Lab 05: Implementing digital clock on a Nexys 4 FPGA board

Objectives

1. Introduce students to the concept of counters in digital systems. Explain the fundamental purpose of counters in counting sequences of numbers and their applications in various electronic devices.
2. Teach students the theory and implementation of binary counters. Discuss the binary counting sequence and guide students in designing and simulating binary counters using Verilog. Emphasize the importance of understanding binary arithmetic in counter design.
3. Explain the concept of up/down counters and their ability to count in both ascending and descending sequences. Guide students in designing and implementing up/down counters in Verilog. Discuss the control signals required for switching between counting modes.

5.1. Introduction to Counters in Digital Systems:

Counters are essential components in digital systems used to count sequences of numbers. They play a fundamental role in various electronic devices and are crucial for tasks involving timing, control, and arithmetic operations. Counters are primarily used to generate specific sequences of binary numbers, allowing digital systems to perform tasks such as counting events, generating timing signals, controlling operations, and addressing memory locations. Understanding the concept of counters is fundamental to digital circuit design and is essential for students pursuing fields related to electrical engineering, computer science, and embedded systems.

5.2. Types of Counters

Counters are of two types:

1. Asynchronous counter.
2. Synchronous counter

5.2.1. Asynchronous counter

A digital counter is a set of flip flop. The flip flop are connected such that their combined state at any time is binary equivalent of total no. of pulses that have occurred up to that time. Thus its name implies a counter is used to count pulse. A counter is used as frequency dividers. To obtain waveform with frequency that is specific fraction of clock frequency.

5.2.2. Synchronous counter

When counter is clocked such that each flip flop in the counter is triggered at the same time, the counter is called as synchronous counter. The gates propagation delay at reset time will not be present or we may say will not occur.

5.2.3. Basic Counting Operations:

- Counters can count up, count down, or alternate between counting up and counting down. The direction of counting is controlled by specific signals, allowing flexibility in the counting process.

5.2.4. Event Counting:

Counters can count external events, such as button presses, sensor outputs, or any other input signals. This is common in applications where the number of occurrences of an event needs to be tracked.

5.2.5. Control Logic:

Counters are used in control circuits to sequence operations. For example, in a traffic light controller, counters can be used to time the duration of each light state.

5.2.6. Arithmetic Operations:

Counters can be used to perform basic arithmetic operations, such as addition and subtraction. This is achieved by appropriately configuring the counter and utilizing its output.

5.3. Example

• Design a 2-digit counter in Verilog that counts from 0 to 99 and displays the digits on two 7- segment displays. Additionally, implement a clock divider circuit that slows down the counting speed, and when the counter reaches 10, 11, or 12, the corresponding two-digit number should be displayed on the 7-segment displays. This design aims to create a visual representation of the count while ensuring accurate digit display and proper clock division.

5.3.1. Verilog Code

```
module zero_99(
  input clock,
    input reset,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
```

```
    output g,
    output dp,
    output [3:0]an
    );

reg [3:0]first; //register for the first digit
reg [3:0]second; //register for the second digit

reg [22:0] delay; //register to produce the 0.1 second delay
wire test;

always @ (posedge clock or posedge reset)
 begin
  if (reset)
   delay <= 0;
  else
   delay <= delay + 1;
 end

assign test = &delay; //AND each bit of delay with itself; test will be high only when all bits of delay are high


always @ (posedge test or posedge reset)
 begin
  if (reset) begin
   first <= 0;
   second <= 0;
  end
   else if (first==4'd9) begin  //x9 reached
    first <= 0;
```

```
    if (second == 4'd9) //99 reached
     second <= 0;
     else
      second <= second + 1;

  end
  else
   first <= first + 1;
 end

//Multiplexing circuit below

localparam N = 18;

reg [N-1:0]count;

always @ (posedge clock or posedge reset)
 begin
  if (reset)
   count <= 0;
  else
   count <= count + 1;
 end

reg [6:0]sseg;
reg [3:0]an_temp;
always @ (*)
 begin
  case(count[N-1:N-2])
```

```
   2'b00 :
    begin
     sseg = first;
     an_temp = 4'b1110;
    end

   2'b01:
    begin
     sseg = second;
     an_temp = 4'b1101;
    end

   2'b10:
    begin
     sseg = 6'ha; //unknown sent to produce '-'
     an_temp = 4'b1011;
    end

   2'b11:
    begin
     sseg = 6'ha; //unknown sent to produce '-'
     an_temp = 4'b0111;
    end
  endcase
 end
assign an = an_temp;

reg [6:0] sseg_temp;
always @ (*)
 begin
```

```
  case(sseg)
   4'd0 : sseg_temp = 7'b1000000; //0
   4'd1 : sseg_temp = 7'b1111001; //1
   4'd2 : sseg_temp = 7'b0100100; //2
   4'd3 : sseg_temp = 7'b0110000; //3
   4'd4 : sseg_temp = 7'b0011001; //4
   4'd5 : sseg_temp = 7'b0010010; //5
   4'd6 : sseg_temp = 7'b0000010; //6
   4'd7 : sseg_temp = 7'b1111000; //7
   4'd8 : sseg_temp = 7'b0000000; //8
   4'd9 : sseg_temp = 7'b0010000; //9
    default : sseg_temp = 7'b0111111; //dash
  endcase
 end
assign {g, f, e, d, c, b, a} = sseg_temp;
assign dp = 1'b1; //we dont need the decimal here so turn all of them off


endmodule
```

5.4. Task

1. Design a digital clock system using a common anode 7-segment display. The digital clock should accurately display hours, minutes, and seconds in a 24-hour format. The system should have the following functionalities: 1. Display the seconds, incrementing from 00 to 30. When the count reaches 30, reset the seconds to 00 and increment the minutes by 1. If the minutes reach 2, reset to 00 and increment the hours by 1.