EE5331-DSP Architectures and Embedded Systems

Programming Assignment 3

1 Introduction

In this assignment, we will implement a 4-bit Johnson counter and a timer on the FPGA board.

2 Implementing a Johnson Counter on an FPGA

In the first part of this assignment, we will implement a Johnson counter. A 4-bit Johnson counter is shown in Fig. 1. The Johnson counter is like a ring counter, however, the inverted output of the last flip-flop is connected to the input of the first flip-flop. A 3-bit Johnson counter will have states given by 000, 100, 110, 111, 011 and 001 and this sequence repeats.

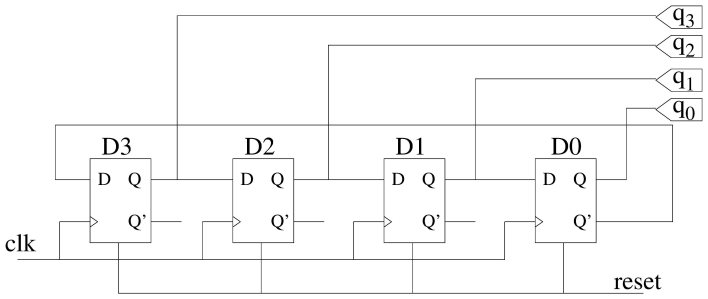

Figure 1: 4-bit Johnson counter

Task : Implement a 4-bit Johnson counter on the FPGA. You need to write a clock divider program and create an appropriate ucf file. The output for the Johnson counter should appear on LEDs on the FPGA board. Partial Verilog programs for some of the other parts are given below.

```
module johnson4bit(q3,q2,q1,q0,clk,rst);
input clk, rst;
output q0,q1,q2,q3;
wire q3bar, q2bar, q1bar, q0bar;
dflipflop_withreset D3(q3,q3bar,---,rst,clk);
dflipflop_withreset D2(q2,q2bar,---,rst,clk);
dflipflop_withreset D1(q1,q1bar,---,rst,clk);
dflipflop_withreset D0(q0,q0bar,q1,rst,clk);
endmodule
```

```
module dflipflop_withreset(q, qbar, d, rst, clk);
output q,qbar;
input d, rst, clk;
reg q,qbar;
always @ (posedge clk)
 begin
   if (~rst)
    begin
      q <= 1'b0;
      qbar <= 1'b1;
```

```
    end
   else
     begin
      q <= ----;  //  complete the description of  f/f
      qbar <= ----; // A Johnson counter transfers whatever is at
     end          // q to the data i/p of next f/f except for the leftmost
 end             // f/f whose i/p is qbar from the rightmost f/f
endmodule


- Write a clock divider module in Verilog
```

3 Timer Implementation on an FPGA

Embedded systems typically include a *timer* in some form (implemented using a microcontroller or other platforms). Here we will design a timer in Verilog and implement it on the FPGA. For this purpose, we will use the sliding switches and LEDs on the Xilinx Starter Kit as shown in Figure 2.

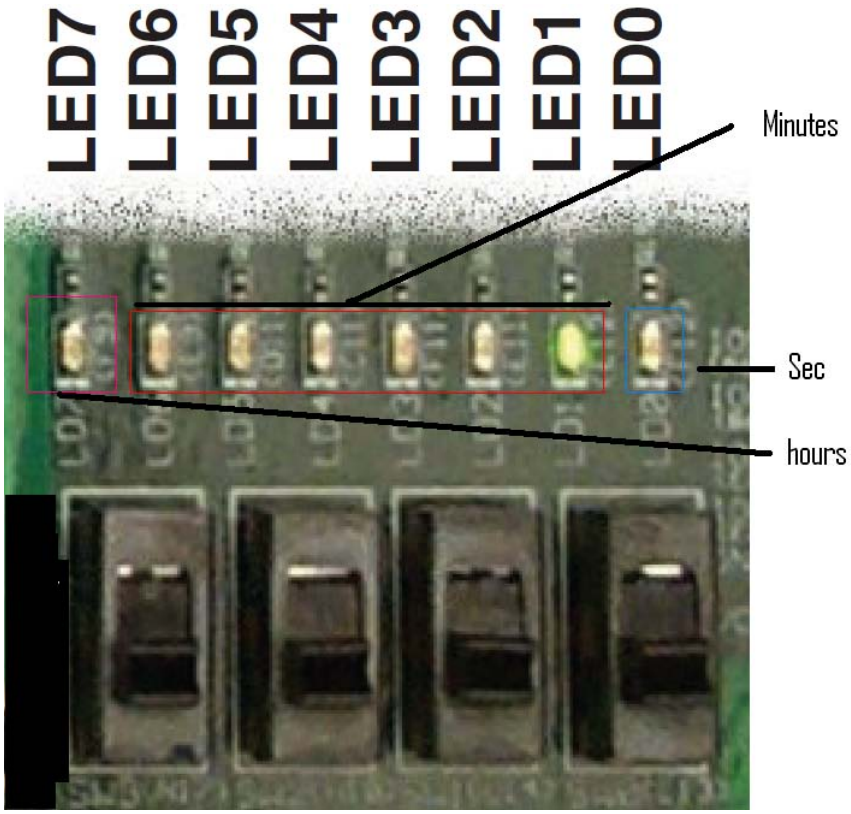

Figure 2: Starter Kit's LEDs to be used for display of seconds, minutes and hours

Task: Complete the following program to display a digital clock. When *start* is set to 0, initialization of the block (which includes *count*) is done. When *start* is 1, the *clock operation* must be performed (and displayed on the FPGA board). Write a *.ucf* file, take input from a sliding switch on the FPGA board and direct the output on (to) the LEDs.

```
module digital_clock_1Hz(start, clk,  sec, minutes, hours);

input start;
input clk;
```

```
output  sec;
output reg [5:0]minutes;
output reg hours;


reg [.....]count = .........; // fill in size of  count  to generate a 1-Hz clock
reg clk_1Hz = 1'b1, clk_60sec = 1'b1 ;
reg [5:0]sec_count;


assign sec = clk_1Hz;


// 1-second  clock generating block

always@(posedge clk)
begin

 if(start == 1'b0)
begin
count   <= ......;
   clk_1Hz <= !clk_1Hz ;
end
 else
begin
if(count == .......)   //  fill appropriate count to generate a  1-second clock

begin
  count <= ..... ;
clk_1Hz <= !clk_1Hz;
end
else
count <= count + 1'b1;
   end

end

//1-minute clock generating block

always@(posedge clk_1Hz)
begin
  if(start == 1'b0)
begin
sec_count <= 6'd0;
clk_60sec <= !clk_60sec;

end
  else
begin
if(sec_count == ....... )   // fill appropriate value for count to generate a 1-minute clock
begin
sec_count <= 6'd0;
clk_60sec <= ...... ;
         end
else
sec_count <= ........;
end

end
```

```
// minutes and hours counting block

always@(posedge clk_60sec)
begin
  if(start == 1'b0)
begin
minutes <= 6'd0;
 hours  <= 1'b0 ;
end
  else
begin
if(minutes == ......... ) // fill in to generate increment for "hour"
begin
minutes <= 6'd0 ;
hours   <= ........;
          end
else
minutes <= ........;
end

end


endmodule
```