

ECSE 4770 LAB 2 REPORT

Yuchen Wang, Rin: 661960546, section 2

Partner: Yilu Zhou, Rin: 661931826, section 2

Abstract

- This lab is aimed to help student use Verilog build a finite state machine and learn the memory arrays.
- In this lab, student needs to build a 5-bit simple counter that can skip state 16 in Verilog, simulating a 5-bit up/down counter that skips all prime numbers, using count to read a memory block address, and run this counter on the DE10-lite FPGA board with hex display readout.

Simple 5-Bit Counter

- In this part, the student needs to implement the 5-bit timer with a basic 4-bit counter, 74163. 74163 is a synchronous counter, so the load and clear function need to be triggered with a input clock. To skip the state 16, students use extra logic gate to implement this function. For the MSB, a JK Flip-flop is used to implement it. The basic idea is that the ripple-carry output can trigger the JK flip-flop so the MSB will change. This function will be discussed in further section.

- **Schematic:**

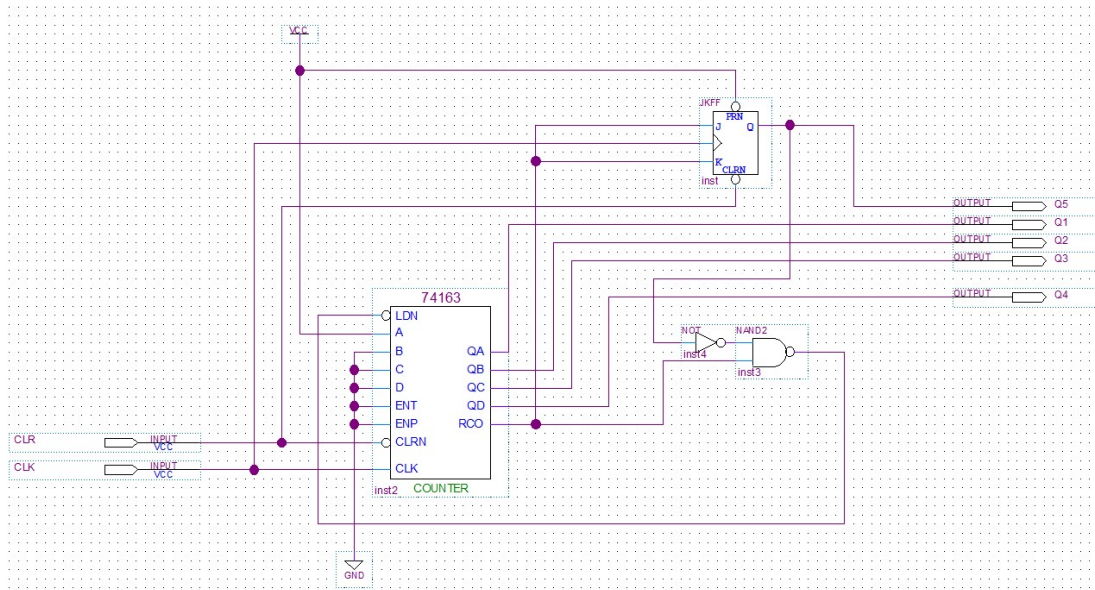


Figure 1: 5-bit simple timer schematic

- **RTL view**

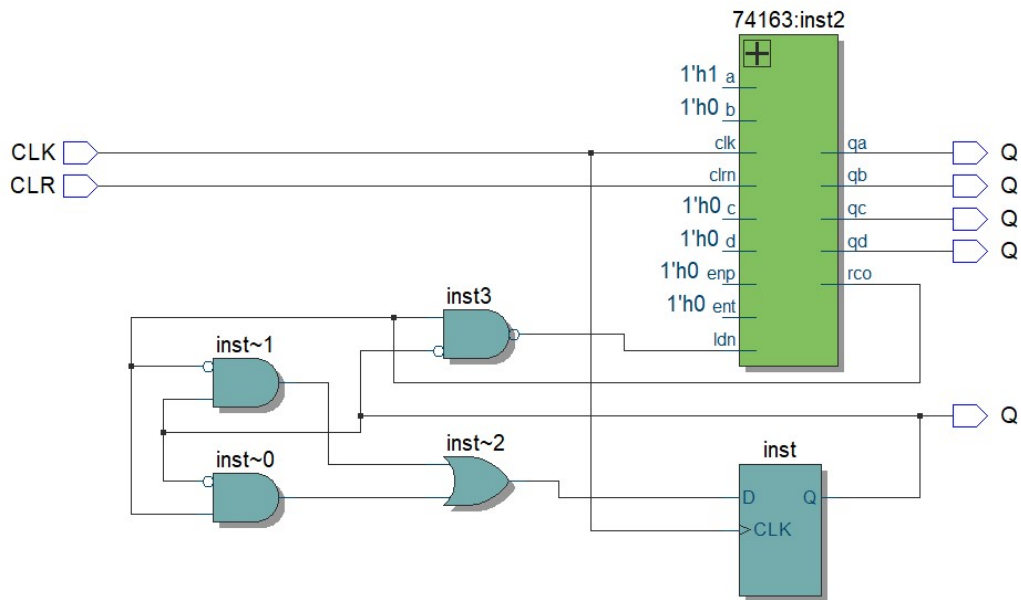


Figure 2: 5-bit simple timer RTL view

Verilog Code:

```
module counter74163(input CLK, input CLRN, input LDN, input [3:0] D,
output reg [3:0] Q, output reg RCO);
    always @(posedge CLK) begin
        if (!CLRN) begin // Clear all the bit
            Q = 4'b0000;
            RCO <= 1'b0;
        end
    end
    always @(posedge CLK) begin // synchronous counter
        if (!LDN) begin // trigger load, load input D
            Q = D;
            RCO <= 1'b0;
        end
        if (LDN) begin //no load, regular count
            Q =Q+1'b1; // Count
        end
        if (Q == 4'b1111)begin //RCO setting
            RCO <= 1'b1;
        end
        else begin // RCO clear
            RCO <= 1'b0;
        end
    end
end
endmodule
```

```
module JKFF ( // Basic JK flip flop
    input J, input K, input CLK, input set, output reg Q_JK
);
    always @(posedge set) begin //clear the flip flop
        Q_JK<= 0;
    end
    always @(posedge CLK) begin
        case ({J,K})
            2'b00: Q_JK<= Q_JK;
            2'b01: Q_JK<= 0;
            2'b10: Q_JK<= 1;
        end
    end
end
```

```

        2'b11: Q_JK<= ~Q_JK;
    endcase
end
endmodule

```

```

module AND (    // 2-bit AND
    input A, input B, output reg C
);
    always @(*) begin
        C<= A & B;
    end
endmodule

```

```

// 5-Bit Simple Counter skip state 16
module counter5(
    input CLK,
    input CLR,
    output [4:0] Q5
);
    wire RCO;
    wire AND1;
    wire [3:0] Q4;
    wire Q1;
    // Wire connection
    AND AND1(.A(RCO), .B(!Q1), .C(AND1));
    counter74163 counter74163(.CLK(CLK), .CLR(CLR), .LDN(!AND1),
        .D(4'b0001), .Q(Q4), .RCO(RCO));
    JKFF JKFF(.CLK(CLK), .J(RCO), .K(RCO), .set(CLR), .Q_JK(Q1));
    // Output
    assign Q5={Q1, Q4};
endmodule

```

The basic function of the 74163 is that when the CLK gets a positive edge, the output will increase 1 and when it reach 15, the ripple-carry output turns to 1 (otherwise it is 0). The next CLK input will make the output and RCO to 0.

To implement a 5 bit counter, An extra JK flip-flop is needed to implement the extra bit since 74163 is a 4-bit counter. Here this JK flip-flop will implement the MSB. As learned in COCO, the 74163 is implemented by four JK flip-flop. The method here is almost the same. The RCO is connected to both J and K in the flip-flop, and CLK is the same as the input CLK. By initialization, Q is set to 0 at the beginning. In the most case RCO is 0, where J and K are 0, so the Q of the flip-flop is the same as previous Q. When RCO is 1, where J and K are 1, next CLK will trigger a clear in the 74163, and here since J and K are 1, Q will be equal to inverted Q at this case. This help flip-flop the MSB when RCO is high.

The method to skip state 16 here is that when the 74163 reach state 15, the ripple-carry output (RCO) will turns from 0 to 1. After this state, the Q1, where the JK FF is 0 and the 5 bit counter state is 0b01111, the 2 bit AND whose output is ***!Q1 AND RCO*** will turns to 1. Then after an inverter, the LDN gets a 0 to trigger load process. Then after next positive edge in CLK, the 74163 will load the input of A, B, C, D, which is ***.D(4'b0001)*** in Verilog code. The 74613 does not increase but load 0b0001 here. So the output of 5-bit counter will be 0b10001 that skips state 16. When the 5-bit reach 0b11111, the Q1 is 1 so for 2-bit AND, the output, ***!Q1 AND RCO***, is 0 which will not trigger load after an inverter.

- Verification

Test branch code

```
module counter5_tb;
    reg CLK;
    reg CLR;
    wire [4:0] OUT;
    integer i;

    counter5 counter5(.CLK(CLK), .CLR(CLR), .Q5(OUT));
```

```

task display;
    $display("CLK: %1b, OUT:%5b, %2d",CLK, OUT, OUT);
endtask

initial begin
    $dumpfile("counter5.vcd");
    $dumpvars;
    $display("Test start");
    #1; // stop for 1 unit time
    CLR = 0;
    // create a positive edge to trigger initialization
    CLK = 0;
    #1;
    CLK = 1;
    #1;
    CLR = 1;
    for(i = 0; i<100 ; i++) begin    // repeat 100 times
        CLK = 0;
        #1;
        CLK = 1;
        display;
        #1;
    end
end

endmodule

```

- **Simulation Results:**

```
wangy62@DESKTOP-08TD290: /mnt/c/Users/wangy62/Lab2$ iverilog -o myDFF 5bit_counter_tb
.v counter5bit.v JKFF.v counter74163.v AND.v
wangy62@DESKTOP-08TD290: /mnt/c/Users/wangy62/Lab2$ vvp myDFF
VCD info: dumpfile counter5.vcd opened for output.
Test start
CLK: 1, OUT:00000, 0
CLK: 1, OUT:00001, 1
CLK: 1, OUT:00010, 2
CLK: 1, OUT:00011, 3
CLK: 1, OUT:00100, 4
CLK: 1, OUT:00101, 5
CLK: 1, OUT:00110, 6
CLK: 1, OUT:00111, 7
CLK: 1, OUT:01000, 8
CLK: 1, OUT:01001, 9
CLK: 1, OUT:01010, 10
CLK: 1, OUT:01011, 11
CLK: 1, OUT:01100, 12
CLK: 1, OUT:01101, 13
CLK: 1, OUT:01110, 14
CLK: 1, OUT:01111, 15
CLK: 1, OUT:10001, 17
CLK: 1, OUT:10010, 18
CLK: 1, OUT:10011, 19
CLK: 1, OUT:10100, 20
CLK: 1, OUT:10101, 21
CLK: 1, OUT:10110, 22
CLK: 1, OUT:10111, 23
CLK: 1, OUT:11000, 24
CLK: 1, OUT:11001, 25
CLK: 1, OUT:11010, 26
CLK: 1, OUT:11011, 27
CLK: 1, OUT:11100, 28
CLK: 1, OUT:11101, 29
CLK: 1, OUT:11110, 30
CLK: 1, OUT:11111, 31
CLK: 1, OUT:00000, 0
CLK: 1, OUT:00001, 1
CLK: 1, OUT:00010, 2
CLK: 1, OUT:00011, 3
CLK: 1, OUT:00100, 4
CLK: 1, OUT:00101, 5
CLK: 1, OUT:00110, 6
CLK: 1, OUT:00111, 7
CLK: 1, OUT:01000, 8
CLK: 1, OUT:01001, 9
CLK: 1, OUT:01010, 10
CLK: 1, OUT:01011, 11
CLK: 1, OUT:01100, 12
CLK: 1, OUT:01101, 13
CLK: 1, OUT:01110, 14
CLK: 1, OUT:01111, 15
CLK: 1, OUT:10001, 17
CLK: 1, OUT:10010, 18
CLK: 1, OUT:10011, 19
CLK: 1, OUT:10100, 20
CLK: 1, OUT:10101, 21
CLK: 1, OUT:10110, 22
CLK: 1, OUT:10111, 23
```

Figure 3: Verilog simulation result

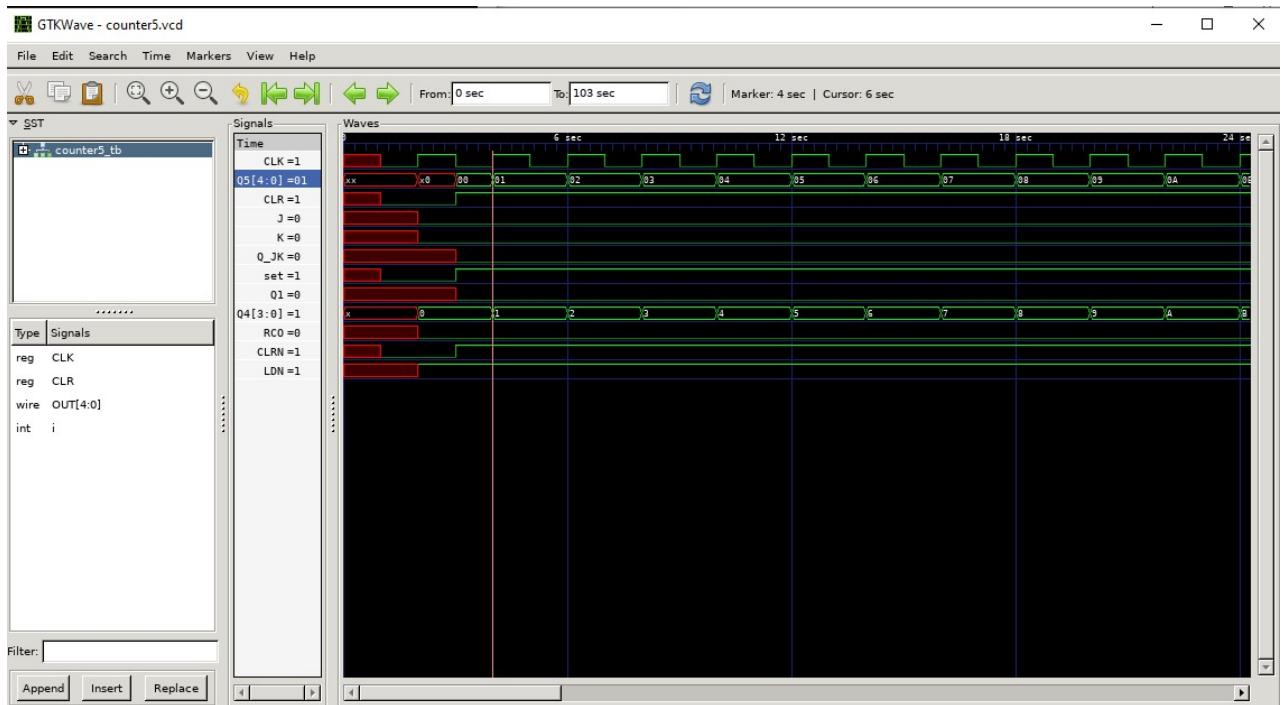


Figure 4: Simulation waveform result, initialization

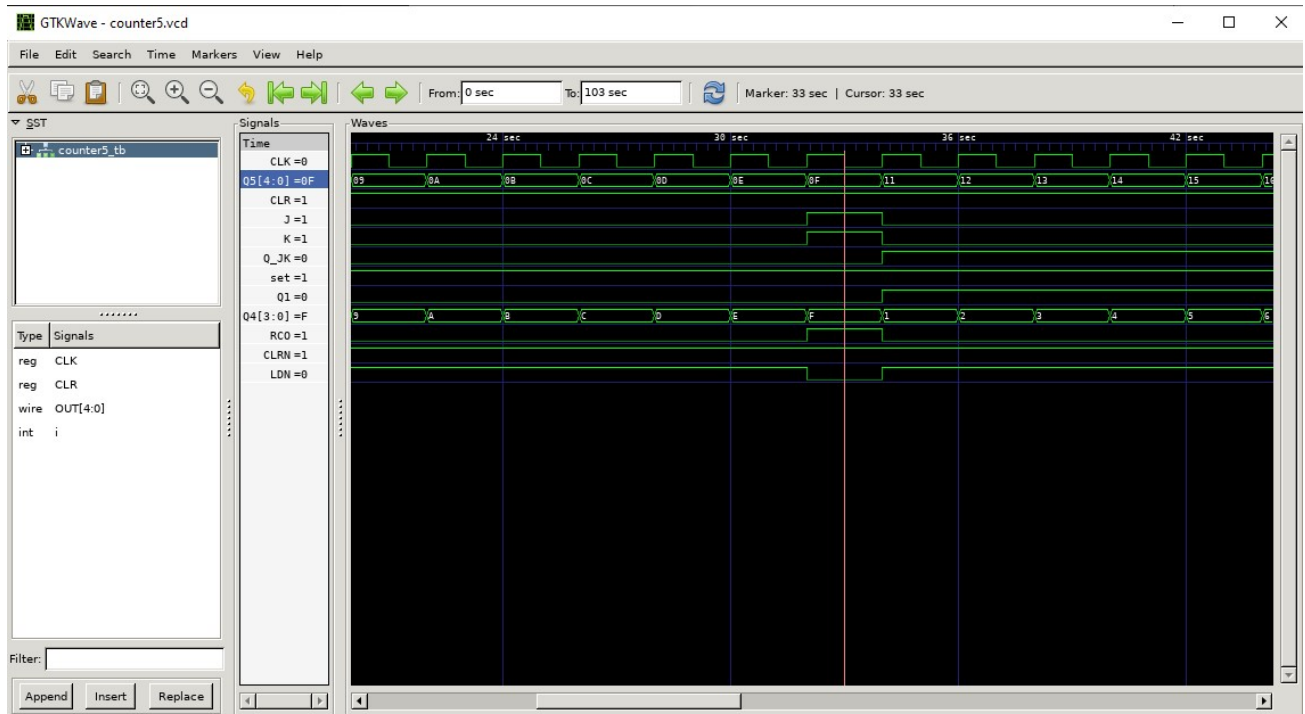


Figure 5: Simulation waveform result, state 15 to state 17

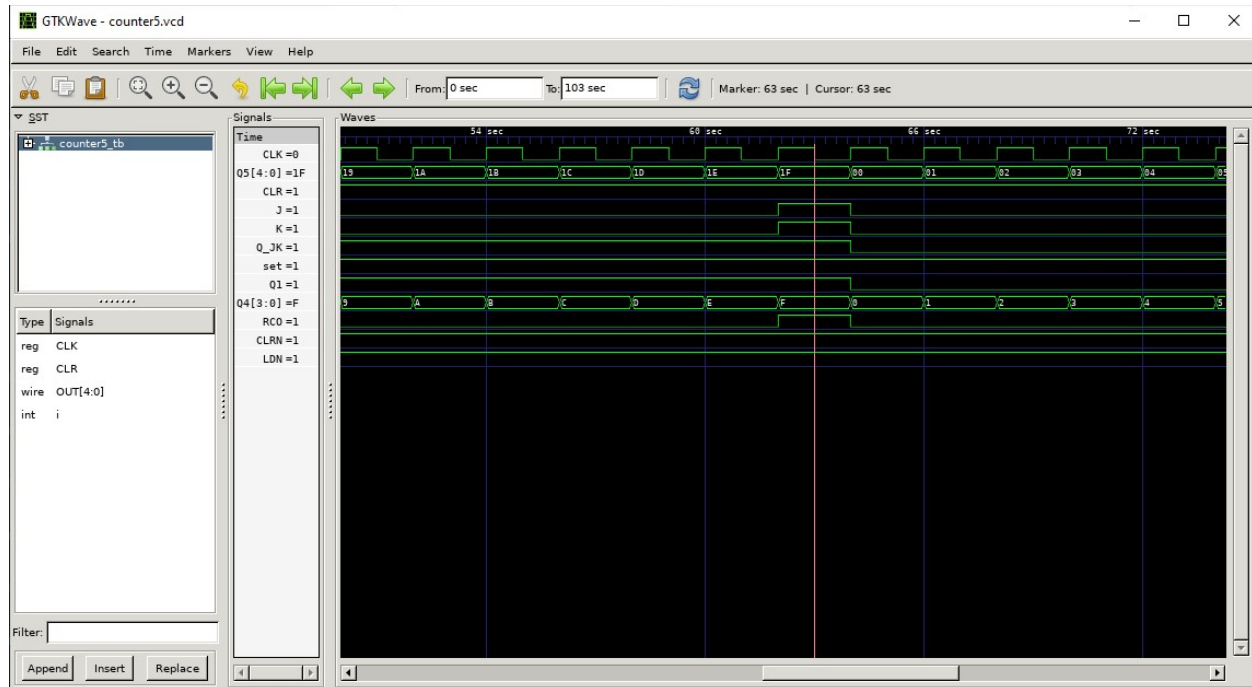


Figure 6: Simulation waveform result, state 31 to state 0

- The JK flip-flop input is the same as RCO. LDN is only low when the output is 0x1F. And in the Figure 5 we can see the state 16, 0x10, is skipped. As presented in the Verilog output file, the counter works properly

Complex 5-Bit Up/Down Counter

- Students need to use memory in this section to implement the 5-bit counter that skip all the prime numbers and read the non-prime number address in the memory file. There is no constrain in this section. With the conditional statements, it is easier or students to implement this function.
- The memory in this section needs to be treated carefully. In the memory file, all the prime numbers are set equal to themselves. Once the counter find that the memory is equal to its address, it will skip this state continue to next state. Here 2 and 3 need to be cared since they are continuous prime number. So the counter will check it again if the new state is still a prime number. 0 will also be treated specially, in the code it will be checked if it is 0, because we do no jump 0.
- The non-prime address in the memory file will store the largest prime number that evenly goes into the number address.
- A conditional statement is used to determine whether count down or count up.
- The Verilog verification are as followed

Verilog Code:

```
module counter_5(input CLK, input ORDER, input CLR, output reg[7:0] OUT);
    reg [7:0] ram [0:31];
    reg [4:0] Q;
    integer timer;
    integer i;
    initial begin
        $readmemh("initram.mem", ram);
    end
    always @(posedge CLK) begin
```

```

if(CLR) begin
    Q <= 5'b00000;
    OUT <= 8'b0000_0000;
    timer <= 0;
end
else if(timer > 10) begin
    if(ORDER) begin
        Q = Q + 1'b1;
        // Compared address and content for prime numbers.
        if(Q == ram[Q] && Q != 0) begin
            Q = Q + 1'b1; // Plus 1 to Q.
            // Special case to skip 2 and 3 together.
            if(Q == ram[Q] && Q != 0) begin
                Q = Q + 1'b1;
            end
            OUT = Q;
        end
        // When Q is not a prime number.
        else begin
            OUT = Q;
        end
    end
    else begin
        Q = Q - 1'b1; // Minus 1 to Q.
        // Detect prime numbers.
        if(Q == ram[Q] && Q != 0) begin
            Q = Q - 1'b1;
            // Skip 2 and 3 together.
            if(Q == ram[Q] && Q != 0) begin
                Q = Q - 1'b1;
            end
            OUT = Q;
        end
        else begin
            OUT = Q;
        end
    end
    end
    //$display("%2d, %2d, %2d", Q,ram[Q],OUT);
    timer = 0;
    $display("Count: %2d, OUT: %2d, Address: %2d", Q,ram[Q],OUT);
end
else begin
    timer = timer + 1;
end
end
end

```

```
endmodule
```

Test branch code:

```
module counter_tb;
    reg CLK;
    reg CLR;
    reg ORDER;
    wire [7:0] OUT;
    integer i;

    counter_5 COUNTER_5(.CLK(CLK), .ORDER(ORDER), .CLR(CLR), .OUT(OUT));

    initial begin
        $dumpfile("P2.vcd");
        $dumpvars;
        $display("GO");

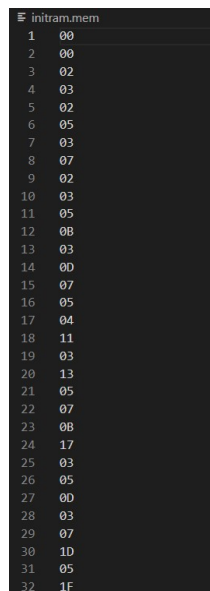
        CLR = 1;
        CLK = 0;
        #1;
        CLK = 1;
        #1;
        CLR = 0;
        CLK = 0;
        #1;

        ORDER = 1;

        for(i = 0; i < 500; i++) begin
            CLK = 0;
            #1;
            CLK = 1;
            #1;
        end
        $display("REVERSE");
        ORDER = 0;
        for(i = 0; i < 500; i++) begin
            CLK = 0;
            #1;
            CLK = 1;
            #1;
        end
    end
end
```

```
end  
  
endmodule
```

Memory file:



Address	Value
1	00
2	00
3	02
4	03
5	02
6	05
7	03
8	07
9	02
10	03
11	05
12	08
13	03
14	00
15	07
16	05
17	04
18	11
19	03
20	13
21	05
22	07
23	08
24	17
25	03
26	05
27	00
28	03
29	07
30	10
31	05
32	1F

- All number in memory file is in hex form.
- The display function is not in the test branch but in the Verilog code. It is necessary to check the address number with it content. So only in the code can display this in the Verilog file.
- **Simulation results:**

```
Select wangy62@DESKTOP-OBTD290: /mnt/c/Users/wangy62/Lab2-2
wangy62@DESKTOP-OBTD290: /mnt/c/Users/wangy62/Lab2-2$ iverilog -o complexCounter P2_tb.v P2_Quartus.v
wangy62@DESKTOP-OBTD290: /mnt/c/Users/wangy62/Lab2-2$ vvp complexCounter
VCD info: dumpfile P2.vcd opened for output.
GO
Count: 1, OUT: 0, Address: 1
Count: 4, OUT: 2, Address: 4
Count: 6, OUT: 3, Address: 6
Count: 8, OUT: 2, Address: 8
Count: 9, OUT: 3, Address: 9
Count: 10, OUT: 5, Address: 10
Count: 12, OUT: 3, Address: 12
Count: 14, OUT: 7, Address: 14
Count: 15, OUT: 5, Address: 15
Count: 16, OUT: 4, Address: 16
Count: 18, OUT: 3, Address: 18
Count: 20, OUT: 5, Address: 20
Count: 21, OUT: 7, Address: 21
Count: 22, OUT: 11, Address: 22
Count: 24, OUT: 3, Address: 24
Count: 25, OUT: 5, Address: 25
Count: 26, OUT: 13, Address: 26
Count: 27, OUT: 3, Address: 27
Count: 28, OUT: 7, Address: 28
Count: 30, OUT: 5, Address: 30
Count: 0, OUT: 0, Address: 0
Count: 1, OUT: 0, Address: 1
Count: 4, OUT: 2, Address: 4
Count: 6, OUT: 3, Address: 6
Count: 8, OUT: 2, Address: 8
Count: 9, OUT: 3, Address: 9
Count: 10, OUT: 5, Address: 10
Count: 12, OUT: 3, Address: 12
Count: 14, OUT: 7, Address: 14
Count: 15, OUT: 5, Address: 15
Count: 16, OUT: 4, Address: 16
Count: 18, OUT: 3, Address: 18
Count: 20, OUT: 5, Address: 20
Count: 21, OUT: 7, Address: 21
Count: 22, OUT: 11, Address: 22
Count: 24, OUT: 3, Address: 24
Count: 25, OUT: 5, Address: 25
Count: 26, OUT: 13, Address: 26
Count: 27, OUT: 3, Address: 27
```

Figure 7: Complex counter simulation 1

```
Select wangy62@DESKTOP-OBTD290: /mnt/c/Users/wangy62/Lab2-2
Count: 21, OUT: 7, Address: 21
Count: 22, OUT: 11, Address: 22
Count: 24, OUT: 3, Address: 24
Count: 25, OUT: 5, Address: 25
Count: 26, OUT: 13, Address: 26
Count: 27, OUT: 3, Address: 27
Count: 28, OUT: 7, Address: 28
Count: 30, OUT: 5, Address: 30
REVERSE
Count: 28, OUT: 7, Address: 28
Count: 27, OUT: 3, Address: 27
Count: 26, OUT: 13, Address: 26
Count: 25, OUT: 5, Address: 25
Count: 24, OUT: 3, Address: 24
Count: 22, OUT: 11, Address: 22
Count: 21, OUT: 7, Address: 21
Count: 20, OUT: 5, Address: 20
Count: 18, OUT: 3, Address: 18
Count: 16, OUT: 4, Address: 16
Count: 15, OUT: 5, Address: 15
Count: 14, OUT: 7, Address: 14
Count: 12, OUT: 3, Address: 12
Count: 10, OUT: 5, Address: 10
Count: 9, OUT: 3, Address: 9
Count: 8, OUT: 2, Address: 8
Count: 6, OUT: 3, Address: 6
Count: 4, OUT: 2, Address: 4
Count: 1, OUT: 0, Address: 1
Count: 0, OUT: 0, Address: 0
Count: 30, OUT: 5, Address: 30
Count: 28, OUT: 7, Address: 28
Count: 27, OUT: 3, Address: 27
Count: 26, OUT: 13, Address: 26
Count: 25, OUT: 5, Address: 25
Count: 24, OUT: 3, Address: 24
Count: 22, OUT: 11, Address: 22
Count: 21, OUT: 7, Address: 21
Count: 20, OUT: 5, Address: 20
Count: 18, OUT: 3, Address: 18
Count: 16, OUT: 4, Address: 16
Count: 15, OUT: 5, Address: 15
Count: 14, OUT: 7, Address: 14
Count: 12, OUT: 3, Address: 12
```

Figure 7: Complex counter simulation 2

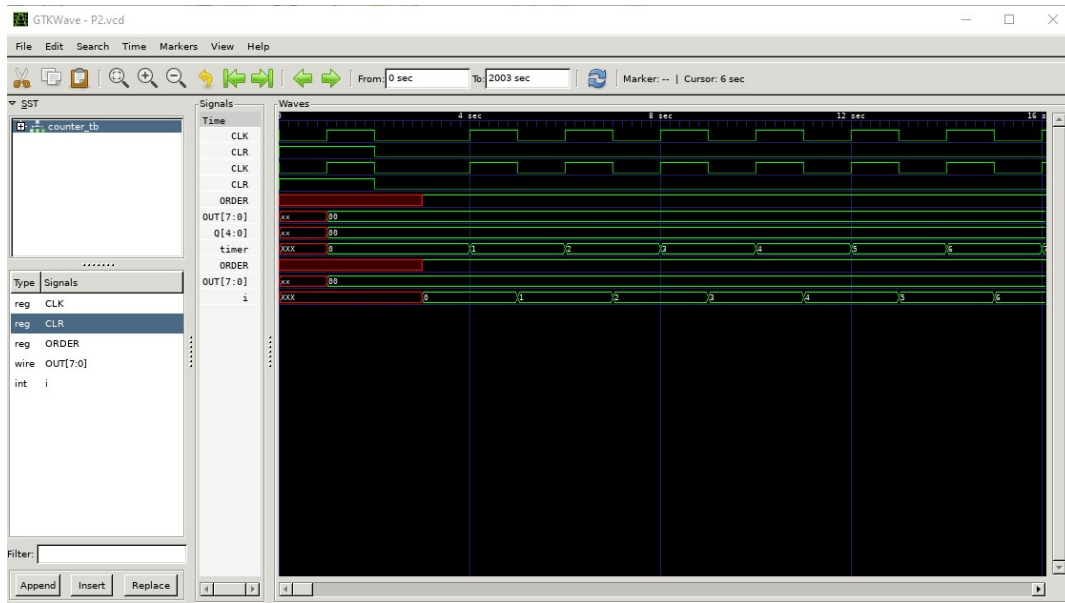


Figure 8: Simulation waveform result, initialization

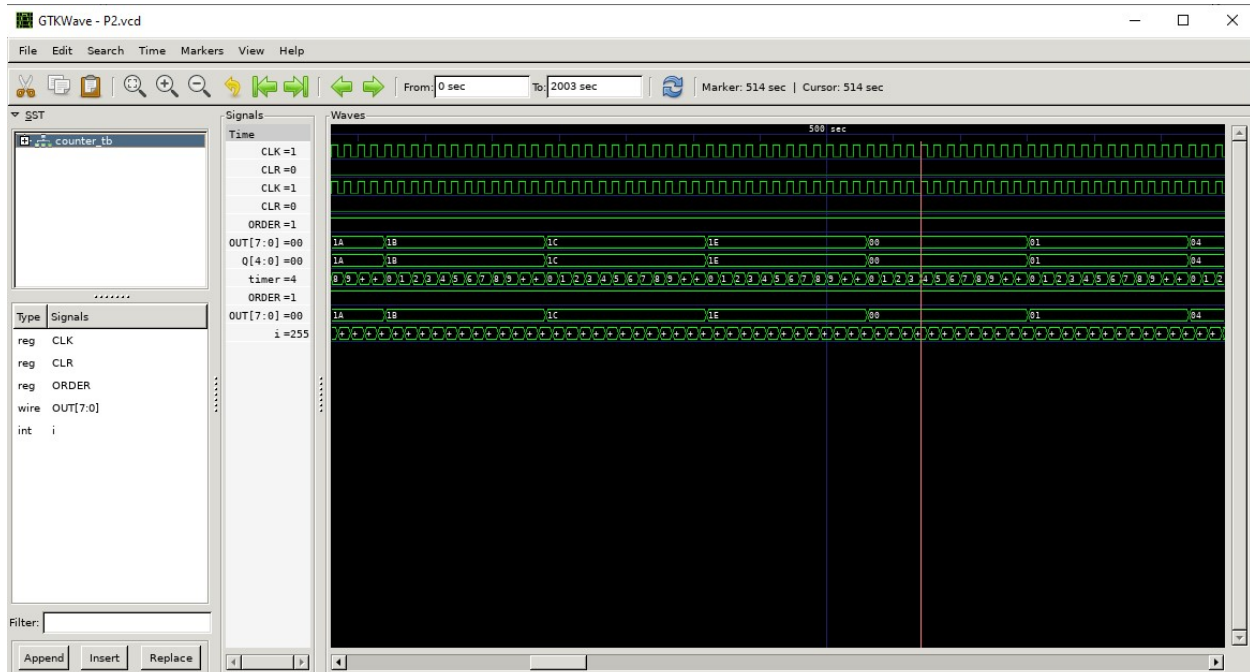


Figure 9: Simulation waveform result, count up

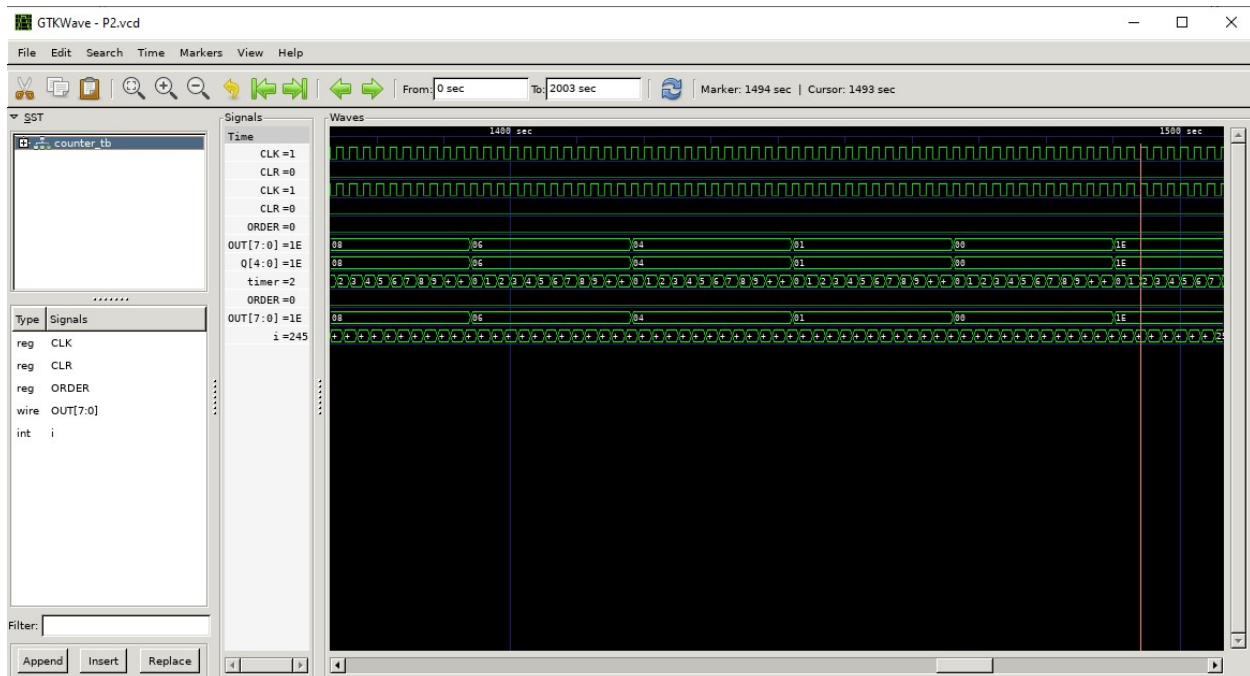


Figure 10: Simulation waveform result, count down

- For the simulation waveform, OUT[7:0] shows the counting output and the address. It skips all the prime numbers as shown in the Verilog output file. As we can see, the timer overflow ten times of the clock rise. Counter counting down and up is also determined by the ORDER

- **Quartus implementation:**

- Since we need to implement both the address and the memory continent, Another Verilog part need to be implemented. Which read the counter output and by checking the memory file, make both address and continent as output. The code are as followed:

```
module omni_ram(input [7:0] ADDR_IN, output reg[7:0] ADDR_OUT, output reg[7:0] RAM_OUT);
    reg [7:0] ram [0:31];
```



```
initial begin
    $readmemh("initram.mem", ram);
end
always @(*) begin
    ADDR_OUT <= ADDR_IN;
    RAM_OUT <= ram[ADDR_IN];
end
endmodule
```

Schematic of the Quartus:

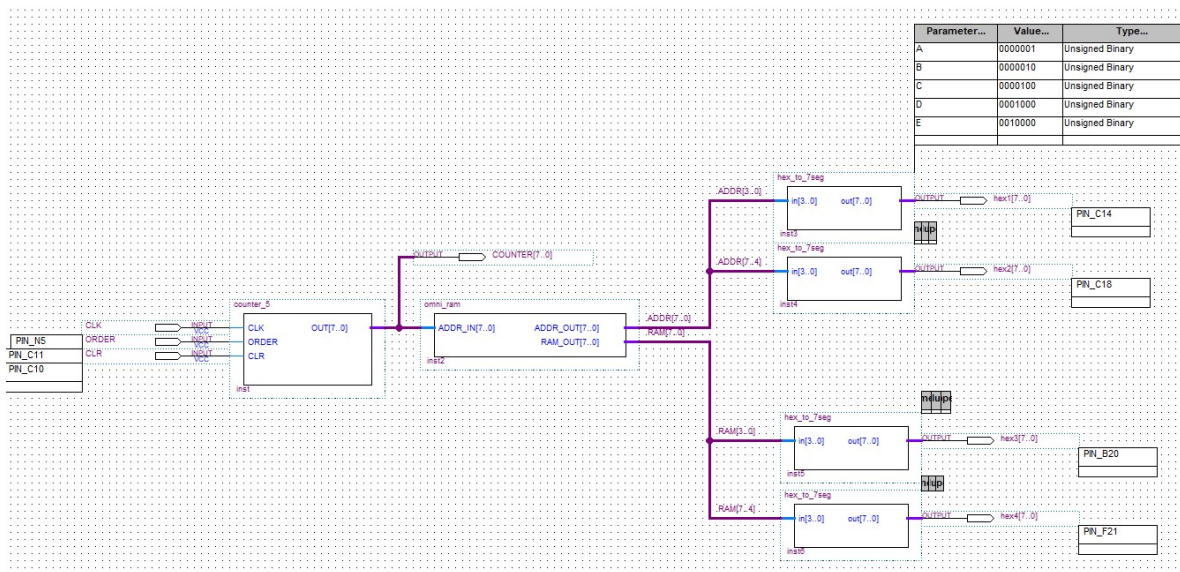


Figure 8: Complex counter schematic

- The CLK is connected to the FPGA board internal clock, PIN_N5. It provide a 10MHz clock, so the code need to be modified to display the count in 1Hz. The overflow time should be changed to 10000000, so that the display time is 1s.
- For simulation in Quartus, timer overflow number is very large, so it needs to be modified again when testing, so student choose to use the Verilog simulation result.

- Simulation result:
 - The simulation result is in a video on YouTube: <https://youtu.be/wAmxhKyu7F0>

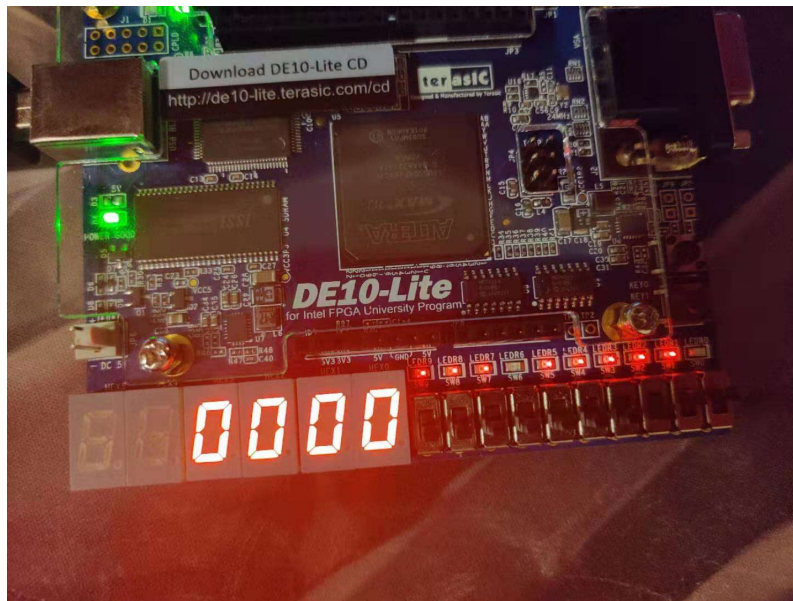


Figure 11: CLR switch clean the counter

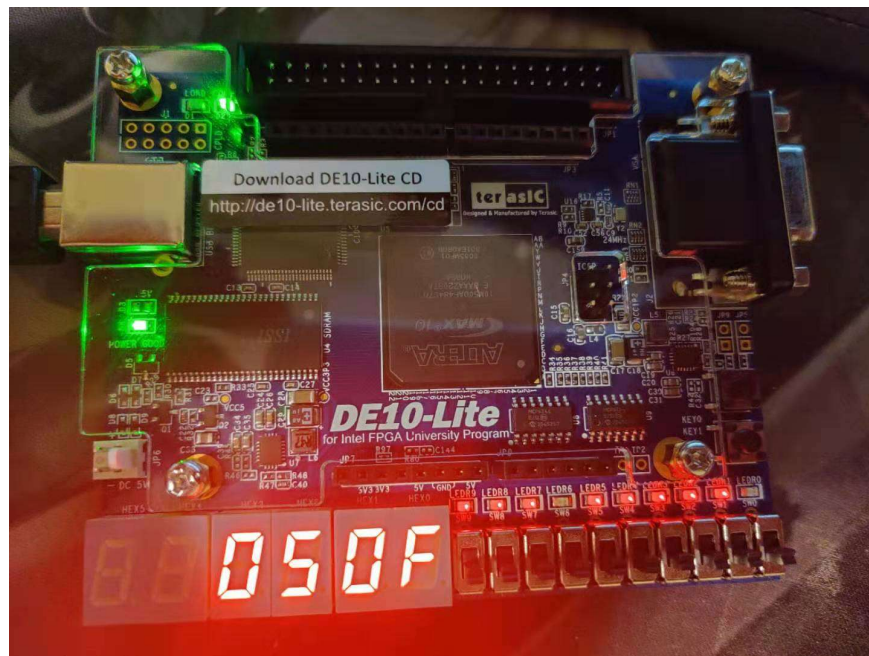


Figure 12: A count up example

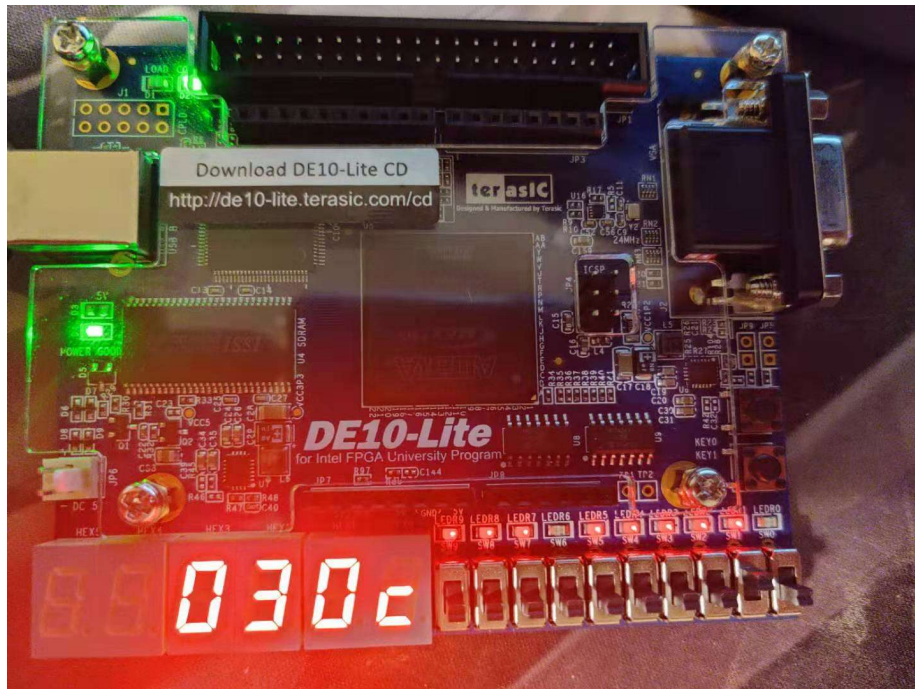


Figure 13: A count down example

- **Verification**
 - Each part of Verilog code is verified by a specialized test branch. By checking the result and waveform students can check if each part has any problem. The final assembled component is also checked in this way.

Conclusion

- The lab result turns out as expected. Students learned how to use memory file and counter. The memory function is useful in implementing many logic components.