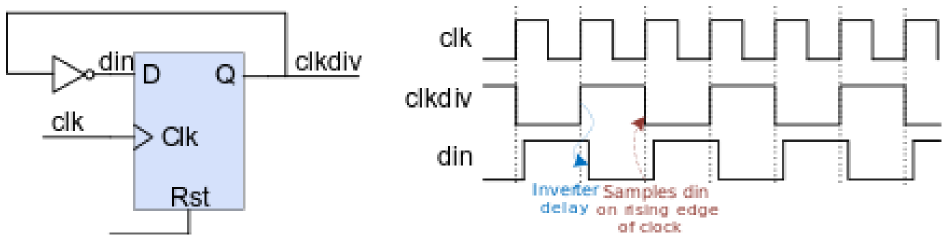
Experiment No. 8

Clocking of Artix-7 FPGA on Basys3 Kit

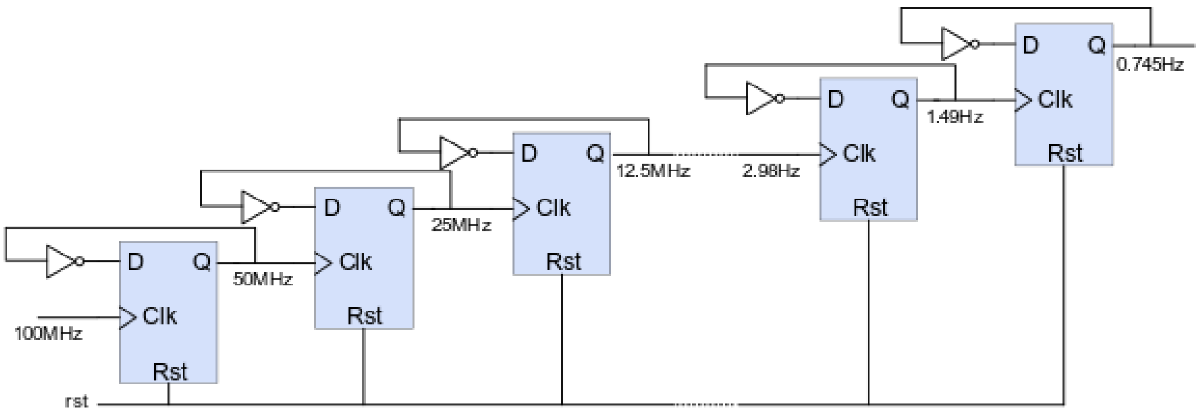
OBJECTIVE: To understand the clocking mechanism for Artix-7 FPGA on Basys3 kit by realizing various sequential circuits through Verilog.

THEORY: Sequential digital circuits need a clock signal for them to function. Usually, the clock signal comes from a crystal oscillator on-board. The Basys3 board includes a single 100MHz oscillator connected to pin W5. However, some peripheral controllers do not need such a high frequency to operate. Therefore, the frequency of the clock must be slowed down.

There is a simple circuit that can divide the clock frequency by half:



The clock can further be divided by cascading the above circuit:



Each stage divided the frequency by 2.

After the first stage, the frequency becomes: $\frac{100MHz}{2} = 50MHz.$

After the second stage, the frequency becomes: $\frac{100MHz}{2*2} = 25MHz.$

After the third stage, the frequency becomes: $\frac{100MHz}{2*2*2} = 12.5MHz.$

…….

Like this, after the n^{th} stage, the frequency becomes:

$$\frac{100MHz}{\underbrace{2 \cdot 2 \cdot \ldots \cdot 2}_{n}} = \frac{100,000,000}{2^n} Hz$$

For example, the output of a 3-stage clock divider circuit can be seen as a 3-bit up counter. At each bit, the output is a square wave of frequency $\frac{f_{clk}}{2^{(bit\ index)+1}}$, which can be used as a slowed-down clock:

	Stage-3 output: Square wave of frequency $\frac{f_{clk}}{2^{(2)+1}}$	**Stage-2 output:** Square wave of frequency $\frac{f_{clk}}{2^{(1)+1}}$	**Stage-1 output:** Square wave of frequency $\frac{f_{clk}}{2^{(0)+1}}$
f_{clk}	Q[2]	Q[1]	Q[0]
0→1	0	0	0
0→1	0	0	1
0→1	0	1	0
0→1	0	1	1
0→1	1	0	0
0→1	1	0	1
0→1	1	1	0
0→1	1	1	1
0→1	0	0	0

EXAMPLE 8.1:

Write a Verilog description to slow down the input clock frequency of 100MHz to blink the onboard LED of Basys3 kit at 0.745 Hz.

Solution:

$$\frac{Input\ clock\ frequency}{2^{(bit\ index)+1}} = desired\ frequency$$

$$\frac{100\ MHz}{2^{(bit\ index)+1}} = 0.745\ Hz$$

$$2^{(bit\ index)+1} = 134228187.9\ Hz$$

$$(bit\ index) + 1 = \frac{log\ 134228187.9}{log\ 2}$$

$$(bit\ index) + 1 = 27$$

$$bit\ index = 26$$

Therefore, design a 27-bit up counter and consider its MSB (i.e., 26th bit) output as the slowed-down clock to blink the onboard LED

Verilog Code:

```
   module Clk_Div(input clk, reset, output counter);
    reg [26:0] counter_up;

    // up counter
    always @(posedge clk or posedge reset)
    begin
        if(reset)
            counter_up <= 0;
        else
            counter_up <= counter_up + 1;
    end
    assign counter = counter_up[26];
endmodule
```

Simulation Results:

Input: clk = 10ns (100MHz), reset = 1→0

Output: counter = 1.34s (0.745Hz) square wave

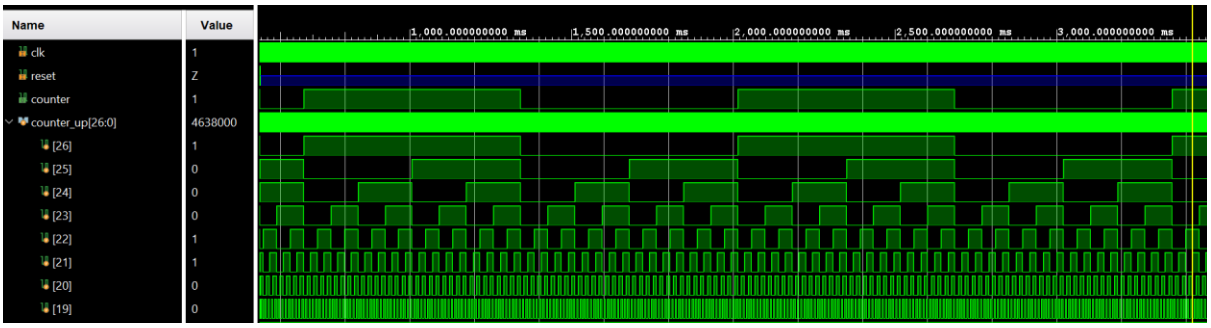


Fig. 8.1: Simulation results of example 8.1

IO pin assignment:

clk: W5	reset: T17	counter: U16

Hardware results:

LED “LD0” of BASYS3 blinks at the rate of 1.34 seconds.

EXAMPLE 8.2:

Write a Verilog description to blink the onboard LED of the Basys3 kit at the rate of 1 second.

Solution:

The relation between the input frequency and the desired frequency, given in example 8.1, doesn't always result in the "*bit index,*" an integer.

For example, for the desired frequency of 1Hz (or 1s):

$$\frac{100\ \mathrm{MHz}}{2^{(\mathrm{bit\ index})+1}} = 1\mathrm{Hz}$$

$$2^{(\mathrm{bit\ index})+1} = 100\ \mathrm{MHz}$$

$$(\mathrm{bit\ index}) + 1 = \frac{\log 100000000}{\log 2}$$

$$(\mathrm{bit\ index}) + 1 = 26.57$$

$$\mathrm{bit\ index} = 25.57$$

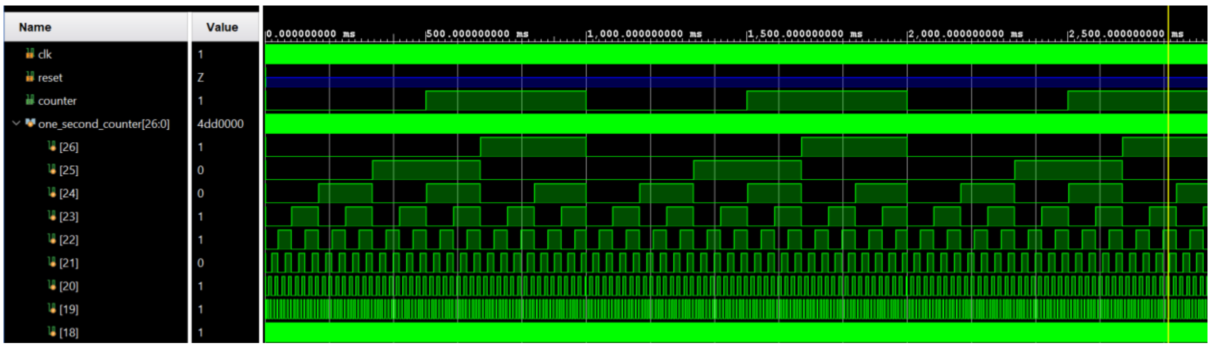
where the "*bit index*" must be adjusted to either 25 (which corresponds to 1.49Hz or 0.67s) or 26 (which corresponds to 0.745 Hz or 1.34s). Therefore, this example gives an alternative method of generating the desired clock of period 1 second.

Verilog Code:

```
module Clk_Div(input clk, reset, output counter);
  reg [26:0] one_second_counter;
  // up counter
  always @(posedge clk or posedge reset)
  begin
    if(reset==1)
        begin
        one_second_counter = 0;
        end
    else
        begin
         one_second_counter = one_second_counter + 1;
         if(one_second_counter==99999999)
           begin
           one_second_counter = 0;
           end
        end
   end
   assign counter = (one_second_counter>49999999)?1:0;
endmodule
```

Simulation Results:

Input: clk = 10ns (100MHz), reset = 1→0; **Output**: counter = 1s (1Hz) square wave



Name	Value
clk	1
reset	Z
counter	1
one_second_counter[26:0]	4dd0000
[26]	1
[25]	0
[24]	0
[23]	1
[22]	1
[21]	0
[20]	1
[19]	1
[18]	1

Fig. 8.2: Simulation results of example 8.2

IO pin assignment:

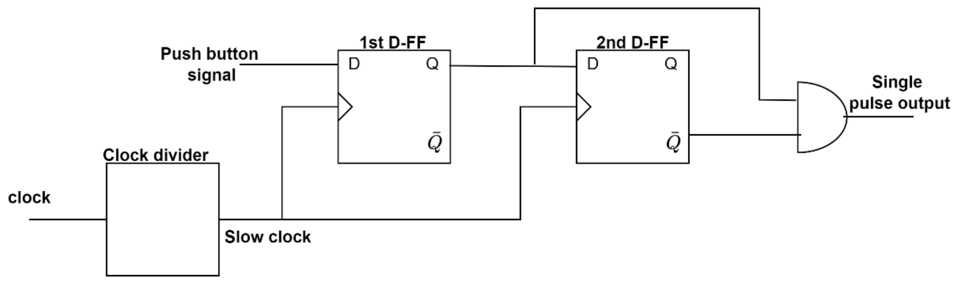
clk: W5	reset: T17	counter: U16

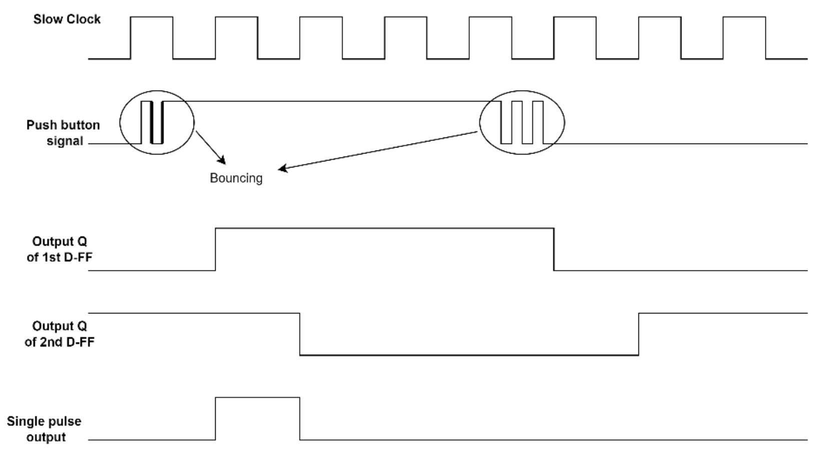
Hardware results:

LED "LD0" of BASYS3 blinks at the rate of 1 second.

EXAMPLE 8.3: Write a Verilog code to debounce the on-board push button of the Basys 3 FPGA kit.

Solution: Mechanical buttons cause an unpredictable bounce in the signal when toggled. Therefore, a simple debouncing circuit that can generate only a single pulse when the button on FPGA is pressed is:





Verilog Code:

```
module debounce(input pb_1,clk,
     output pb_out,slow_clk,Q1,Q2,Q2_bar);

     wire slow_clk;
     wire Q1,Q2,Q2_bar;
     clock_div u1(clk,slow_clk);

     my_dff d1(slow_clk, pb_1,Q1 );
     my_dff d2(slow_clk, Q1,Q2 );
     assign Q2_bar = ~Q2;
     assign pb_out = Q1 & Q2_bar;
endmodule
```

// Slow clock for debouncing

```
module clock_div(input Clk_100M, output reg slow_clk);
     reg [26:0]counter=0;

     always @(posedge Clk_100M)
      begin
         counter <= (counter>=249999)?0:counter+1;
                                                  //Uncomment it for simulation
         slow_clk <= (counter <125000)?1'b0:1'b1;
                                                  //Uncomment it for simulation
       //counter <= (counter>=99999999)?0:counter+1;
                                                  // Uncomment it for HW implementation
       //  slow_clk <= (counter <49999999)?1'b0:1'b1;
                                                  // Uncomment it for HW implementation
      end
endmodule
```

// D-flip-flop for debouncing module

```
module my_dff(input DFF_CLOCK, D, output reg Q);
    always @ (posedge DFF_CLOCK) begin
        Q <= D;
    end
endmodule
```

Test bench:

```
`timescale 1ns / 1ps
module tb_button;
     reg pb_1;
     reg clk;
```

```
    wire pb_out;
    // Instantiate the debouncing Verilog code
    debounce uut (
    .pb_1(pb_1),
    .clk(clk),
    .pb_out(pb_out),
    .slow_clk(slow_clk),
    .Q1(Q1),
    .Q2(Q2),
    .Q2_bar(Q2_bar)
    );

initial
    begin
    clk = 0;
    forever #5 clk = ~clk;
    end

initial
    begin
    pb_1 = 0;
    #3000000;  //3ms
    pb_1=1;
    #100000;  //0.1ms
    pb_1 = 0;
    #100000;  //0.1ms
    pb_1=1;
    #100000;  //0.1ms
    pb_1 = 0;
    #100000;   //0.1ms
    pb_1=1;
    #200000;  //0.2ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #200000;  //0.2ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #10000000;  //10ms
    pb_1 = 0;
    #100000;  //0.1ms
    pb_1=1;
    #200000;//0.2ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #100000;  //0.1ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #100000;//0.1ms
    pb_1 = 0;
    #20000000;  //20ms
    pb_1=1;
    #100000;  //0.1ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #100000;  //0.1ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #200000;//0.2ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #200000;  //0.2ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #10000000;  //10ms
    pb_1 = 0;
    #100000;  //0.1ms
    pb_1=1;
    #200000;//0.2ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #100000;  //0.1ms
    pb_1 = 0;
    #100000;//0.1ms
    pb_1=1;
    #100000;//0.1ms
    pb_1 = 0;
  end
endmodule
```

Simulation Results:

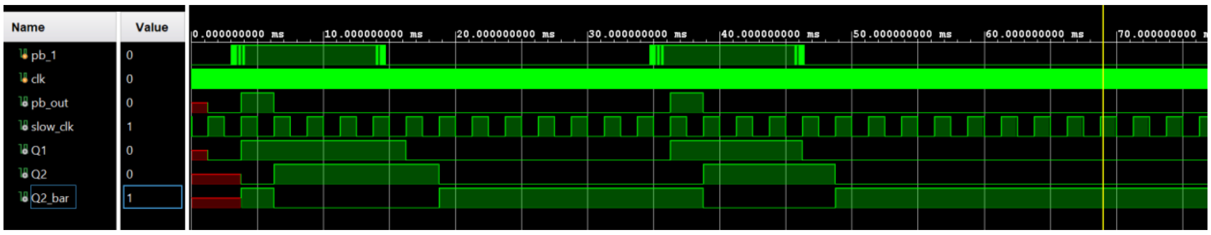


Fig. 8.3: Simulation results of example 8.3

IO Pin assignment:

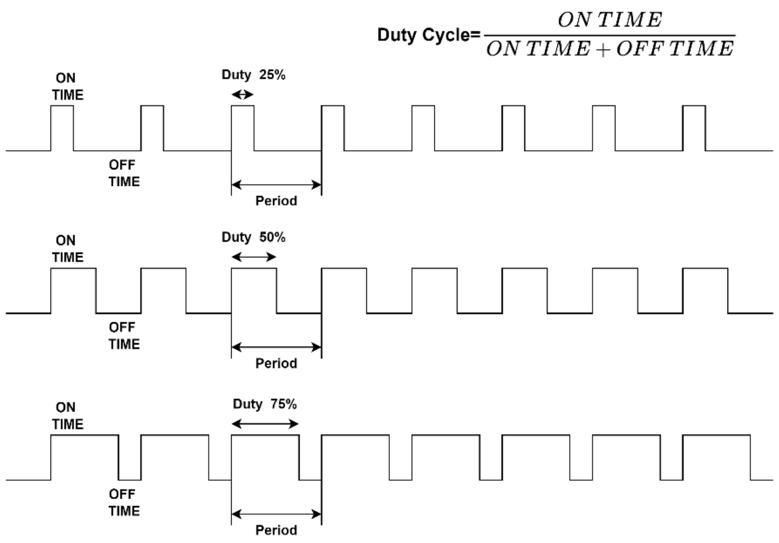
Input	**Pin No**	**Output**	**Pin No**
pb_1	T17	pb_out	U16
clk	W5	slow_clk	V14
		Q1	L1
		Q2	P1
		Q2_bar	N3

Hardware Results: Variation of output due to button BTNR press

BTNR	**Before press**	**Press and Hold**	**Release**
LD7	Blinks at rate of 1s	Blinks at rate of 1s	Blinks at a rate of 1s
LD15	OFF	ON	OFF at next active edge of slow_clk
LD14	OFF	ON at the next active edge of slow_clk	OFF at the second active edge of slow_clk
LD13	ON	OFF at the next active edge of slow_clk	ON at the second active edge of slow_clk
LD0	OFF	ON at the next active edge of slow_clk and then immediately OFF at the next active edge	OFF

EXAMPLE 8.4: Realize a pulse width modulated(PWM) wave of varying duty cycles on the onboard LED of basys-3 through the Verilog description

Solution:



Verilog Code:

```verilog
module PWM
 (
 input clk, // 100MHz clock input
 input increase_duty,  // input to increase 10% duty cycle
 input decrease_duty,  // input to decrease 10% duty cycle
 output duty_inc,
 output duty_dec,
 output PWM_OUT,  // 1Hz PWM output signal
 output PWM_clk,
 output DUTY_CYCLE,
 output reg [3:0] Anode_Activate,  // anode signals of  7-segment LED display
 output reg [6:0] LED_out    // cathode patterns of the 7-segment LED display
    );

 reg [26:0] one_second_counter=0;
 wire PWM_clk;
 wire tmp1,tmp2,duty_inc;// temporary flip-flop signals for debouncing the increasing button
 wire tmp3,tmp4,duty_dec;// temporary flip-flop signals for debouncing the decreasing button
 reg[3:0] counter_PWM=0;  // counter for creating PWM signal
 reg[3:0] DUTY_CYCLE=5;  // initial duty cycle is 50%

 // PWM clock
 always @(posedge clk)
  begin
        one_second_counter = one_second_counter + 1;
   //    if(one_second_counter==2)  //Comment it for FPGA implementation
        if(one_second_counter>99999999)  //Comment it for simulation
          begin
          one_second_counter = 0;
          end
   end
 //assign PWM_clk = (one_second_counter>0)?1:0;
                                             //Comment it for FPGA implementation
   assign PWM_clk = (one_second_counter>49999999)?1:0;
                                                 // Comment it for simulation

 // debouncing FFs for increasing button
 DFF_PWM PWM_DFF1(PWM_clk,increase_duty,tmp1);
 DFF_PWM PWM_DFF2(PWM_clk,tmp1, tmp2);
 assign duty_inc =  tmp1 & (~ tmp2);
 // debouncing FFs for decreasing button
 DFF_PWM PWM_DFF3(PWM_clk,decrease_duty, tmp3);
 DFF_PWM PWM_DFF4(PWM_clk,tmp3, tmp4);
 assign duty_dec =  tmp3 & (~ tmp4);

// vary the duty cycle using the debounced buttons above
 always @(posedge PWM_clk)
 begin
   if(duty_inc==1 && DUTY_CYCLE <= 9)
    DUTY_CYCLE <= DUTY_CYCLE + 1; // increase duty cycle by 10%
   else if(duty_dec==1 && DUTY_CYCLE>=1)
    DUTY_CYCLE <= DUTY_CYCLE - 1; //decrease duty cycle by 10%
 end

// Create PWM signal with variable duty cycle controlled by 2 buttons
 always @(posedge PWM_clk)
 begin
   counter_PWM <= counter_PWM + 1;
   if(counter_PWM>=9)
    counter_PWM <= 0;
 end
 assign PWM_OUT = counter_PWM < DUTY_CYCLE ? 1:0;

 always @(*)
    begin
        Anode_Activate = 4'b1110;
        case(DUTY_CYCLE)
        4'b0000: LED_out = 7'b0000001; // "0"
        4'b0001: LED_out = 7'b1001111; // "1"
        4'b0010: LED_out = 7'b0010010; // "2"
        4'b0011: LED_out = 7'b0000110; // "3"
        4'b0100: LED_out = 7'b1001100; // "4"
        4'b0101: LED_out = 7'b0100100; // "5"
        4'b0110: LED_out = 7'b0100000; // "6"
        4'b0111: LED_out = 7'b0001111; // "7"
        4'b1000: LED_out = 7'b0000000; // "8"
        4'b1001: LED_out = 7'b0000100; // "9"
        default: LED_out = 7'b0000001; // "0"
        endcase
    end
endmodule

// Debouncing DFFs for push buttons on FPGA
module DFF_PWM(clk,D,Q);
input clk,D;
output reg Q;
always @(posedge clk)
```

```
begin
  Q <= D;
end
endmodule
```

Test bench:

```
`timescale 1ns / 1ps
module tb_PWM_Generator_Verilog;
 // Inputs
 reg clk;
 reg increase_duty;
 reg decrease_duty;
 // Outputs
 wire PWM_OUT;
 wire PWM_clk;
 wire duty_inc;
 wire duty_dec;
 wire [3:0]Anode_Activate;
 wire [6:0]LED_out;
 wire [3:0]DUTY_CYCLE;

// Instantiate the PWM Generator with variable duty cycle in Verilog
 PWM PWM_Generator_Unit(
  .clk(clk),
  .increase_duty(increase_duty),
  .decrease_duty(decrease_duty),
  .PWM_OUT(PWM_OUT),
  .PWM_clk(PWM_clk),
  .duty_inc(duty_inc),
  .duty_dec(duty_dec),
  .Anode_Activate(Anode_Activate),
  .LED_out(LED_out),
  .DUTY_CYCLE(DUTY_CYCLE)
 );

// Create 100Mhz clock
 initial begin
 clk = 0;
 forever #5 clk = ~clk;
 end
 initial begin
  increase_duty = 0;
  decrease_duty = 0;
  #1000;
```

```
   increase_duty = 1;
  #100; // increase duty cycle by 10%
    increase_duty = 0;
  #100;
    increase_duty = 1;
  #100; // increase duty cycle by 10%
    increase_duty = 0;
  #100;
    increase_duty = 1;
  #100; // increase duty cycle by 10%
    increase_duty = 0;
  #100;
    decrease_duty = 1;
  #100; // decrease duty cycle by 10%
    decrease_duty = 0;
  #100;
    decrease_duty = 1;
  #100; // decrease duty cycle by 10%
    decrease_duty = 0;
  #100;
    decrease_duty = 1;
  #100; // decrease duty cycle by 10%
    decrease_duty = 0;
  #100;
    decrease_duty = 1;
  #100; // decrease duty cycle by 10%
    decrease_duty = 0;
  #100;
    decrease_duty = 1;
  #100; // decrease duty cycle by 10%
    decrease_duty = 0;
 end
endmodule
```

Simulation Results:

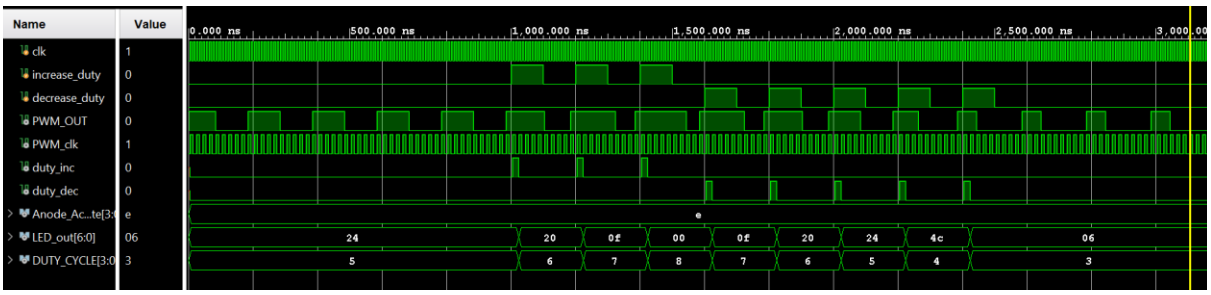


Fig. 8.4: Simulation results of example 8.4

IO Pin Assignment:

		Anode_Activate [3:0]		LED_out [6:0]			
Increase_duty	T18	Bit 3: Display 1	W4	Bit 6: Seg a	W7	Bit 2: Seg e	U5
Decrease_duty	U17	Bit 2: Display 2	V4	Bit 5: Seg b	W6	Bit 1: Seg f	V5
Clock	W5	Bit 1: Display 3	U4	Bit 4: Seg c	U8	Bit 0: Seg g	U7
Duty_inc	N3	Bit 0: Display 4	U2	Bit 3: Seg d	V8		
Duty_dec	P3						
PWM_clk	L1		**DYTY_CYCLE [3:0]**				
PWM_OUT	V3		Bit3: V19	Bit2: U19	Bit1: E19	Bit0: U16	

Hardware Results:

Initial state		**BTNU press**	**BTNU press**	**BTND press**	**BTND press**	**BTND press**	**BTND press**
LD5	Blinks at rate of 1s	Blinks at rate of 1s	Blinks at rate of 1s	Blinks at rate of 1s	Blinks at rate of 1s	Blinks at rate of 1s	Blinks at rate of 1s
SSD	5	6	7	6	5	4	3
LD9	PWM output of 50%duty cycle	PWM output of 60% duty cycle	PWM output of 70% duty cycle	PWM output of 60% duty cycle	PWM output of 50% duty cycle	PWM output of 40% duty cycle	PWM output of 30% duty cycle
LD13	OFF	ON for 1 PWM clock cycle	ON for 1 PWM clock cycle	OFF	OFF	OFF	OFF
LD12	OFF	OFF	OFF	ON for 1 PWM clock cycle	ON for 1 PWM clock cycle	ON for 1 PWM clock cycle	ON for 1 PWM clock cycle
V19	OFF	OFF	OFF	OFF	OFF	OFF	OFF
U19	ON	ON	ON	ON	ON	ON	OFF
E19	OFF	ON	ON	ON	OFF	OFF	ON
U16	ON	OFF	ON	OFF	ON	OFF	ON

EXERCISE PROBLEMS:

1. Write a Verilog description for the “1011” sequence detector using Moore FSM. Simulate, synthesize, and implement it on the on-board peripherals of Basys3 FPGA kit.
2. Write a Verilog description to design an 8-bit counter using T flip-flops. Your design needs to be hierarchical, using a T flip-flop in behavioral modeling, and the rest should be either in dataflow or gate-level modeling. Develop a testbench and validate the design. Assign Clock input, Clear_n, Enable, and Q. Implement the design and verify the functionality in hardware.
3. Write a Verilog description to implement a 4-digit ring counter on the 7-segment display. Verify its functionality on Basys3 FPGA kit.