;
                   else
                          result=operanda+operandb;
             end
```

```
end

endmodule
```

Q4 [10] As you know, a program counter (PC) is a register in a computer system that keeps track of the next instruction to be executed. Write Verilog code to implement a PC that has an 8-bit output and the following inputs:

clock: PC is a register - a sequential circuit, so it needs a clock
reset_n: a control signal which is asserted when low, to reset the PC to a known value. Make this value a PARAMETER, so that it may be changed for different applications.
branch: if this signal is high, the PC is set to an 8_bit value passed as input (name it newpc).
increment: the current value of PC is increased by 1, if this signal is high.

```
module pc (input clk, reset_n, branch, increment, input [7:0] newpc,
output reg [7:0] pc);
parameter RESET_LOCATION = 8'h00;

always @(posedge clk)
begin
 if (!reset_n)
   pc <= RESET_LOCATION;
 else if (branch)
   pc <= newpc;
 else if (increment)
   pc = pc + 8'b00000001;
end

endmodule
```

Q5 [10] A processor has several general purpose registers. All these registers can be implemented as one module called a register file. Implement a simple register file in Verilog that has three registers and the following parameters:
Input:
clock: a register needs it
reset_n: resets all registers to 0, when this signal is low.
write: a new value is written to a selected register, only if this signal is high.
wr_select: a 2-bit input to select a register for writing a new value in it.
data: an 8-bit input that can be written to a selected register, if above conditions are met.

select0: a 2-bit input to select a register for reading its contents and making it available on an 8-bit output, called selected0.
Output:
An 8-bit value - selected0.

```
// This module implements the register file

module regfile (input clk, reset_n, write, input [7:0] data,
                     input [1:0] select0, wr_select,
                     output reg [7:0] selected0);
reg [7:0] reg0 ;
reg [7:0] reg1 ;
reg [7:0] reg2 ;

   always @(posedge clk)
         begin
                    if (!reset_n)
                      begin
                         reg0<=8'h00;
                        reg1<=8'h00;
                        reg2<=8'h00;
                     end
                     else if (write)
                          begin
                             case (wr_select)
                                  2'b00: reg0<=data;
                                  2'b01: reg1<=data;
                                  2'b10: reg2<=data;
                             endcase
                         end
          end

   always @(posedge clk)
          begin
             case (select0)
                2'b00: selected0<=reg0;
                2'b01: selected0<=reg1;
                2'b10: selected0<=reg2;
            endcase
          end

endmodule
```

Q6 [10] A delay counter is a circuit that takes a specified delay value in 1/100s of

a second (an 8-bit binary number) and outputs a logic high signal when the delay is done. Its implementation requires an upcounter that divides the input clock by a suitable value so that a downcounter holding the delay value is decremented every millisecond. If the downcounter reaches zero, the output done=1 otherwise it remains low. Write a Verilog module to implement such a circuit with following parameters: Input:
clock - it has an input clock of frequency= 1MHz.
reset_n: resets all counters to 0 when this signal is low.
start: a high signal that causes the specified delay to be loaded into a downcounter and resets an upcounter to zero. enable: when high, causes the upcounter to start incrementing by one each clock cycle if its count is less than the value required to divide the input clock, however if that value is reached, the downcounter is decremented by one and the upcounter is reset to zero. Output: done = 1 when the downcounter = 0

```
module delay_counter (input clk, reset_n, start, enable,
      input [7:0] delay, output done );
parameter BASIC_PERIOD=14'd10000;
//14 bits, required to hold 100000 decimal

// for 1 MHz clock, the 1/100 second is 10,000 clock cycle, that is 14'd10000

reg [7:0] downcounter;
reg [19:0] timer;

always @( posedge clk)
begin
if (!reset_n)
begin
timer<=14'd0;//14'b00000000000000;
downcounter<=8'h00;
end
else if (start==1'b1)
begin
timer<=14'd0;
downcounter<=delay;
end
else if (enable==1'b1)
begin
if (timer<BASIC_PERIOD)
timer<=timer+14'd1;//14'b00000000000001;
else
begin
downcounter<=downcounter-8'b1;
timer<=14'd0;//14'b00000000000000;
```

```
end
end
end

assign done=(downcounter==8'b00000000)?1'b1:1'b0;
endmodule
```