# FPGA Application Week 5 Homework

CYEE 10828241 Chen Da-Chuan

March 26, 2023

## Contents

# List of Figures

# List of Tables



# 1 Counter controlled with button and switch

## 1.1 Without debounce

### 1.1.1 Objective

Code a counter controlled by button and switch. One of the counter is controlled with button and the other is controlled with switch.

### 1.1.2 Operation

Two buttons and two switches are needed for this code. One button/switch is for counting up and the other is for counting down. Two of the six 7-segment LED is used to display counting result controlled by buttons, the other two is used to display counting result controlled by switches.

### 1.1.3 Code

```
1
2   //=============================================================
3   //  This code is generated by Terasic System Builder
4   //=============================================================
5
6   module DE10_LITE_BUTTON_COUNTER(
7
8       ////////// CLOCK //////////
9       input                       ADC_CLK_10,
10      input                       MAX10_CLK1_50,
11      input                       MAX10_CLK2_50,
12
13      ////////// SEG7 //////////
14      output          [7:0]       HEX0,
15      output          [7:0]       HEX1,
16      output          [7:0]       HEX2,
17      output          [7:0]       HEX3,
18      output          [7:0]       HEX4,
19      output          [7:0]       HEX5,
20
21      ////////// KEY //////////
22      input           [1:0]       KEY,
23
24      ////////// SW //////////
25      input           [9:0]       SW
26
27  );
28
29  //=============================================================
30  //  REG/WIRE declarations
31  //=============================================================
32
33  wire [6:0] count_SW, count_KEY;
34  wire [3:0] count_KEY_digit, count_KEY_ten;
35  wire [3:0] count_SW_digit, count_SW_ten;
36
37  //=============================================================
38  //  Structural coding
39  //=============================================================
40
41  assign count_KEY_digit = count_KEY[6:0]%10;
42  assign count_KEY_ten   = (count_KEY[6:0]/10)%10;
43  assign count_SW_digit  = count_SW[6:0]%10;
44  assign count_SW_ten    = (count_SW[6:0]/10)%10;
45
46  COUNTER_BUTTON ucounter(
47      .rst(SW[9]),
48      .KEY(KEY),
49      .SW(SW[1:0]),
50      .count_KEY(count_KEY),
51      .count_SW(count_SW)
52  );
53
54  SEG7_LUT_6 u_seg(
55      .oSEG0(HEX0),
56      .oSEG1(HEX1),
57      .oSEG2(HEX2),
58      .oSEG3(HEX3),
59      .oSEG4(HEX4),
60      .oSEG5(HEX5),
61      .iDIG({4'he, count_SW_ten, count_SW_digit, 4'he, count_KEY_ten, count_KEY_digit})
62  );
63
64  endmodule
65
```

Figure 1: Top level module

```
1   module COUNTER_BUTTON(
2       input           rst,
3       input       [1:0] SW,
4       input       [1:0] KEY,
5       output reg  [6:0] count_KEY,
6       output reg  [6:0] count_SW
7   );
8
9   always @ (KEY or SW or rst)
10  begin
11      if (rst)
12      begin
13          count_KEY <= 0;
14          count_SW <= 0;
15      end
16      else if (!KEY[0])
17      begin
18          count_KEY <= count_KEY+1;
19      end
20      else if (!KEY[1])
21      begin
22          count_KEY <= count_KEY-1;
23      end
24      else if (SW[0])
25      begin
26          count_SW <= count_SW+1;
27      end
28      else if (SW[1])
29      begin
30          count_SW <= count_SW-1;
31      end
32  end
33
34  endmodule
35
```

Figure 2: "COUNTER_BUTTON" sub-module

1. figure 1» Line6-27: Define the input and output ports.
2. figure 1» Line33-36: Declare wires used for counting switch/button up/down and their corresponding counter.
3. figure 1» Line41-44: Declare assign statement for each digit of both counter.
4. figure 1» Line46-52: Include the sub-module "COUNTER_BUTTON" for both switches and buttons.
5. figure 1» Line54-62: Include the 7-segment LED display module assigned with previously calculated digits.
6. figure 2» Line1-7: Define the input and output ports.

7. figure 2» Line11-15: When either key/switch/reset is triggered, if it's reset signal then reset both counter of key and switch to zero.
8. figure 2» Line16-19: When either key/switch/reset is triggered, if it's key[0] then count up.
9. figure 2» Line20-23: When either key/switch/reset is triggered, if it's key[1] then count down.
10. figure 2» Line24-27: When either key/switch/reset is triggered, if it's switch[0] then count up.
11. figure 2» Line28-31: When either key/switch/reset is triggered, if it's switch[1] then count down.

#### 1.1.4 Execution Result

The result of this code is as expected can count multiple times everytime a button or switch is pressed, which is because the FPGA chip can process the same adding or substracting procedure multiple times with its high clock speed.

The result showcasing video is available on YouTube: `https://youtu.be/FHwe28Ld7lw`.

### 1.2 With debounce

#### 1.2.1 Objective

Code a counter controlled by button and switch. One of the counter is controlled with button and the other is controlled with switch. This is exactly the same as the previous section except that the button and switch are debounced.

#### 1.2.2 Operation

Two buttons and two switches are needed for this code. One button/switch is for counting up and the other is for counting down. Two of the six 7-segment LED is used to display counting result controlled by buttons, the other two is used to display counting result controlled by switches.

#### 1.2.3 Code

```
1   //=======================================================
2   //  This code is generated by Terasic System Builder
3   //=======================================================
4
5   module DE10_LITE_BUTTON_COUNTER(
6
7       //////////// CLOCK //////////
8       input                       ADC_CLK_10,
9       input                       MAX10_CLK1_50,
10      input                       MAX10_CLK2_50,
11
12      //////////// SEG7 //////////
13      output          [7:0]       HEX0,
14      output          [7:0]       HEX1,
15      output          [7:0]       HEX2,
16      output          [7:0]       HEX3,
17      output          [7:0]       HEX4,
18      output          [7:0]       HEX5,
19
20      //////////// KEY //////////
21      input           [1:0]       KEY,
22
23      //////////// SW //////////
24      input           [9:0]       SW
25
26  );
27
28  //=======================================================
29  //  REG/WIRE declarations
30  //=======================================================
31
32  wire [6:0] count_SW, count_KEY;
33  wire [3:0] count_KEY_digit, count_KEY_ten;
34  wire [3:0] count_SW_digit, count_SW_ten;
35
36  //=======================================================
37  //  Structural coding
38  //=======================================================
39
40  assign count_KEY_digit = count_KEY[6:0]%10;
41  assign count_KEY_ten   = (count_KEY[6:0]/10)%10;
42  assign count_SW_digit  = count_SW[6:0]%10;
43  assign count_SW_ten    = (count_SW[6:0]/10)%10;
44
45  COUNTER_DEBOUNCE u_counter(
46      .clk(MAX10_CLK1_50),
47      .rst(SW[9]),
48      .KEY(KEY),
49      .SW(SW[1:0]),
50      .count_KEY(count_KEY),
51      .count_SW(count_SW)
52  );
53
54  SEG7_LUT_6 u_seg(
55      .OSEG0(HEX0),
56      .OSEG1(HEX1),
57      .OSEG2(HEX2),
58      .OSEG3(HEX3),
59      .OSEG4(HEX4),
60      .OSEG5(HEX5),
61      .IDIG({4'he, count_SW_ten, count_SW_digit, 4'he, count_KEY_ten, count_KEY_digit})
62  );
63
64  endmodule
65
66
```

Figure 3: Top level module

```
1   module COUNTER_DEBOUNCE(
2       input           clk,
3       input           rst,
4       input      [1:0] SW,
5       input      [1:0] KEY,
6       output reg [6:0] count_KEY,
7       output reg [6:0] count_SW
8   );
9
10  reg key1_delay, key0_delay;
11  reg sw1_delay, sw0_delay;
12
13  always @ (posedge clk)
14  begin
15      if (rst)
16      begin
17          count_KEY <= 0;
18          count_SW  <= 0;
19      end
20      // This section compare current value with previously stored value in reg
21      else if (key0_delay && !KEY[0]) //press enable
22      begin
23          count_KEY <= count_KEY+1;
24      end
25      else if (!key1_delay && KEY[1]) //release enable
26      begin
27          count_KEY <= count_KEY-1;
28      end
29      else if (!sw0_delay && SW[0])   //switch-up enable
30      begin
31          count_SW <= count_SW+1;
32      end
33      else if (sw1_delay && !SW[1])   //switch-down enable
34      begin
35          count_SW <= count_SW-1;
36      end
37
38      // This section stores current value in reg
39      key1_delay <= KEY[1];
40      key0_delay <= KEY[0];
41      sw1_delay  <= SW[1];
42      sw0_delay  <= SW[0];
43  end
44
45  endmodule
46
```

Figure 4: "COUNTER_DEBOUNCE" sub-module

1. figure 3» Line6-27: Define the input and output ports.
2. figure 3» Line33-36: Declare wires used for counting switch/button up/down and their corresponding counter.
3. figure 3» Line41-44: Declare assign statement for each digit of both counter.

4. figure 3» Line46-53: Include the sub-module "COUNTER_DEBOUNCE" for both switches and buttons.
5. figure 3» Line55-63: Include the 7-segment LED display module assigned with previously calculated digits.
6. figure 4» Line1-8: Define the input and output ports.
7. figure 4» Line10-11: Declare register for the previous state of the button/switch.
8. figure 4» Line15-19: When its on the positive edge of clock, if the reset signal is triggered, then reset both counter of key and switch to zero.
9. figure 4» Line21-24: When its on the positive edge of clock, if the key[0] is triggered and the previous signal is high, then count up.
10. figure 4» Line25-28: When its on the positive edge of clock, if the key[1] isn't triggered and the previous signal is low, then count down.
11. figure 4» Line29-32: When its on the positive edge of clock, if the switch[0] is triggered and the previous signal is low, then count up.
12. figure 4» Line33-36: When its on the positive edge of clock, if the switch[1] isn't triggered and the previous signal is high, then count down.
13. figure 4» Line39-42: Store the current state of the button/switch to the register.

#### 1.2.4 Execution Result

The code only add or substract once everytime a button or switch is touched. However, the same code sometimes have maximum value of 99 which is the expected value, sometimes 27.

The result showcasing video is available on YouTube: `https://youtu.be/Kn6KyxE9-J4`.

# 2 Counter controlled with chip internal clock

## 2.1 Counter by 10

### 2.1.1 Objective

Code a counter controlled by chip internal clock. The counter will count up by 1 everytime the clock is triggered, and reset to zero when it reaches 10.

### 2.1.2 Operation

This code takes clock and reset signal as input and output the counting result to a 4 bits register.

### 2.1.3 Code

```
1  module counter_v1(
2      input clk,
3      input rst,
4      output reg [3:0] count_10
5  );
6
7  always @ (posedge clk or negedge rst)
8  begin
9      if (!rst)
10     begin
11         count_10 <= 0;
12     end
13     else if (count_10 == 9)
14     begin
15         count_10 <= 0;
16     end
17     else
18         count_10 <= count_10 + 1;
19 end
20
21 endmodule
22
```

Figure 5: Top level module

```
1  `timescale 1ns/1ns
2
3  module counter_tb_v1;
4
5  reg        clk, rst;
6  wire [3:0] count_10;
7
8  counter_v1 ucounter(
9      .clk(clk),
10     .rst(rst),
11     .count_10(count_10)
12 );
13
14 initial
15 begin
16     #0
17     rst = 0;
18     clk = 0;
19     #15
20     rst = 1;
21 end
22
23 always
24     #10
25     clk = ~clk;
26
27 endmodule
```

Figure 6: Simulation code

1. figure 5» Line1-5: Define the input and output ports.
2. figure 5» Line6-12: When its on the positive edge of clock or negative edge of reset, if reset is triggered, reset the counter to zero.
3. figure 5» Line13-16: When its on the positive edge of clock, if value of counter is 9 which is the maximum allowed value, then reset the counter to zero.
4. figure 5» Line17-18: When its on the positive edge of clock, if value of counter is less than 9, then count up.
5. figure 6» Line1-1: Define time unit as 1ns, and time precision as 1ns.
6. figure 6» Line5-6: Declare wire for counter value and register for clock and reset signal.
7. figure 6» Line8-12: Include and instantiate the counter module.
8. figure 6» Line14-21: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.
9. figure 6» Line23-25: Inverse "clk" every 10 time unit.

### 2.1.4 Execution Result

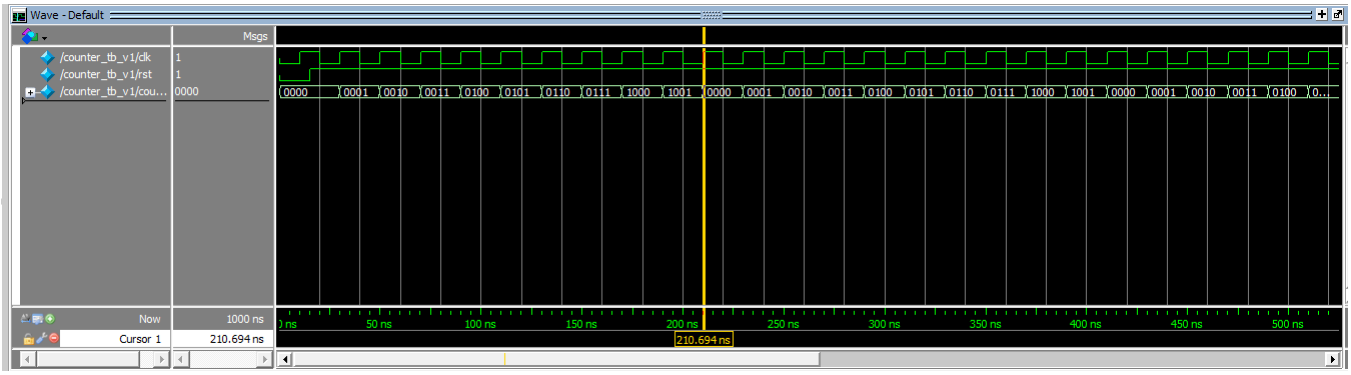


Figure 7: Simulation result

Table 1: Simulation result of first 210 ns

| **rst** | **count_10** in binary | **count_10** in decimal |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 1 | 0010 | 2 |
| 1 | 0011 | 3 |
| 1 | 0100 | 4 |
| 1 | 0101 | 5 |
| 1 | 0110 | 6 |
| 1 | 0111 | 7 |
| 1 | 1000 | 8 |
| 1 | 1001 | 9 |
| 1 | 0000 | 0 |

When the reset signal is high, the counter counts up and stores value in register. All the value is correct.

## 2.2 Counter with frequency divider by 5

### 2.2.1 Objective

Code a counter triggered with clock, and include a frequency divider by 5.

### 2.2.2 Operation

This module takes clock and reset signal as input and output the counting result to a 4 bits register and frequency divided clock signal.

### 2.2.3 Code

```
1  module counter_v2(
2      input clk,
3      input rst,
4      output reg [3:0] count_10,
5      output reg       clk_div10
6  );
7
8  always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_10  <= 0;
13         clk_div10 <= 0;
14     end
15     else if (count_10 == 4)
16     begin
17         count_10 <= 0;
18         clk_div10 <= ~clk_div10;
19     end
20     else
21         count_10 <= count_10 + 1;
22 end
23
24 endmodule
```

Figure 8: Top level module

```
1   `timescale 1ns/1ns
2
3   module counter_tb_v2;
4
5   reg        clk, rst;
6   wire [3:0] count_10;
7   wire       clk_div10;
8
9   counter_v2 ucounter(
10      .clk(clk),
11      .rst(rst),
12      .count_10(count_10),
13      .clk_div10(clk_div10)
14  );
15
16  initial
17  begin
18      #0
19      rst = 0;
20      clk = 0;
21      #15
22      rst = 1;
23  end
24
25  always
26      #10
27      clk = ~clk;
28
29  endmodule
30
```

Figure 9: Simulation code

1. figure 8» Line1-6: Define the input and output ports.
2. figure 8» Line10-14: When its on the positive edge of clock or negative edge of reset, if reset is triggered, reset the counter and divider register to zero.
3. figure 8» Line15-19: When its on the positive edge of clock, if value of counter is 4 which is the maximum allowed value, then reset the counter to zero and invert divider register signal.
4. figure 8» Line20-21: When its on the positive edge of clock, if value of counter is less than 4, then count up.
5. figure 9» Line1-1: Define time unit as 1ns, and time precision as 1ns.
6. figure 9» Line5-7: Declare wire for counter and dividre value, register for clock and reset signal.
7. figure 9» Line9-14: Include and instantiate the counter module.
8. figure 9» Line16-23: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.Lin
9. figure 9» Line25-27: Inverse "clk" every 10 time unit.

### 2.2.4 Execution Result

Table 2: Simulation result of first 250 ns

| **rst** | **count_10** in binary | **count_10** in decimal | **clk_div10** |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 0 |
| 1 | 0010 | 2 | 0 |
| 1 | 0011 | 3 | 0 |
| 1 | 0100 | 4 | 0 |
| 1 | 0000 | 0 | 1 |
| 1 | 0001 | 1 | 1 |
| 1 | 0010 | 2 | 1 |
| 1 | 0011 | 3 | 1 |
| 1 | 0100 | 4 | 1 |
| 1 | 0000 | 0 | 0 |

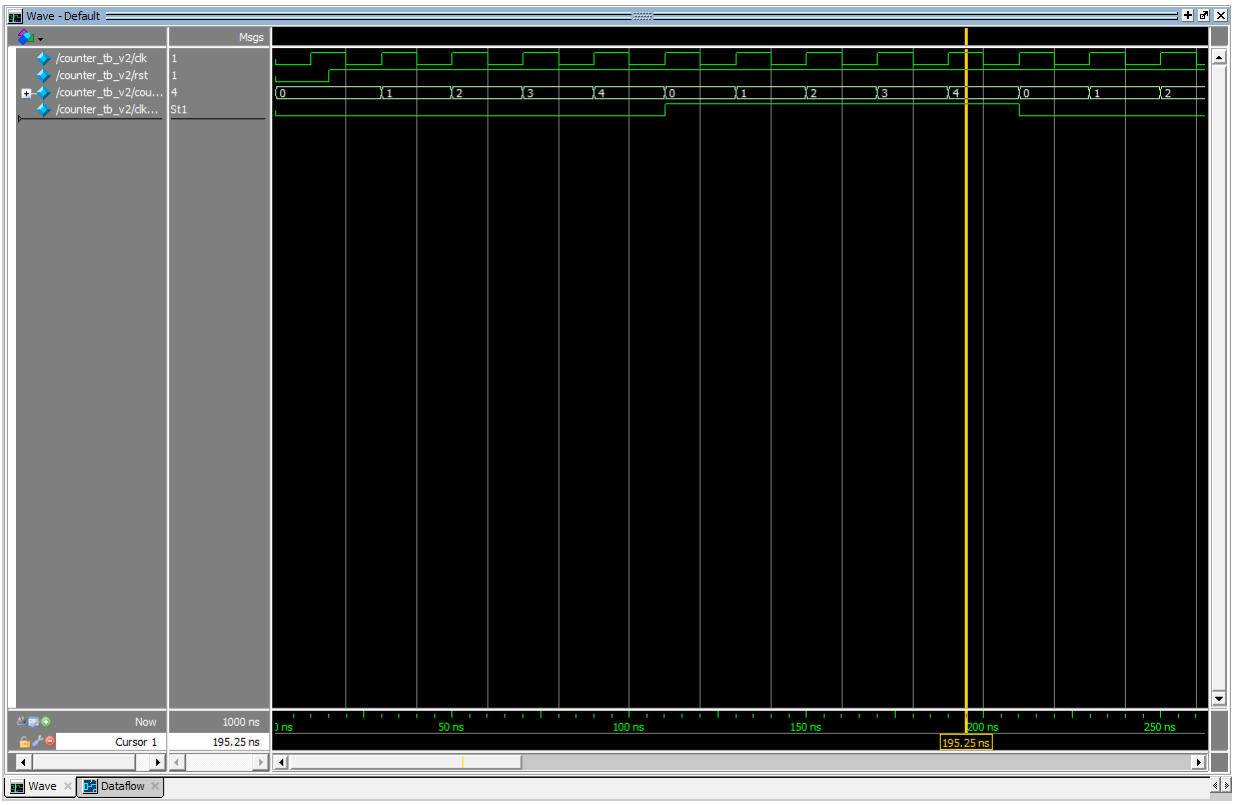


Figure 10: Simulation result

When the reset signal is high, the counter counts up and stores value in register. However, because the maximum counter value is 4, the counter will reset to zero after 5 counts. The frequency divider will invert the signal every 5 counts. This makes the counter can't count to 10 while the output divided clock having the period of 10 count which is what we are looking for.

## 2.3 Counter with frequency divider by 10

### 2.3.1 Objective

Code a counter triggered with clock, and include a frequency divider by 10. The difference with previous design is that the counter can count properly up to 9 and the output divided clock is still correct.

### 2.3.2 Operation

This module takes clock and reset signal as input and output the counting result to a 4 bits register and frequency divided clock signal.

### 2.3.3 Code

1. figure 11» Line1-6: Define the input and output ports.
2. figure 11» Line10-13: When its on the positive edge of clock or negative edge of reset, if reset is triggered, reset the counter to zero.
3. figure 11» Line14-17: When its on the positive edge of clock or negative edge of reset, if value of counter is 9 which is the maximum allowed value, then reset the counter to zero.
4. figure 11» Line18-19: When its on the positive edge of clock or negative edge of reset, if value of counter is less than 9, then count up.

```
1  module counter_v3(
2      input clk,
3      input rst,
4      output reg [3:0] count_10,
5      output reg       clk_div10
6  );
7
8  always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_10  <= 0;
13     end
14     else if (count_10 == 9)
15     begin
16         count_10 <= 0;
17     end
18     else
19         count_10 <= count_10 + 1;
20 end
21
22 always @ (posedge clk or negedge rst)
23 begin
24     if (!rst)
25     begin
26         clk_div10 <= 0;
27     end
28     else if (count_10 < 4 | count_10 == 9)
29     begin
30         clk_div10 <= 1;
31     end
32     else
33         clk_div10 <= 0;
34 end
35
36 endmodule
```

Figure 11: Top level module

```
1  `timescale 1ns/1ns
2
3  module counter_tb_v3;
4
5  reg        clk, rst;
6  wire [3:0] count_10;
7  wire       clk_div10;
8
9  counter_v3 ucounter(
10     .clk(clk),
11     .rst(rst),
12     .count_10(count_10),
13     .clk_div10(clk_div10)
14 );
15
16 initial
17 begin
18     #0
19     rst = 0;
20     clk = 0;
21     #15
22     rst = 1;
23 end
24
25 always
26     #10
27     clk = ~clk;
28
29 endmodule
30
```

Figure 12: Simulation code

5. figure 11» Line24-27: When its on the positive edge of clock or negative edge of reset, if reset is triggered, reset the divider register to zero.
6. figure 11» Line28-31: When its on the positive edge of clock or negative edge of reset, if value of counter is either 9/0/1/2/3, then set the divider register high.
7. figure 11» Line32-33: When its on the positive edge of clock or negative edge of reset, if value of counter is either 4/5/6/7/8, then set the divider register low.
8. figure 12» Line1-1: Define time unit as 1ns, and time precision as 1ns.
9. figure 12» Line5-7: Declare wire for counter and dividre value, register for clock and reset signal.
10. figure 12» Line9-14: Include and instantiate the counter module.
11. figure 12» Line16-23: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.
12. figure 12» Line25-27: Inverse "clk" every 10 time unit.

### 2.3.4 Execution Result

Table 3: Simulation result of first 250 ns

| **rst** | **count_10** in binary | **count_10** in decimal | **clk_div10** |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 1 | 0010 | 2 | 1 |
| 1 | 0011 | 3 | 1 |
| 1 | 0100 | 4 | 1 |
| 1 | 0101 | 5 | 0 |
| 1 | 0110 | 6 | 0 |
| 1 | 0111 | 7 | 0 |
| 1 | 1000 | 8 | 0 |
| 1 | 1001 | 9 | 0 |
| 1 | 0000 | 0 | 1 |
| 1 | 0001 | 1 | 1 |

When the reset signal is high, the counter counts up and stores value in register. This time the counter can keep the value correct while keeping divided clock signal correct.

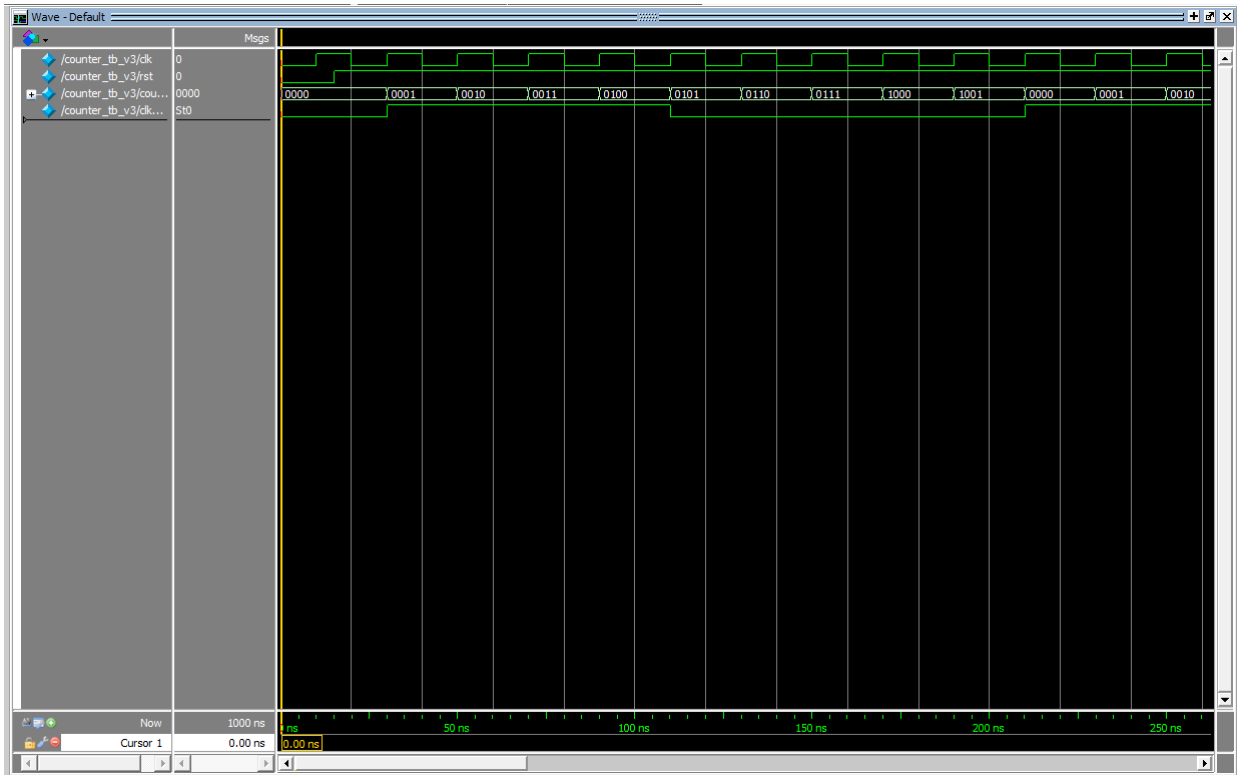


Figure 13: Simulation result

## 2.4 Counter with frequency divider by 10 with dislocation

### 2.4.1 Objective

Code a slightly simplier counter triggered with clock, and include a frequency divider by 10. The difference with previous design is that the output clock is delayed by 1 clock cycle.

### 2.4.2 Operation

This module takes clock and reset signal as input and output the counting result to a 4 bits register and frequency divided clock signal.

### 2.4.3 Code

1. figure 14» Line1-6: Define the input and output ports.
2. figure 14» Line10-13: When its on the positive edge of clock or negative edge of reset, if reset is triggered, reset the counter to zero.
3. figure 14» Line14-17: When its on the positive edge of clock or negative edge of reset, if value of counter is 9 which is the maximum allowed value, then reset the counter to zero.
4. figure 14» Line18-19: When its on the positive edge of clock or negative edge of reset, if value of counter is less than 9, then count up.
5. figure 14» Line24-27: When its on the positive edge of clock or negative edge of reset, if reset is triggered, reset the divider register to zero.
6. figure 14» Line28-31: When its on the positive edge of clock or negative edge of reset, if value of counter is either 0/1/2/3/4, then set the divider register high.
7. figure 14» Line32-33: When its on the positive edge of clock or negative edge of reset, if value of counter is either 5/6/7/8/9, then set the divider register low.

```
1  module counter_v4(
2      input clk,
3      input rst,
4      output reg [3:0] count_10,
5      output reg       clk_div10
6  );
7
8  always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_10  <= 0;
13     end
14     else if (count_10 == 9)
15     begin
16         count_10 <= 0;
17     end
18     else
19         count_10 <= count_10 + 1;
20 end
21
22 always @ (posedge clk or negedge rst)
23 begin
24     if (!rst)
25     begin
26         clk_div10 <= 0;
27     end
28     else if (count_10 < 5)
29     begin
30         clk_div10 <= 1;
31     end
32     else
33         clk_div10 <= 0;
34 end
35
36 endmodule
```

Figure 14: Top level module

```
1  `timescale 1ns/1ns
2
3  module counter_tb_v4;
4
5  reg         clk, rst;
6  wire [3:0] count_10;
7  wire        clk_div10;
8
9  counter_v4 ucounter(
10     .clk(clk),
11     .rst(rst),
12     .count_10(count_10),
13     .clk_div10(clk_div10)
14 );
15
16 initial
17 begin
18     #0
19     rst = 0;
20     clk = 0;
21     #15
22     rst = 1;
23 end
24
25 always
26     #10
27     clk = ~clk;
28
29 endmodule
30
```

Figure 15: Simulation code

8. figure 15» Line1-1: Define time unit as 1ns, and time precision as 1ns.
9. figure 15» Line5-7: Declare wire for counter and dividre value, register for clock and reset signal.
10. figure 15» Line9-14: Include and instantiate the counter module.
11. figure 15» Line16-23: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.
12. figure 15» Line25-26: Inverse "clk" every 10 time unit.

### 2.4.4 Execution Result

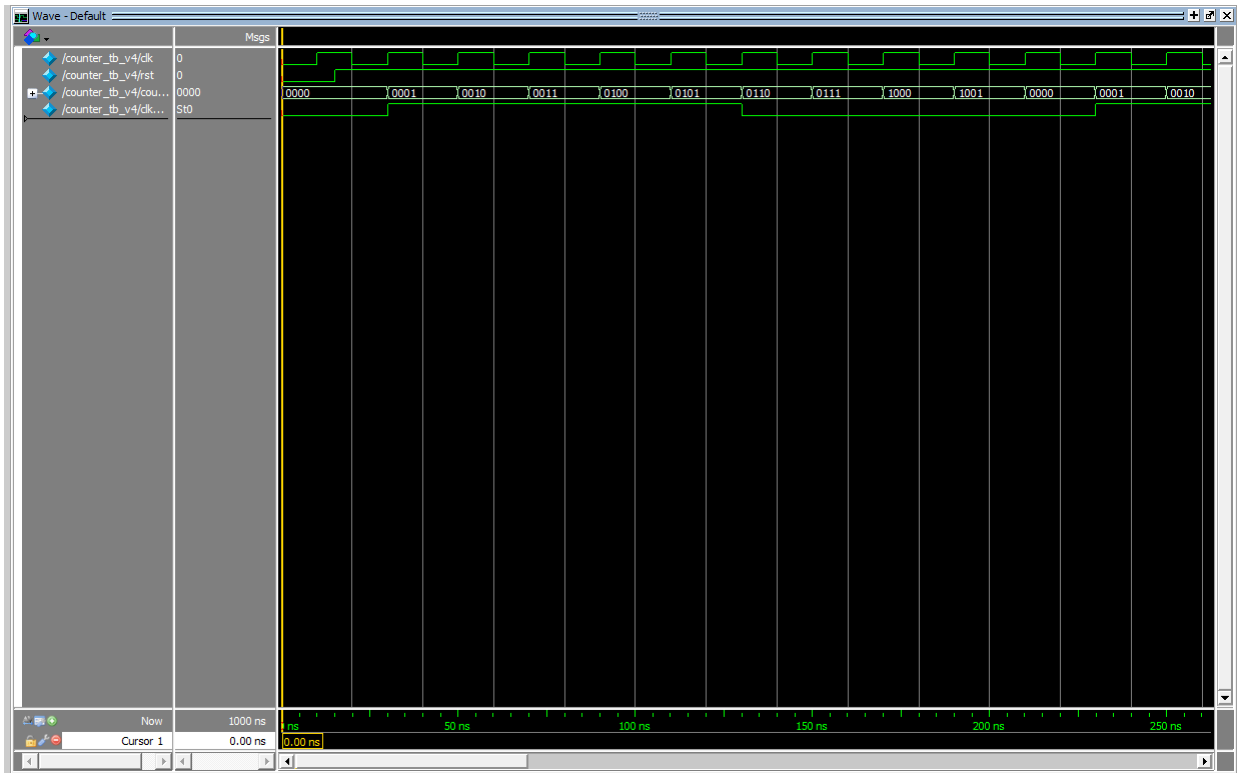


Figure 16: Simulation result of first 250 ns

Table 4: Simulation result of first 250 ns

| **rst** | **count_10** in binary | **count_10** in decimal | **clk_div10** |
|---|---|---|---|
| 0 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |
| 1 | 0010 | 2 | 1 |
| 1 | 0011 | 3 | 1 |
| 1 | 0100 | 4 | 1 |
| 1 | 0101 | 5 | 1 |
| 1 | 0110 | 6 | 0 |
| 1 | 0111 | 7 | 0 |
| 1 | 1000 | 8 | 0 |
| 1 | 1001 | 9 | 0 |
| 1 | 0000 | 0 | 0 |
| 1 | 0001 | 1 | 1 |

When the reset signal is high, the counter counts up and stores value in register. With a simplier value limitation, we can still output correct divided clock signal with a delay of 1 clock cycle.

## 2.5 Counter with frequency divider by 50x10

### 2.5.1 Objective

Code a slightly simplier counter triggered with clock, and include a frequency divider by 50 and 500. This is consisted with two counter modules, one for 50 and one for 10. The output divided clock signal is one clock cycle delayed.

### 2.5.2 Operation

This module takes clock and reset signal as input and output the counting result to a 3/5 bits register and frequency divided clock signal of $\frac{1}{50} \Big/ \frac{1}{500}$.

### 2.5.3 Code

```
1  module counter_v5(
2      input clk,
3      input rst,
4      output wire [3:0] count_10,
5      output wire [5:0] count_50,
6      output wire       clk_div10,
7      output wire       clk_div50
8  );
9
10 counter_v5_50 counter_50(
11     .clk(clk),
12     .rst(rst),
13     .count_50(count_50),
14     .clk_div50(clk_div50)
15 );
16
17 counter_v5_10 counter_10(
18     .clk(clk_div50),
19     .rst(rst),
20     .count_10(count_10),
21     .clk_div10(clk_div10)
22 );
23
24  endmodule
```

Figure 17: Top level module

```
1  module counter_v5_10(
2      input clk,
3      input rst,
4      output reg [3:0] count_10,
5      output reg       clk_div10
6  );
7
8   always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_10  <= 0;
13     end
14     else if (count_10 == 9)
15     begin
16         count_10 <= 0;
17     end
18     else
19         count_10 <= count_10 + 1;
20  end
21
22   always @ (posedge clk or negedge rst)
23 begin
24     if (!rst)
25     begin
26         clk_div10 <= 0;
27     end
28     else if (count_10 < 5)
29     begin
30         clk_div10 <= 1;
31     end
32     else
33         clk_div10 <= 0;
34  end
35
36   endmodule
```

Figure 18: Counter/divider by 10 module

1. figure 17» Line1-8: Define the input and output signal of the module.
2. figure 17» Line10-15: Include and instantiate the counter/divider by 50 module.
3. figure 17» Line17-22: Include and instantiate the counter/divider by 10 module.

```
1  module counter_v5_50(
2      input clk,
3      input rst,
4      output reg [5:0] count_50,
5      output reg       clk_div50
6  );
7
8  always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_50  <= 0;
13     end
14     else if (count_50 == 49)
15     begin
16         count_50 <= 0;
17     end
18     else
19         count_50 <= count_50 + 1;
20 end
21
22 always @ (posedge clk or negedge rst)
23 begin
24     if (!rst)
25     begin
26         clk_div50 <= 0;
27     end
28     else if (count_50 < 25)
29     begin
30         clk_div50 <= 1;
31     end
32     else
33         clk_div50 <= 0;
34 end
35
36 endmodule
```

Figure 19: Counter/divider by 50 module

```
1   `timescale 1ns/1ns
2
3   module counter_tb_v5;
4
5   reg         clk, rst;
6   wire [5:0] count_50;
7   wire [3:0] count_10;
8   wire        clk_div50, clk_div10;
9
10  counter_v5 counter(
11      .clk(clk),
12      .rst(rst),
13      .count_10(count_10),
14      .count_50(count_50),
15      .clk_div10(clk_div10),
16      .clk_div50(clk_div50)
17  );
18
19  initial
20  begin
21      #0
22      rst = 0;
23      clk = 0;
24      #15
25      rst = 1;
26  end
27
28  always
29      #10
30      clk = ~clk;
31
32  endmodule
33
```

Figure 20: Simulation code

4. figure 18» Line1-6: Define the input and output signal of the module.
5. figure 18» Line10-13: When its on the positive edge of clock or negative edge of reset, if reset is high, then set the counter to zero.
6. figure 18» Line14-17: When its on the positive edge of clock or negative edge of reset, if counter register is 9, then reset counter register.
7. figure 18» Line18-19: When its on the positive edge of clock or negative edge of reset, if counter register is smaller than 9, then count up.
8. figure 18» Line24-27: When its on the positive edge of clock or negative edge of reset, if reset is high, then set the output clock register to zero.
9. figure 18» Line28-31: When its on the positive edge of clock or negative edge of reset, if counter register is 0/1/2/3/4, then set the output clock register to high.
10. figure 18» Line32-33: When its on the positive edge of clock or negative edge of reset, if counter register is 5/6/7/8/9, then set the output clock register to low.
11. figure 19» Line1-6: Define the input and output signal of the module.
12. figure 19» Line10-13: When its on the positive edge of clock or negative edge of reset, if reset is high, then set the counter to zero.
13. figure 19» Line14-17: When its on the positive edge of clock or negative edge of reset, if counter register is 49, then reset counter register.
14. figure 19» Line18-19: When its on the positive edge of clock or negative edge of reset, if counter register is smaller than 49, then count up.
15. figure 19» Line24-27: When its on the positive edge of clock or negative edge of reset, if reset is high, then set the output clock register to zero.
16. figure 19» Line28-31: When its on the positive edge of clock or negative edge of reset, if counter register is $< 25$, then set the output clock register to high.
17. figure 19» Line32-33: When its on the positive edge of clock or negative edge of reset, if counter register is $> 24$, then set the output clock register to low.
18. figure 20» Line1-1: Define time unit as 1ns, and time precision as 1ns.
19. figure 20» Line5-8: Declare wire for counter and dividre value, register for clock and reset signal.

20. figure 20» Line10-17: Include and instantiate the counter module.
21. figure 20» Line19-26: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.
22. figure 20» Line28-30: Inverse "clk" every 10 time unit.

### 2.5.4 Execution Result

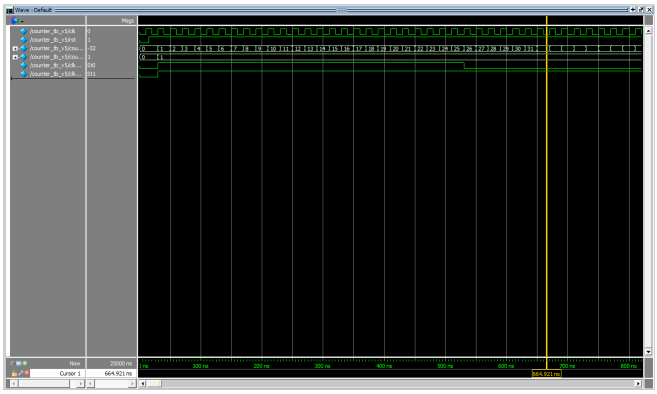

Figure 21: Simulation result of first $\frac{1}{500}$ high clock signal

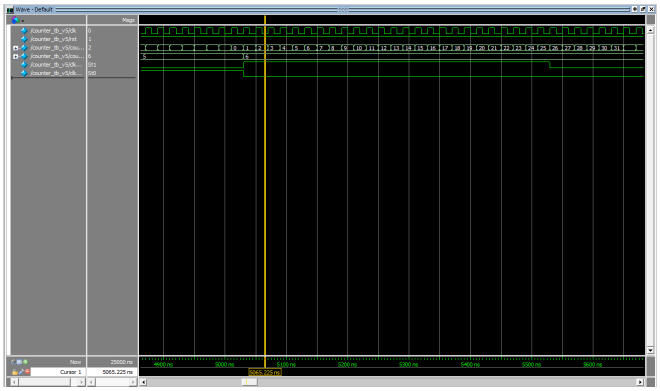

Figure 22: Simulation result of first $\frac{1}{500}$ low clock signal

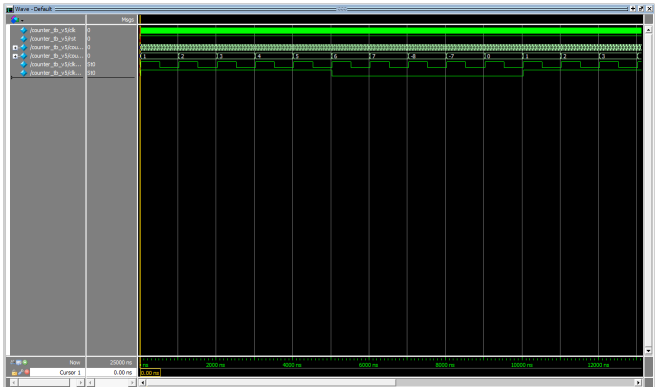

Figure 23: Simulation result of first period of $\frac{1}{500}$ signal

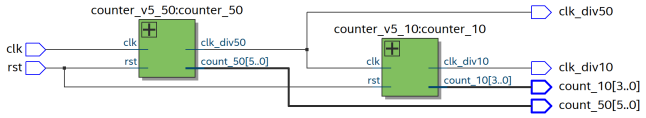


Figure 24: Circuit diagram of two sub-modules.

In figure 21, the $\frac{1}{50}$ and $\frac{1}{500}$ signal rises at the same time at the first "1" of register. In figure 22 $\frac{1}{500}$ lowers when the register of $\frac{1}{50}$ reaches "6". In figure 24 shows how two sub-modules are connected. From the simulation, we can see that the circuit works fine.

## 2.6 Counter with frequency divider by 10x10

### 2.6.1 Objective

Code a slightly simplier counter triggered with clock, and include a frequency divider by 50 and 500. This is consisted with two counter modules both are $\frac{1}{10}$ input clock. The output divided clock signal is one clock cycle delayed.

### 2.6.2 Operation

This module takes clock and reset signal as input and output the counting result to a 4 bits register and frequency divided clock signal of $\frac{1}{10}\Big/\frac{1}{100}$. Module coded in previous section is also used in this section.

### 2.6.3 Code

```
1  module counter_v6(
2      input clk,
3      input rst,
4      output wire [3:0] count_10_low,
5      output wire [3:0] count_10_high,
6      output wire       clk_div10,
7      output wire       clk_div100
8  );
9
10 counter_v5_10 counter_10_low(
11     .clk(clk),
12     .rst(rst),
13     .count_10(count_10_low),
14     .clk_div10(clk_div10)
15 );
16
17 counter_v5_10 counter_10_high(
18     .clk(clk_div10),
19     .rst(rst),
20     .count_10(count_10_high),
21     .clk_div10(clk_div100)
22 );
23
24 endmodule
25
```

Figure 25: Top level modules

```
1  `timescale 1ns/1ns
2
3  module counter_tb_v6;
4
5  reg        clk, rst;
6  wire [3:0] count_10;
7  wire [3:0] count_100;
8  wire       clk_div10, clk_div100;
9
10 counter_v6 counter(
11     .clk(clk),
12     .rst(rst),
13     .count_10_low(count_10),
14     .count_10_high(count_100),
15     .clk_div10(clk_div10),
16     .clk_div100(clk_div100)
17 );
18
19 initial
20 begin
21     #0
22     rst = 0;
23     clk = 0;
24     #15
25     rst = 1;
26 end
27
28 always
29     #10
30     clk = ~clk;
31
32 endmodule
33
```

Figure 26: Simulation code

1. figure 25» Line1-8: Define the input and output signal of the module.
2. figure 25» Line10-15: Include and instantiate the counter module.
3. figure 25» Line17-22: Include and instantiate the counter module.
4. figure 26» Line1-1: Define time unit as 1ns, and time precision as 1ns.
5. figure 26» Line5-8: Declare wire for counter and dividre value, register for clock and reset signal.
6. figure 26» Line10-17: Include and instantiate the counter module.
7. figure 26» Line19-26: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.
8. figure 26» Line28-30: Inverse "clk" every 10 time unit.

### 2.6.4 Execution Result

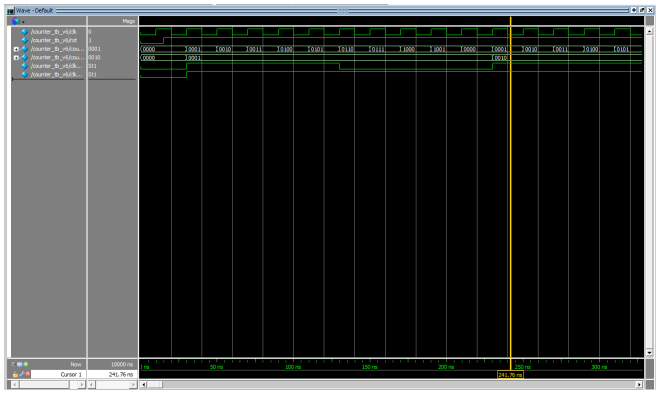

Figure 27: Simulation result of first $\frac{1}{10}$ high clock signal

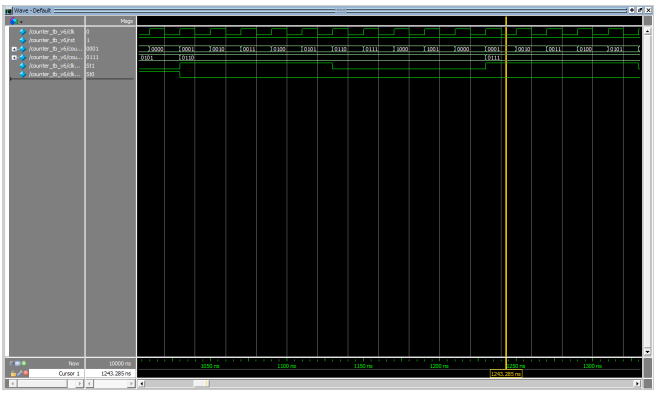

Figure 28: Simulation result of first $\frac{1}{100}$ low clock signal

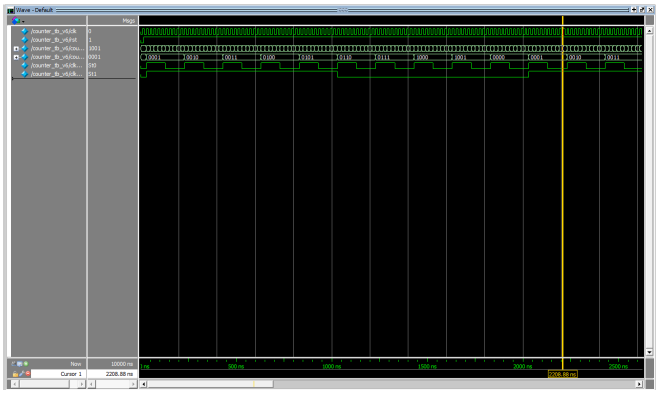

Figure 29: Simulation result of first period of $\frac{1}{100}$ signa

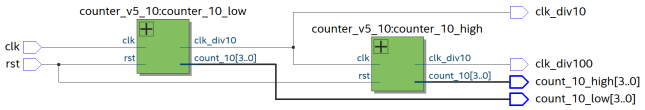


Figure 30: Circuit diagram of two sub-modules.

In figure 27, the $\frac{1}{10}$ and $\frac{1}{100}$ signal rises at the same time at the first "1" of register. In figure 28 $\frac{1}{100}$ lowers when the register of $\frac{1}{10}$ reaches "6". In figure 30 shows how two sub-modules are connected. From the simulation, we can see that the circuit works fine.

## 2.7 Counter with frequency divider by 24x60x60

### 2.7.1 Objective

Code a slightly simplier counter triggered with clock, and include a frequency divider by 60, 60x60, 60x60x24. This is consisted with three counter module $\frac{1}{60}$, $\frac{1}{60}$, and $\frac{1}{24}$ input clock. The output divided clock signal is without one clock cycle delayed.

### 2.7.2 Operation

This module takes clock and reset signal as input and output the counting result to a 5/6 bits register and frequency divided clock signal of $\frac{1}{60}\Big/\frac{1}{3600}\Big/\frac{1}{86400}$.

### 2.7.3 Code

1. figure 31» Line30-32: Declare wire for second, minute, hour counter, and individual clock digit of second, minute, hour.
2. figure 31» Line34-34: Declare six digits of clock to display on 7-segment display.
3. figure 31» Line40-45: Assign each clock digit to their corresponding calculated value.
4. figure 31» Line49-57: When its positive edge of MAX10 chip internal clock 1 or negative edge of reset switch, if reset switch is off, set all clock digit to zero.
5. figure 31» Line58-66: When its positive edge of MAX10 chip internal clock 1 or negative edge of reset switch, if reset switch is on, set all clock digit to the counter output.
6. figure 31» Line69-75: Include and instantiate the counter module.
7. figure 31» Line77-85: Include and instantiate the 7-segment module.
8. figure 32» Line1-10: Define the input and output signal of the module.
9. figure 32» Line12-17: Include and instantiate the $\frac{1}{60}$ counter module.

```
26  //=======================================================
27  //  REG/WIRE declarations
28  //=======================================================
29
30  wire [5:0] count_second, count_minute;
31  wire [4:0] count_hour;
32  wire [3:0] hour_ten, hour_one, minute_ten, minute_one, second_ten, second_one;
33
34  reg [3:0] digit0, digit1, digit2, digit3, digit4, digit5;
35
36  //=======================================================
37  //  Structural coding
38  //=======================================================
39
40  assign hour_ten = count_hour/10;
41  assign hour_one = count_hour%10;
42  assign minute_ten = count_minute/10;
43  assign minute_one = count_minute%10;
44  assign second_ten = count_second/10;
45  assign second_one = count_second%10;
46
47  always @ (posedge MAX10_CLK1_50 or negedge SW[9])
48  begin
49      if (!SW[9])
50      begin
51          digit0 <= 4'hf;
52          digit1 <= 4'hf;
53          digit2 <= 4'hf;
54          digit3 <= 4'hf;
55          digit4 <= 4'hf;
56          digit5 <= 4'hf;
57      end
58      else
59      begin
60          digit0 <= second_one;
61          digit1 <= second_ten;
62          digit2 <= minute_one;
63          digit3 <= minute_ten;
64          digit4 <= hour_one;
65          digit5 <= hour_ten;
66      end
67  end
68
69  counter_v7 u_24hClock(
70      .clk(MAX10_CLK1_50),
71      .rst(SW[9]),
72      .count_60_1(count_second),
73      .count_60_2(count_minute),
74      .count_24(count_hour)
75  );
76
77  SEG7_LUT_6 u_seg(
78      .oSEG0(HEX0),
79      .oSEG1(HEX1),
80      .oSEG2(HEX2),
81      .oSEG3(HEX3),
82      .oSEG4(HEX4),
83      .oSEG5(HEX5),
84      .iDIG ({ digit5, digit4, digit3, digit2, digit1, digit0})
85  );
86
87  endmodule
88
```

Figure 31: Top level modules

```
1   module counter_v7(
2       input clk,
3       input rst,
4       output wire [5:0] count_60_1,
5       output wire [5:0] count_60_2,
6       output wire [4:0] count_24,
7       output wire       clk_div60_1,
8       output wire       clk_div60_2,
9       output wire       clk_div24
10  );
11
12  counter_v7_60 counter_60_1(
13      .clk(clk),
14      .rst(rst),
15      .count_60(count_60_1),
16      .clk_div60(clk_div60_1)
17  );
18
19  counter_v7_60 counter_60_2(
20      .clk(clk_div60_1),
21      .rst(rst),
22      .count_60(count_60_2),
23      .clk_div60(clk_div60_2)
24  );
25
26  counter_v7_24 counter_24(
27      .clk(clk_div60_2),
28      .rst(rst),
29      .count_24(count_24),
30      .clk_div24(clk_div24)
31  );
32
33  endmodule
34
```

Figure 32: Sub-module consisted with 2 $\frac{1}{60}$ counter and 1 $\frac{1}{24}$ counter

10. figure 32» Line19-24: Include and instantiate the $\frac{1}{60}$ counter module.
11. figure 32» Line26-31: Include and instantiate the $\frac{1}{24}$ counter module.
12. figure 33» Line1-6: Define the input and output signal of the module.
13. figure 33» Line10-13: When its positive edge of input clock signal or negative edge of reset switch, if reset switch is off, set the counter register to zero.
14. figure 33» Line14-17: When its positive edge of input clock signal or negative edge of reset switch, if counter register is 23, set the counter register to zero.
15. figure 33» Line18-21: When its positive edge of input clock signal or negative edge of reset switch, if counter register is not 23, set the counter register plus one.
16. figure 33» Line24-27: When its positive edge of input clock signal or negative edge of reset switch, if reset switch is off, set the clock register to zero.
17. figure 33» Line28-31: When its positive edge of input clock signal or negative edge of reset switch, if counter register is $23$ or $< 11$, set the clock register as one.
18. figure 33» Line32-35: When its positive edge of input clock signal or negative edge of reset switch, if counter register is $11 \leq x \leq 23$, set the clock register as zero.
19. figure 34» Line1-6: Define the input and output signal of the module.
20. figure 34» Line10-13: When its positive edge of input clock signal or negative edge of reset switch, if reset switch is off, set the counter register to zero.
21. figure 34» Line14-17: When its positive edge of input clock signal or negative edge of reset switch, if counter register is 59, set the counter register to zero.

```
1  module counter_v7_24(
2      input clk,
3      input rst,
4      output reg [4:0] count_24,
5      output reg       clk_div24
6  );
7
8  always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_24  <= 0;
13     end
14     else if (count_24 == 23)
15     begin
16         count_24 <= 0;
17     end
18     else
19         count_24 <= count_24 + 1;
20 end
21
22 always @ (posedge clk or negedge rst)
23 begin
24     if (!rst)
25     begin
26         clk_div24 <= 0;
27     end
28     // else if (count_24 < 12)
29     else if (count_24 < 11 | count_24 == 23)
30     begin
31         clk_div24 <= 1;
32     end
33     else
34         clk_div24 <= 0;
35 end
36
37 endmodule
```

Figure 33: $\frac{1}{24}$ counter module

```
1  module counter_v7_60(
2      input clk,
3      input rst,
4      output reg [5:0] count_60,
5      output reg       clk_div60
6  );
7
8  always @ (posedge clk or negedge rst)
9  begin
10     if (!rst)
11     begin
12         count_60  <= 0;
13     end
14     else if (count_60 == 59)
15     begin
16         count_60 <= 0;
17     end
18     else
19         count_60 <= count_60 + 1;
20 end
21
22 always @ (posedge clk or negedge rst)
23 begin
24     if (!rst)
25     begin
26         clk_div60 <= 0;
27     end
28     // else if (count_60 < 30)
29     else if (count_60 < 29 | count_60 == 59)
30     begin
31         clk_div60 <= 1;
32     end
33     else
34         clk_div60 <= 0;
35 end
36
37 endmodule
```

Figure 34: $\frac{1}{60}$ counter module

```
1  `timescale 1ns/1ns
2
3  module counter_tb_v7;
4
5  reg         clk, rst;
6  wire [5:0] count_60_1;
7  wire [5:0] count_60_2;
8  wire [4:0] count_24;
9  wire        clk_div60_1, clk_div60_2, clk_div24;
10
11 counter_v7 counter(
12     .clk(clk),
13     .rst(rst),
14     .count_60_1(count_60_1),
15     .count_60_2(count_60_2),
16     .count_24(count_24),
17     .clk_div60_1(clk_div60_1),
18     .clk_div60_2(clk_div60_2),
19     .clk_div24(clk_div24)
20 );
21
22 initial
23 begin
24     #0
25     rst = 0;
26     clk = 0;
27     #15
28     rst = 1;
29 end
30
31 always
32     #10
33     clk = ~clk;
34
35 endmodule
36
```

Figure 35: Simulation code

22. figure 34» Line18-19: When its positive edge of input clock signal or negative edge of reset switch, if counter register is not 59, set the counter register plus one.
23. figure 34» Line24-27: When its positive edge of input clock signal or negative edge of reset switch, if reset switch is off, set the clock register to zero.
24. figure 34» Line29-32: When its positive edge of input clock signal or negative edge of reset switch, if counter register is $59$ or $< 29$, set the clock register as one.
25. figure 34» Line33-34: When its positive edge of input clock signal or negative edge of reset switch, if counter register is $29 \leq x \leq 59$, set the clock register as zero.
26. figure 35» Line1-1: Define time unit as 1ns, and time precision as 1ns.
27. figure 35» Line5-9: Declare wire of count and clock signal, register of clock and reset signal.
28. figure 35» Line11-20: Include and instantiate the counter module.
29. figure 35» Line22-29: Set the initial value of signal "clk" and "rst" to zero, and turn "rst" at 15 time unit.
30. figure 35» Line31-33: Inverse "clk" every 10 time unit.

### 2.7.4 Execution Result

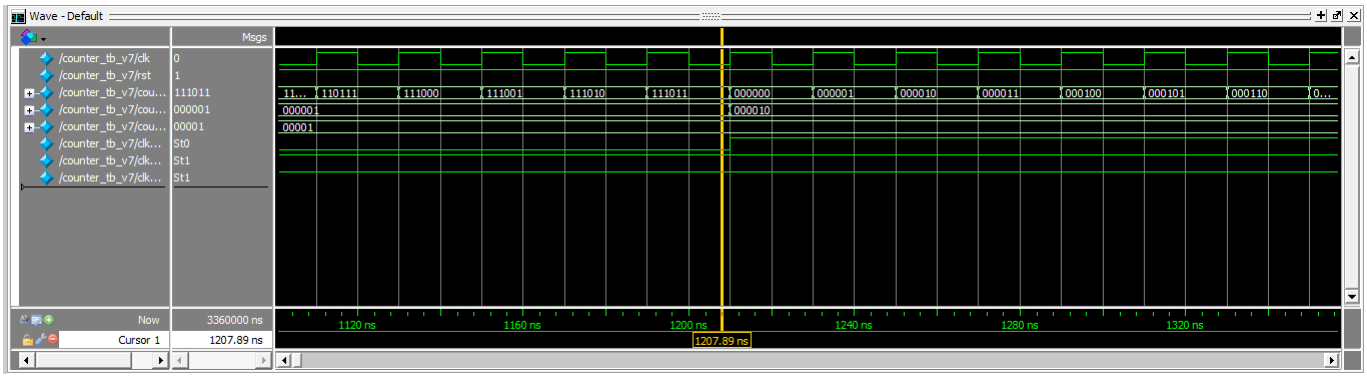


Figure 36: First divider signal rises at the first counter returns to zero

This result shows all three clock waveforms rises at the correct time, which means the counter module works correctly. The result showcasing video is available on YouTube: `https://youtu.be/w5nGt9CgHuY`.

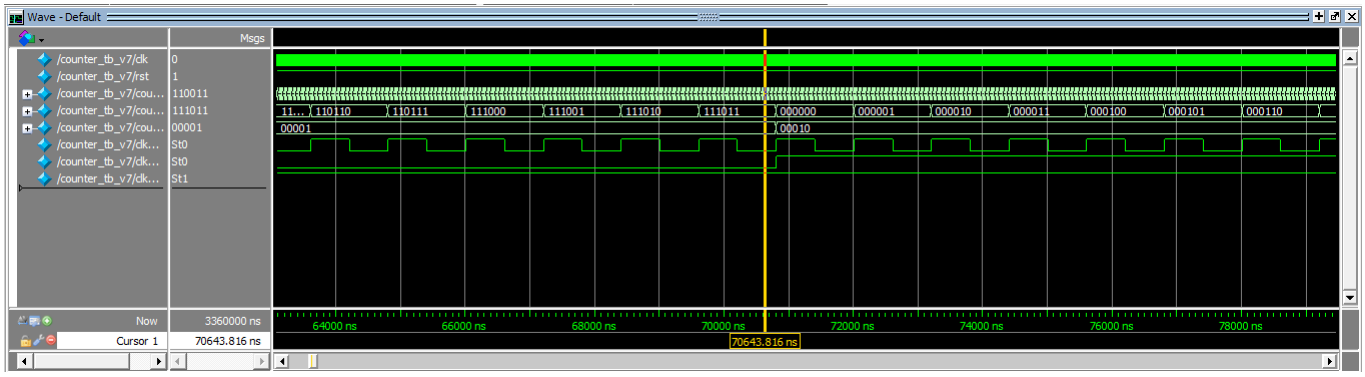


Figure 37: Second divider signal rises at the second counter returns to zero

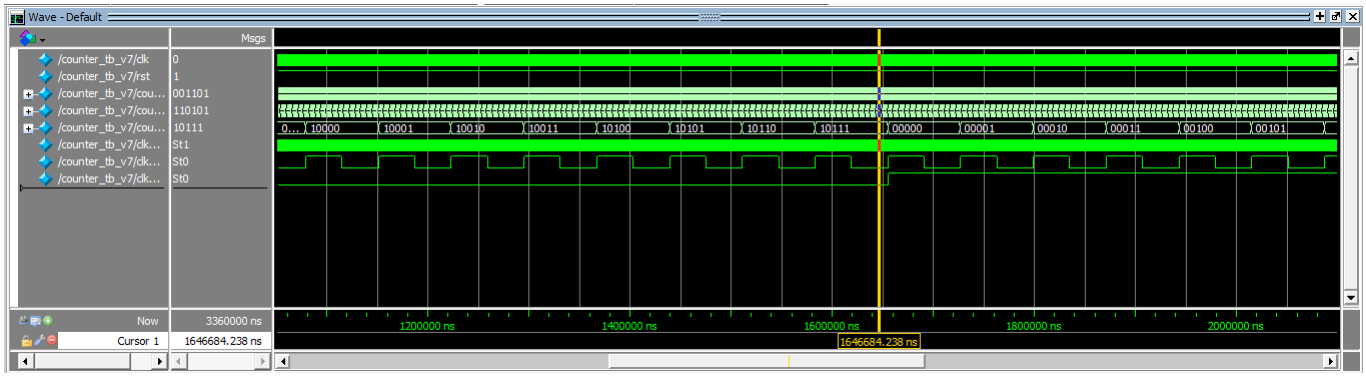


Figure 38: Third divider signal rises at the third counter returns to zero

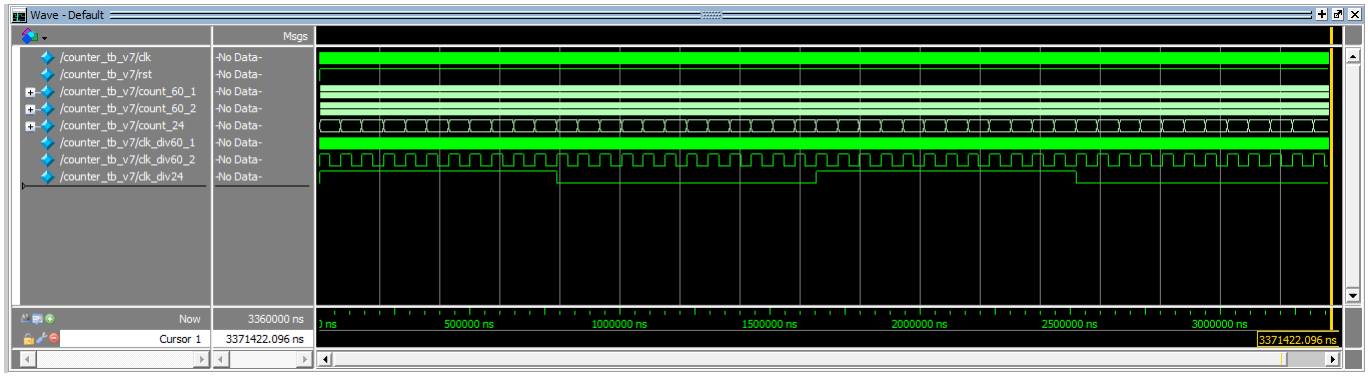


Figure 39: Full two period of the third divider signal

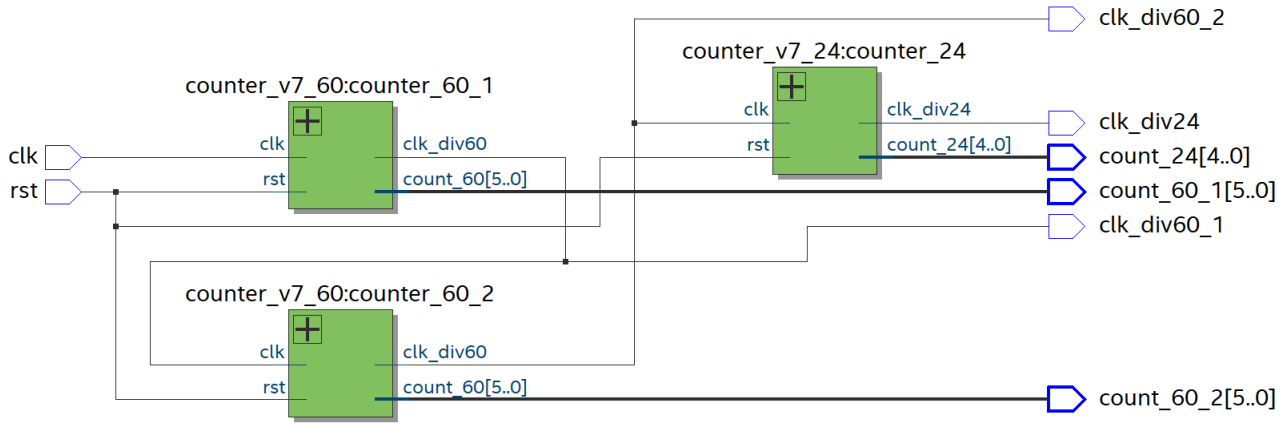


Figure 40: Circuit diagram of three sub-modules.