“Washing machine controller”

Assignment

Presented in Fulfillment of the Requirements for Mixel ASIC summer internship

Presented by:

Name	Abdulrahman NourEldeen
Email	noureldeenabdulrahman@gmail.com
Mobile	0115 777 9727
University	Cairo University
Graduation Year	2023

Jul. 2022

Contents:

Table of figures:

- Analysis and Explanation:

- It’s required to implement a controller for a washing machine. The machine has 5 states, each takes a specific period. The design will deal with different frequencies so the design should handle them all to achieve the required timing at all the given frequencies. At each frequency, each state of the machine will need specific number of clock cycles to pass before the machine moves to the next state. The following table shows the required number of cycles for each state at each case of different frequencies.

Table 1: Number of clock cycles for each state @ different freq.

State/freq.	1MHZ	2MHZ	4MHZ	8MHZ
Filling water	120M	240M	480M	960M
Washing	300M	600M	1200M	2400M
Rinsing	120M	240M	480M	960M
Spinning	60M	120M	240M	480M

So, A counter is needed to count the clock cycles and raise a flag to tell the machine that the current state is finished.

- From the previous information we can conclude that the design we be implemented as a mealy FSM (finite state machine) with 5 states. The design has “coin_in’’ input flag which tells the design to start the operation when asserted. Another input flag “double_wash”, asserted when the user requires a double wash, and this flag tells the machine to repeat washing and rinsing states then continue the operation. “timer_pause” flag is the last one, used to pause the spinning state till de-assertion and it’s required from the design to ignore this input flag if the machine is not in the spinning state. The design output is an active high flag “wash_done” which is asserted when the machine completes the full washing operation, then the machine goes IDLE till the user puts the coins again then the flag is de-asserted, and the machine start to do the job again.

- The following figure shows a simple state diagram representing the idea:

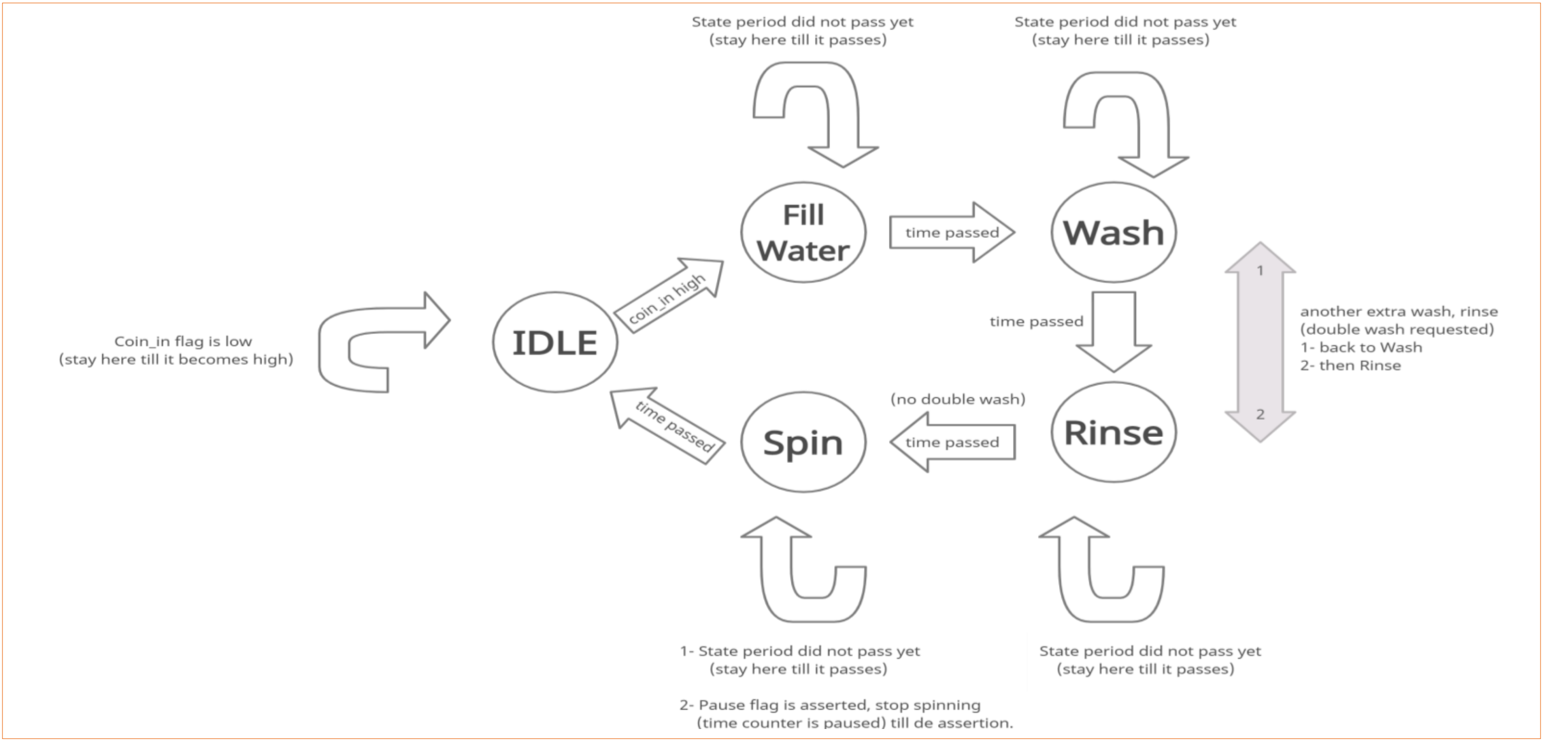

State period did not pass yet
(stay here till it passes)
State period did not pass yet
(stay here till it passes)
Fill Water
time passed
Wash
Coin_in flag is low
(stay here till it becomes high)
IDLE
coin_in high
time passed
1
2
another extra wash, rinse
(double wash requested)
1- back to Wash
2- then Rinse
(no double wash)
time passed
time passed
Spin
Rinse
1- State period did not pass yet
(stay here till it passes)
2- Pause flag is asserted, stop spinning
(time counter is paused) till de assertion.
State period did not pass yet
(stay here till it passes)

Figure 1 simple state diagram

- Solution and Design flow:

- To represent a mealy FSM, we mainly need three essential logic blocks, next state logic, state transition, and output logic.

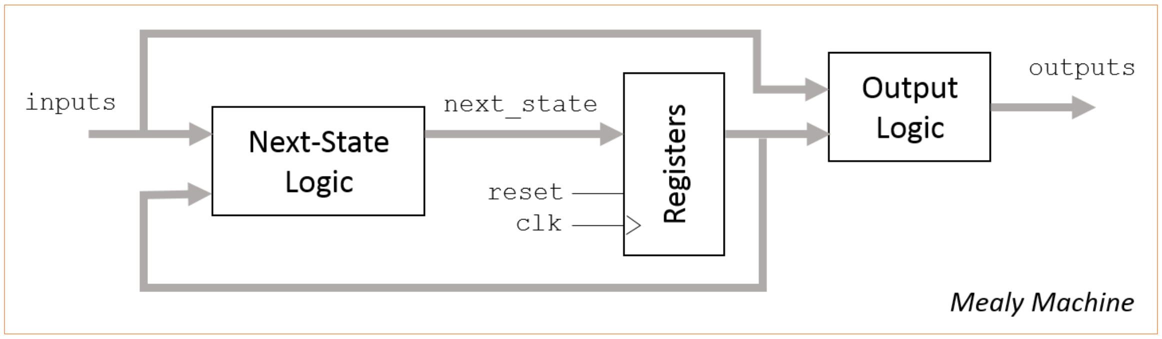


Figure 2 Mealy FSM general block diagram

- As mentioned before, a counter is needed to count the clock cycles to tell the FSM when to switch to the next state. Each state wants from the counter to count a certain number of clocks so at each state the FSM will tell the counter to start counting the required number of clocks and when the counter is done counting, a "count done" flag will be asserted to tell the FSM that the time for the current state is passed. The following figure to clarify the idea:

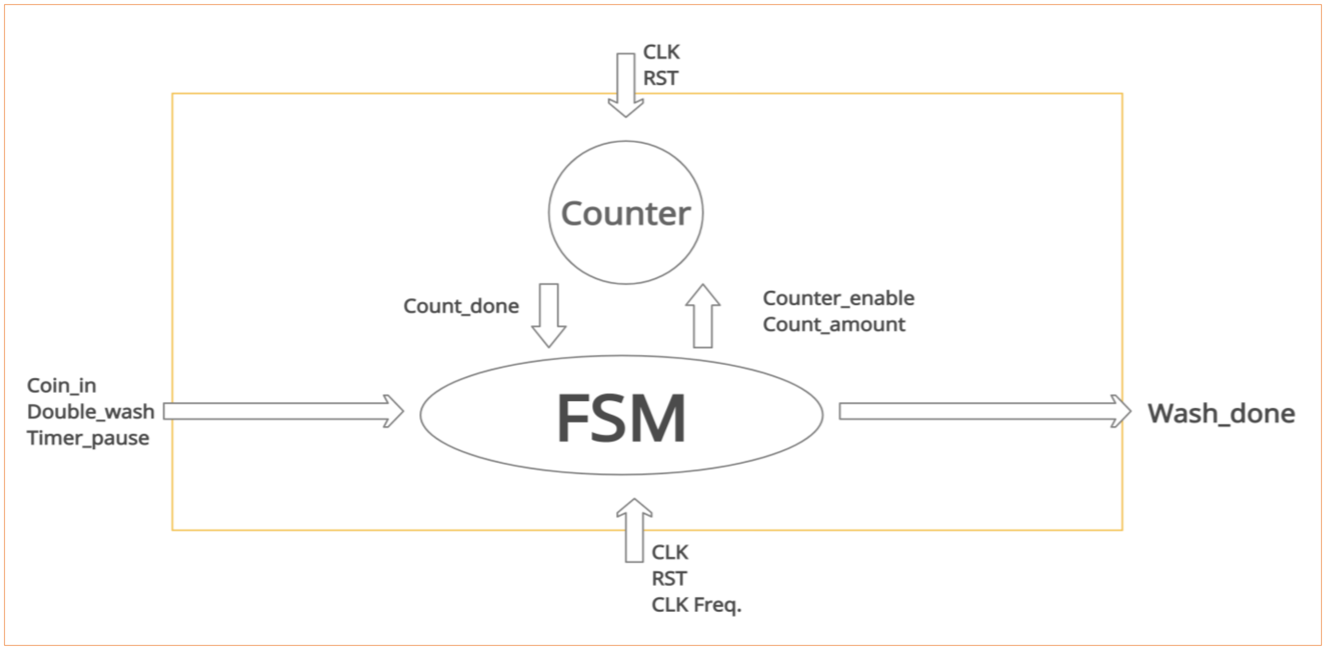
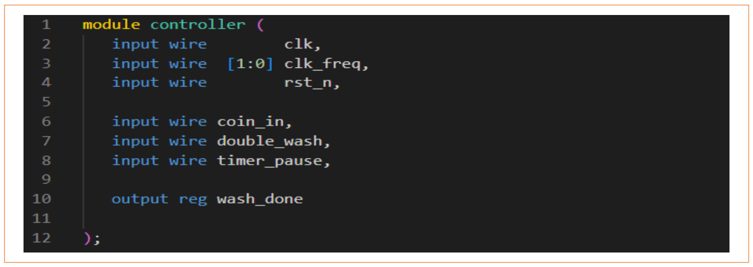


Figure 3 Simple Full design diagram

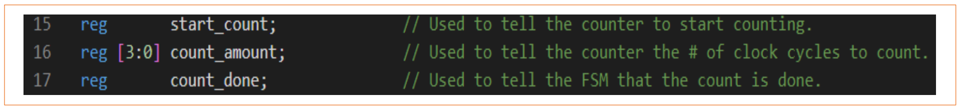
- To handle the double wash input, there will be a condition in the design to check if the "double_wash" flag is asserted, if so, the design will make the FSM repeat the Wash and Rinse states once again and then continue to the next state.
- There will be also a condition to use the "Timer_pause" flag only if the current state is "Spinning".
- The previous conditions will be explained more in the following section "Code explanation".

- Code explanation:
  - **Verilog design code:**



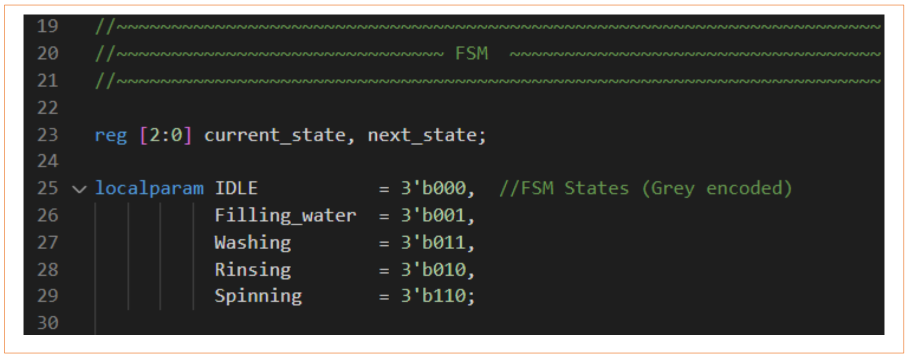
```verilog
module controller (
    input wire        clk,
    input wire  [1:0] clk_freq,
    input wire        rst_n,

    input wire coin_in,
    input wire double_wash,
    input wire timer_pause,

    output reg wash_done

);
```

- The design inputs and outputs.
- The different four clock frequencies are encoded in 2 bits at the input “clk_freq”.



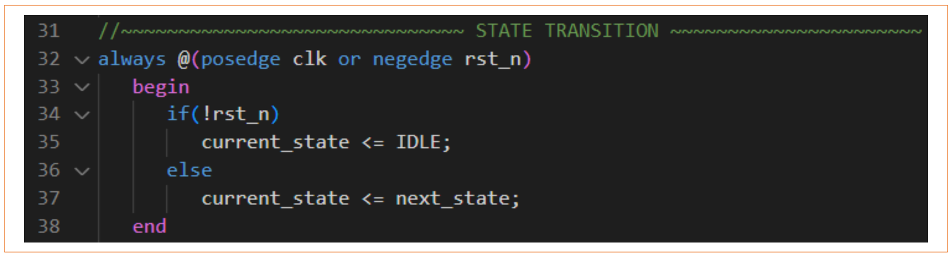
```verilog
reg       start_count;            // Used to tell the counter to start counting.
reg [3:0] count_amount;           // Used to tell the counter the # of clock cycles to count.
reg       count_done;             // Used to tell the FSM that the count is done.
```

- As mentioned before, these variables are used to tell the counter when to start counting and what value to count, and to tell the FSM when the count is done.



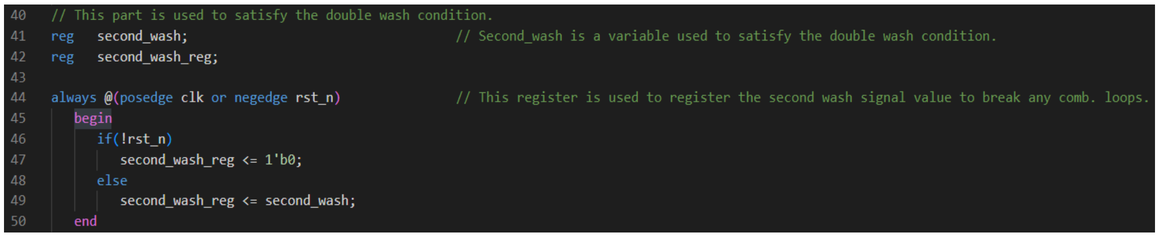
```verilog
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ FSM  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

reg [2:0] current_state, next_state;

localparam IDLE          = 3'b000,  //FSM States (Grey encoded)
           Filling_water = 3'b001,
           Washing       = 3'b011,
           Rinsing       = 3'b010,
           Spinning      = 3'b110;

```

- Local parameters representing the five different states of the machine. I used gray encoding, I found that its better to use in most of the cases.



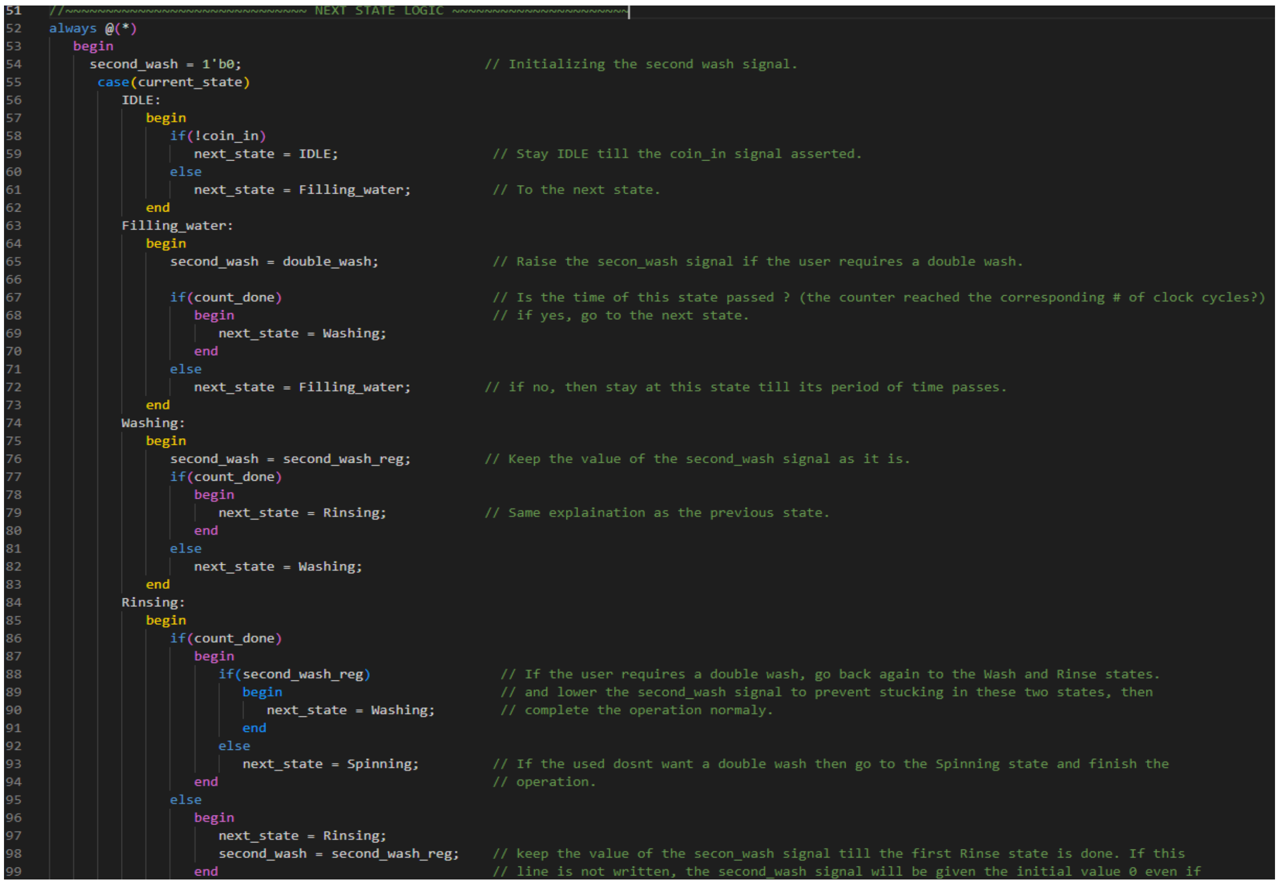
```verilog
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ STATE TRANSITION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
always @(posedge clk or negedge rst_n)
    begin
        if(!rst_n)
            current_state <= IDLE;
        else
            current_state <= next_state;
    end
```

- This piece of code represents the state transition (registers) block mentioned in the previous diagrams. The current machine state goes IDLE when the system resets else the current state goes to the next state.



```verilog
// This part is used to satisfy the double wash condition.
reg   second_wash;                                    // Second_wash is a variable used to satisfy the double wash condition.
reg   second_wash_reg;

always @(posedge clk or negedge rst_n)                // This register is used to register the second wash signal value to break any comb. loops.
   begin
      if(!rst_n)
         second_wash_reg <= 1'b0;
      else
         second_wash_reg <= second_wash;
   end
```

- The above piece of code is used to achieve the double wash condition, “second_wash_reg” is used to store the value of the variable “second_wash”, and if the double wash condition satisfied (second_wash_reg is used in the check condition for double wash) “second_wash” goes low and the condition is satisfied only once. This part will be clarified more in the next piece of code.

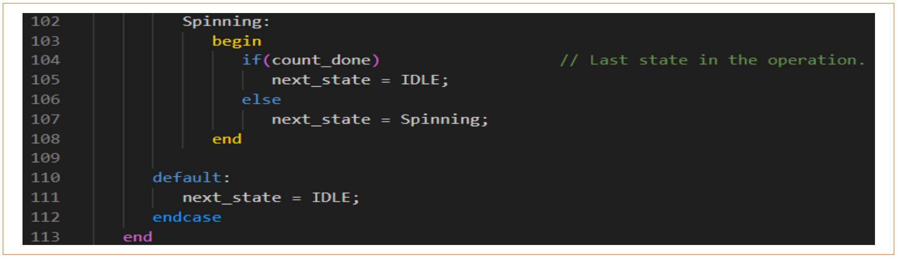


```verilog
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ NEXT STATE LOGIC ~~~~~~~~~~~~~~~~~~~~~~~
always @(*)
   begin
     second_wash = 1'b0;                              // Initializing the second wash signal.
      case(current_state)
         IDLE:
            begin
               if(!coin_in)
                  next_state = IDLE;                   // Stay IDLE till the coin_in signal asserted.
               else
                  next_state = Filling_water;          // To the next state.
            end
         Filling_water:
            begin
               second_wash = double_wash;              // Raise the secon_wash signal if the user requires a double wash.

               if(count_done)                          // Is the time of this state passed ? (the counter reached the corresponding # of clock cycles?)
                  begin                                // if yes, go to the next state.
                     next_state = Washing;
                  end
               else
                  next_state = Filling_water;         // if no, then stay at this state till its period of time passes.
            end
         Washing:
            begin
               second_wash = second_wash_reg;         // Keep the value of the second_wash signal as it is.
               if(count_done)
                  begin
                     next_state = Rinsing;            // Same explaination as the previous state.
                  end
               else
                  next_state = Washing;
            end
         Rinsing:
            begin
               if(count_done)
                  begin
                     if(second_wash_reg)               // If the user requires a double wash, go back again to the Wash and Rinse states.
                        begin                          // and lower the second_wash signal to prevent stucking in these two states, then
                           next_state = Washing;       // complete the operation normaly.
                        end
                     else
                        next_state = Spinning;        // If the used dosnt want a double wash then go to the Spinning state and finish the
                  end                                 // operation.
               else
                  begin
                     next_state = Rinsing;
                     second_wash = second_wash_reg;   // keep the value of the secon_wash signal till the first Rinse state is done. If this
                  end                                 // line is not written, the second_wash signal will be given the initial value 0 even if
```

- The above piece of code represents part of the next state logic of the FSM. “second_wash” is given initial value to avoid comb. loops, the machine stays at IDLE state till the “coin_in” flag is high then it goes to the next state “Filling_water”, at this state, the “second_wash” is given the value of “double_wash” input signal so that if double wash flag is high, the value 1 got registered at the register “double_wash_reg”. the FSM stays at this state till the counter raise the “count_done” flag then the FSM goes to the “Washing” state, here we want to keep the value of “second_wash” as it is so that no looping occurs (if the FSM is at “Rinsing state” and there is a double wash request, the “double_wash” is set to 0 and since the “Washing” state keeps the value of “double_wash” as it is, when the FSM goes to “Rinsing” again the double wash condition won’t be achieved).
- As mentioned, the FSM will wait till the counter raise “count_done” and goes to the following state.

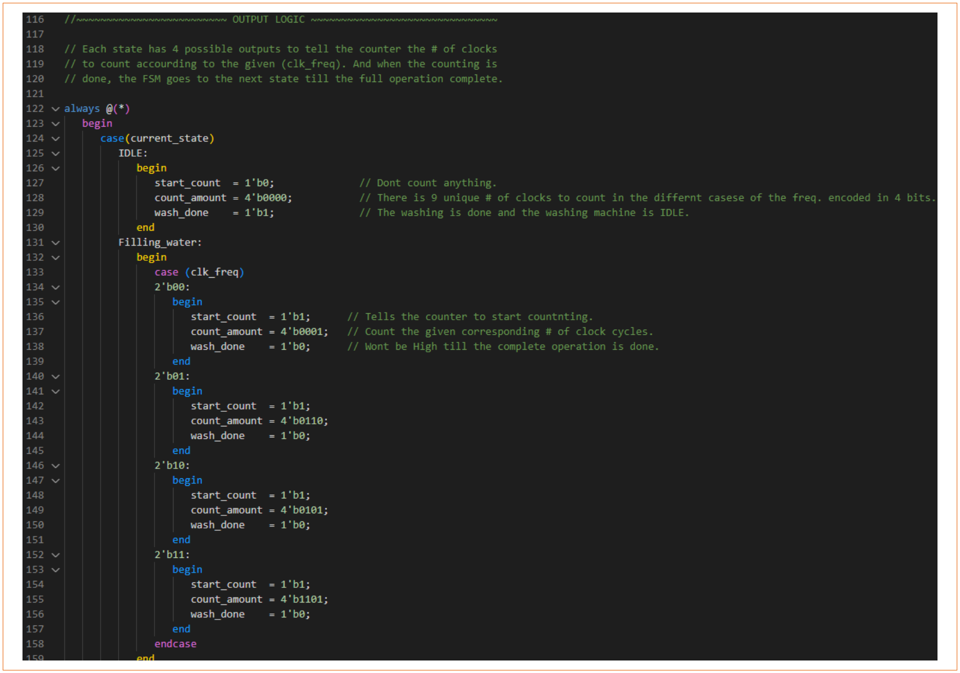
	double_wash = 1	double_wash = 0
State	second_wash =	second_wash =
Idle	0	0
Filling water	1	0
Washing	1	0
Rinsing	Condition satisfied and second_wash set to 0	Condition won't be satisfied
Second wash	0	▨
Second Rinse	0	▨
Spinning	0	0

- The above table gives more explanation for the idea by tracing "second_wash" value.



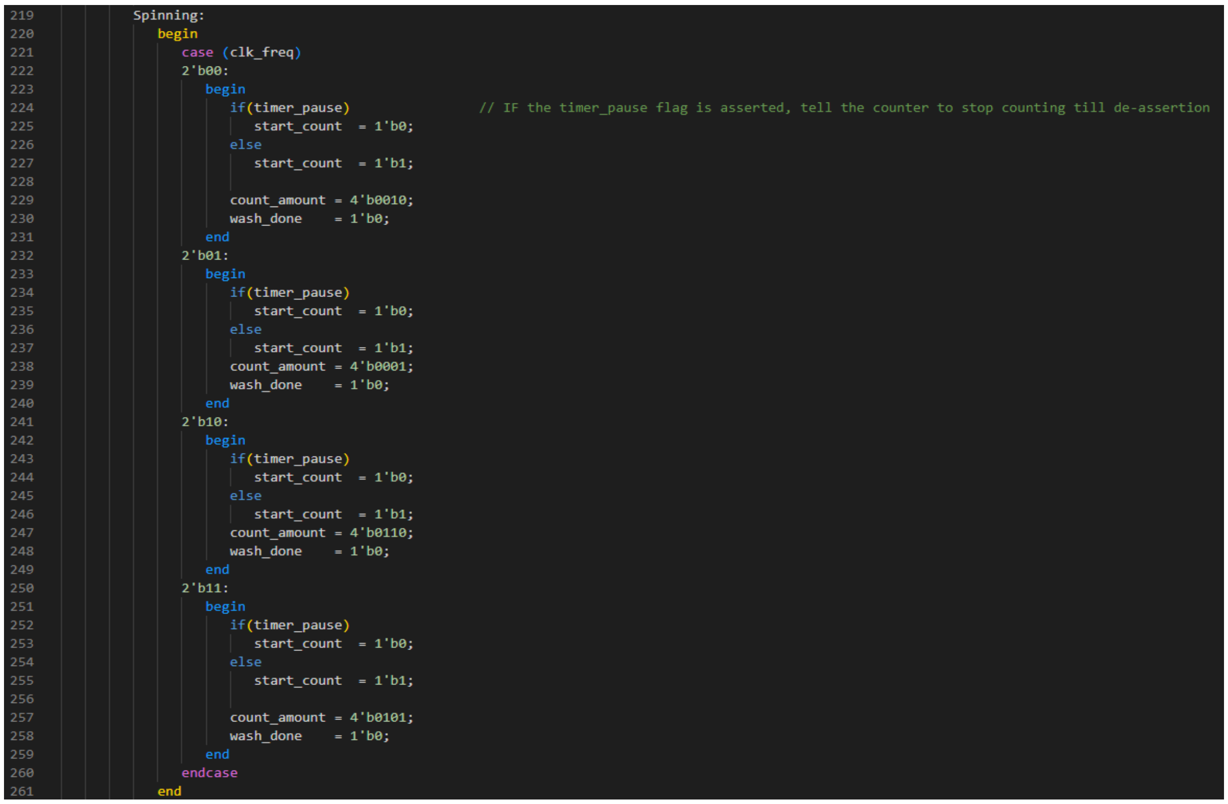
```
            Spinning:
                begin
                    if(count_done)                  // Last state in the operation.
                        next_state = IDLE;
                    else
                        next_state = Spinning;
                end

        default:
            next_state = IDLE;
        endcase
    end
```

- After all the above scenarios, the FSM goes to the "Spinning" state the goes IDLE.



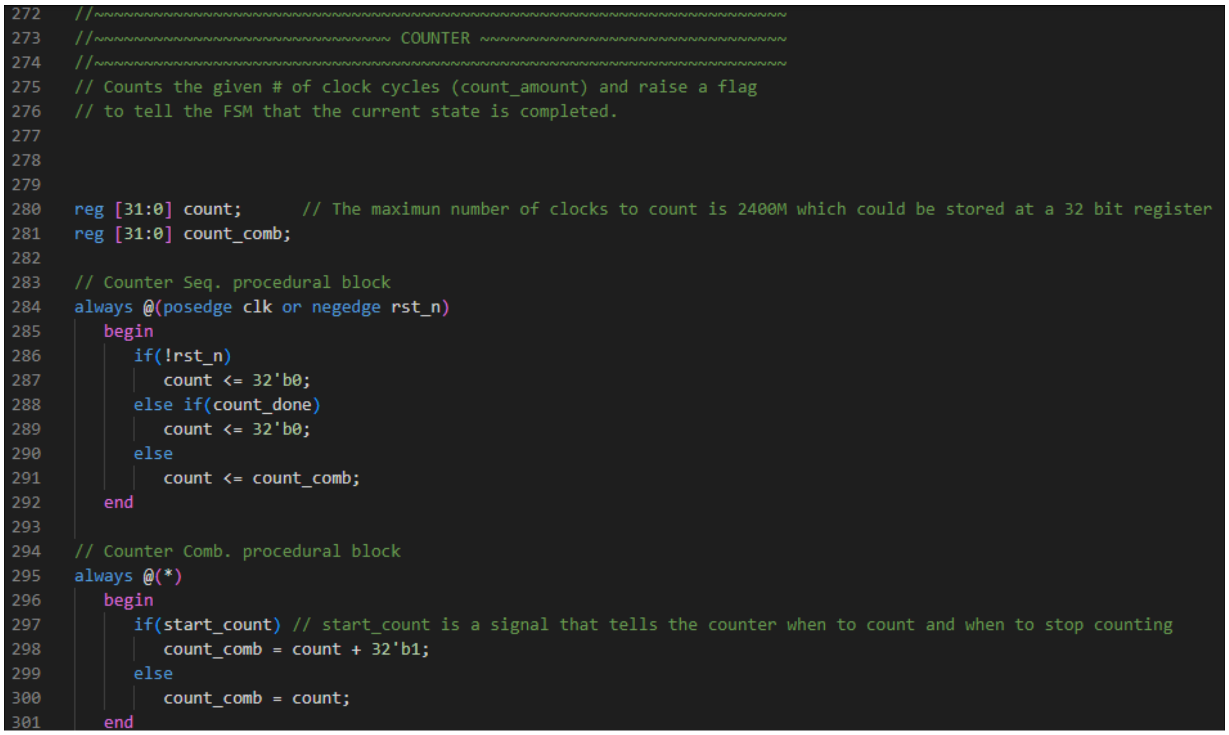
```
//~~~~~~~~~~~~~~~~~~~~~~~~~ OUTPUT LOGIC ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// Each state has 4 possible outputs to tell the counter the # of clocks
// to count accourding to the given (clk_freq). And when the counting is
// done, the FSM goes to the next state till the full operation complete.

always @(*)
    begin
        case(current_state)
            IDLE:
                begin
                    start_count  = 1'b0;            // Dont count anything.
                    count_amount = 4'b0000;         // There is 9 unique # of clocks to count in the differnt casese of the freq. encoded in 4 bits.
                    wash_done    = 1'b1;            // The washing is done and the washing machine is IDLE.
                end
            Filling_water:
                begin
                    case (clk_freq)
                    2'b00:
                        begin
                            start_count  = 1'b1;     // Tells the counter to start countnting.
                            count_amount = 4'b0001;  // Count the given corresponding # of clock cycles.
                            wash_done    = 1'b0;     // Wont be High till the complete operation is done.
                        end
                    2'b01:
                        begin
                            start_count  = 1'b1;
                            count_amount = 4'b0110;
                            wash_done    = 1'b0;
                        end
                    2'b10:
                        begin
                            start_count  = 1'b1;
                            count_amount = 4'b0101;
                            wash_done    = 1'b0;
                        end
                    2'b11:
                        begin
                            start_count  = 1'b1;
                            count_amount = 4'b1101;
                            wash_done    = 1'b0;
                        end
                    endcase
                end
```

- The above piece of code represents a part of the Output logic mentioned in the previous mealy FSM diagram. At IDLE the counter is not counting since the "start_count" signal is low and "wash_done" is high as the machine is idling. At "Filling_water, Washing and Rinsing" states, the logic tells the counter to start counting a given number of clocks, this number is determined according to the operating given frequency.



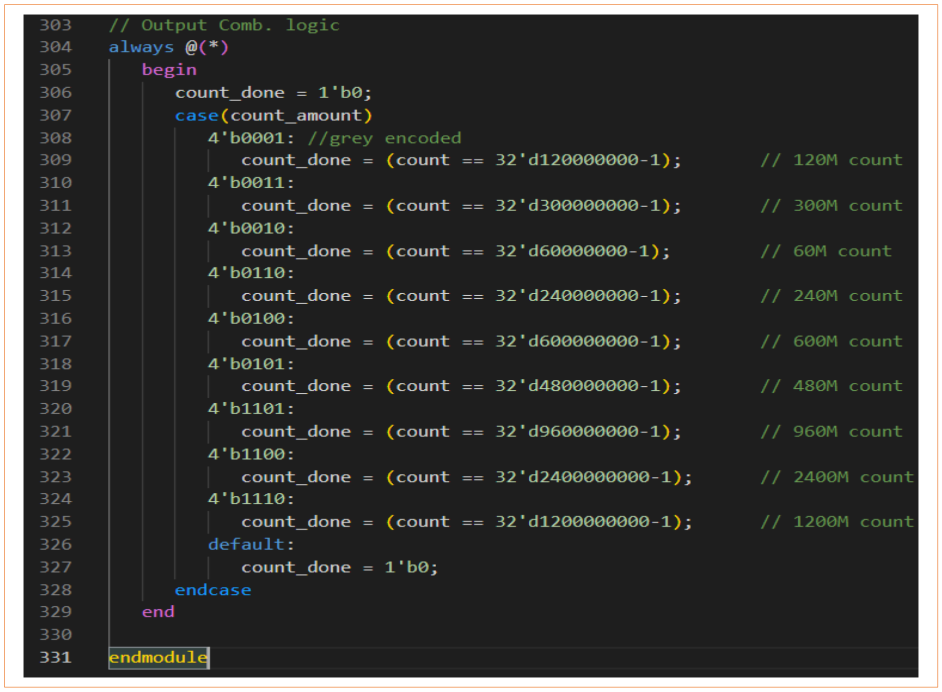
```verilog
Spinning:
begin
    case (clk_freq)
    2'b00:
        begin
            if(timer_pause)                 // IF the timer_pause flag is asserted, tell the counter to stop counting till de-assertion
                start_count  = 1'b0;
            else
                start_count  = 1'b1;

            count_amount = 4'b0010;
            wash_done    = 1'b0;
        end
    2'b01:
        begin
            if(timer_pause)
                start_count  = 1'b0;
            else
                start_count  = 1'b1;
            count_amount = 4'b0001;
            wash_done    = 1'b0;
        end
    2'b10:
        begin
            if(timer_pause)
                start_count  = 1'b0;
            else
                start_count  = 1'b1;
            count_amount = 4'b0110;
            wash_done    = 1'b0;
        end
    2'b11:
        begin
            if(timer_pause)
                start_count  = 1'b0;
            else
                start_count  = 1'b1;

            count_amount = 4'b0101;
            wash_done    = 1'b0;
        end
    endcase
end
```

- At "Spinning" state the logic is different since its affected by the "timer_pause" input flag, the "start_count" signal set to low when this input flag is raised so that the counter stops counting till this flag is de-asserted.



```verilog
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ COUNTER ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Counts the given # of clock cycles (count_amount) and raise a flag
// to tell the FSM that the current state is completed.



reg [31:0] count;      // The maximun number of clocks to count is 2400M which could be stored at a 32 bit register
reg [31:0] count_comb;

// Counter Seq. procedural block
always @(posedge clk or negedge rst_n)
   begin
      if(!rst_n)
         count <= 32'b0;
      else if(count_done)
         count <= 32'b0;
      else
         count <= count_comb;
   end

// Counter Comb. procedural block
always @(*)
   begin
      if(start_count) // start_count is a signal that tells the counter when to count and when to stop counting
         count_comb = count + 32'b1;
      else
         count_comb = count;
   end
```

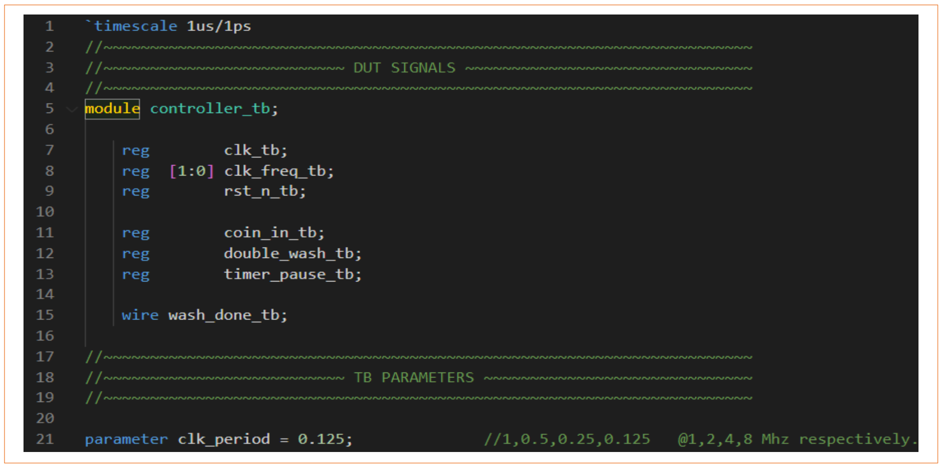
- The above piece of code represents first part of the counter used in the design. Normal counter that counts the clocks and raise a flag when done counting. Sequential and combinational logic are separated.



```verilog
// Output Comb. logic
always @(*)
    begin
        count_done = 1'b0;
        case(count_amount)
            4'b0001: //grey encoded
                count_done = (count == 32'd120000000-1);        // 120M count
            4'b0011:
                count_done = (count == 32'd300000000-1);        // 300M count
            4'b0010:
                count_done = (count == 32'd60000000-1);         // 60M count
            4'b0110:
                count_done = (count == 32'd240000000-1);        // 240M count
            4'b0100:
                count_done = (count == 32'd600000000-1);        // 600M count
            4'b0101:
                count_done = (count == 32'd480000000-1);        // 480M count
            4'b1101:
                count_done = (count == 32'd960000000-1);        // 960M count
            4'b1100:
                count_done = (count == 32'd2400000000-1);       // 2400M count
            4'b1110:
                count_done = (count == 32'd1200000000-1);       // 1200M count
            default:
                count_done = 1'b0;
        endcase
    end

endmodule
```

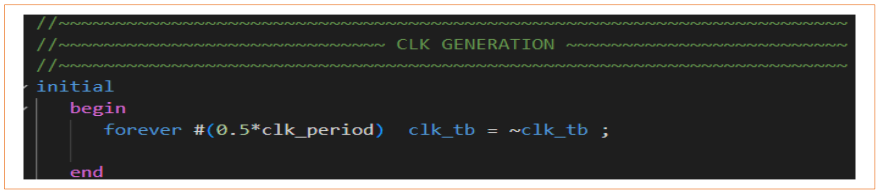
- The above piece of code represents the second part of the counter. “count_done” flag is asserted when the counter reaches the required amount of clock encoded in “count_amount”. -1 because the counter starts counting from 0.

▪ Verilog Testbench code:



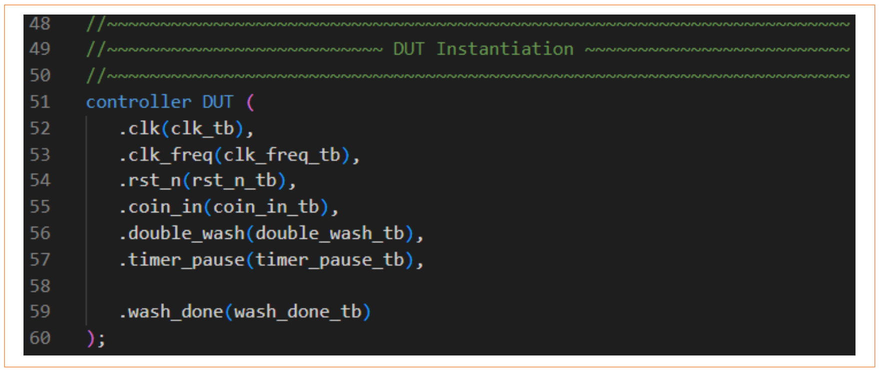
```verilog
`timescale 1us/1ps
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DUT SIGNALS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
module controller_tb;

    reg         clk_tb;
    reg   [1:0] clk_freq_tb;
    reg         rst_n_tb;

    reg         coin_in_tb;
    reg         double_wash_tb;
    reg         timer_pause_tb;

    wire wash_done_tb;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~ TB PARAMETERS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

parameter clk_period = 0.125;              //1,0.5,0.25,0.125   @1,2,4,8 Mhz respectively.
```

- The above piece of code represents the DUT “design under test” signals and the time scale which is set to one microsecond and the test bench used parameters.

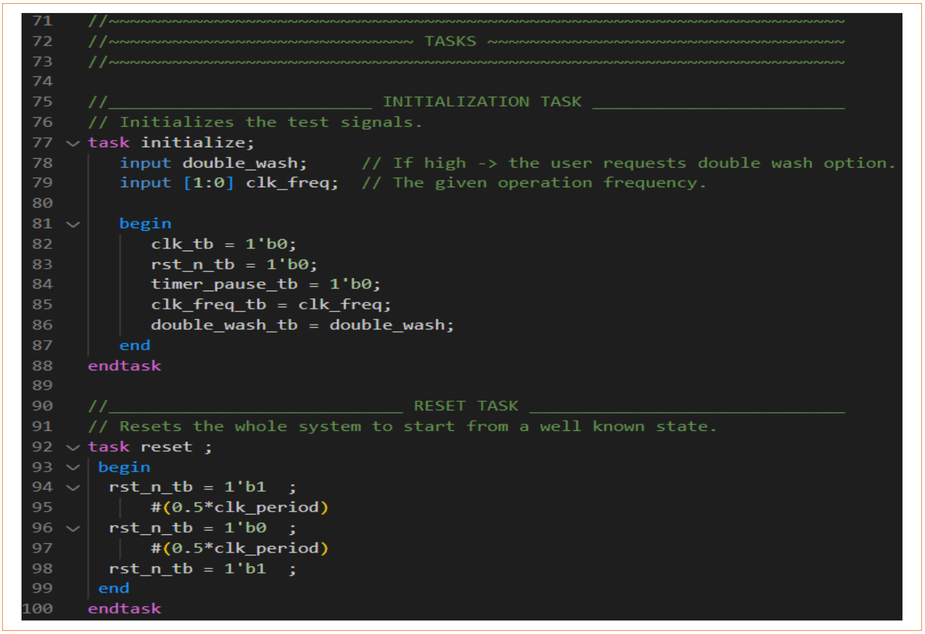


```verilog
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ CLK GENERATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
    begin
        forever #(0.5*clk_period)  clk_tb = ~clk_tb ;

    end
```

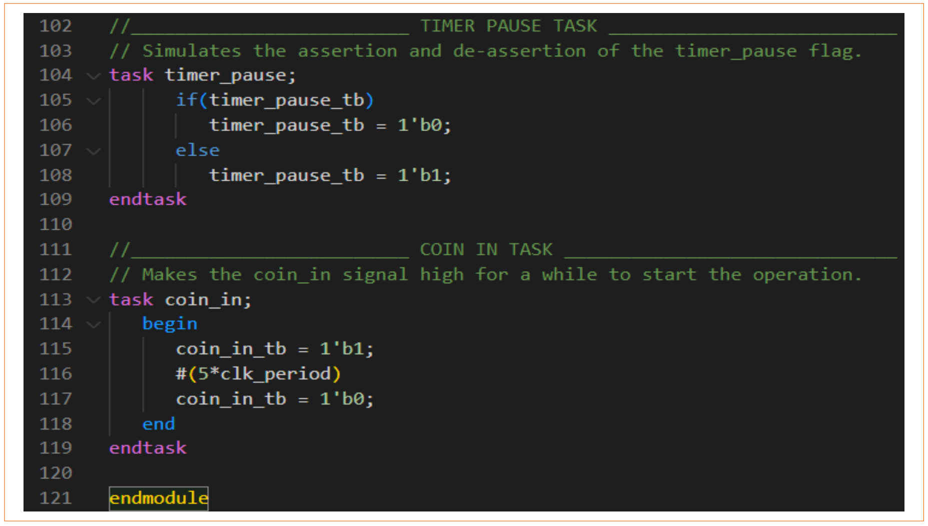
- The above piece of code represents the clock generation logic. Toggle the clk_tb value each half cycle.


```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ DUT Instantiation ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
controller DUT (
    .clk(clk_tb),
    .clk_freq(clk_freq_tb),
    .rst_n(rst_n_tb),
    .coin_in(coin_in_tb),
    .double_wash(double_wash_tb),
    .timer_pause(timer_pause_tb),

    .wash_done(wash_done_tb)
);
```

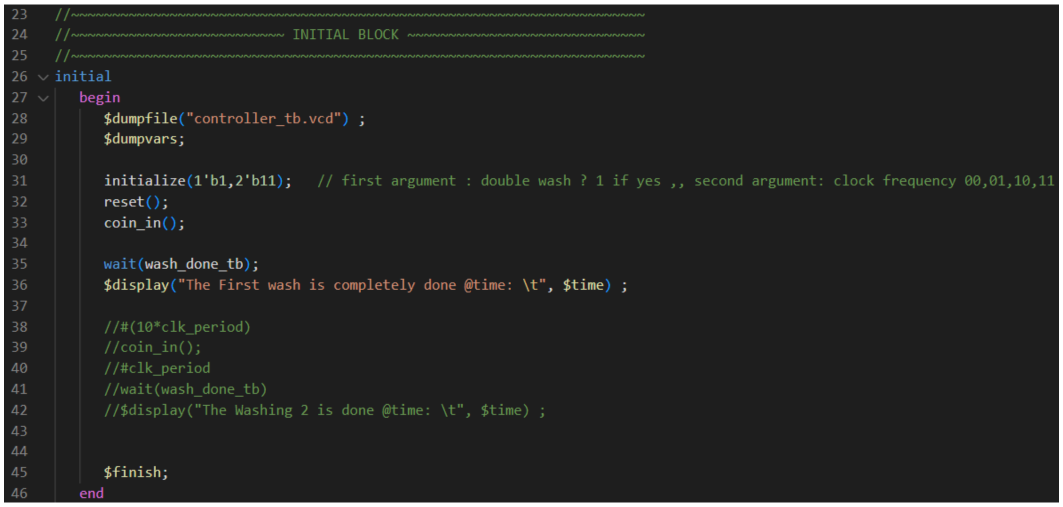
- DUT instantiation.


```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ TASKS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//_________________________ INITIALIZATION TASK ________________________
// Initializes the test signals.
task initialize;
   input double_wash;     // If high -> the user requests double wash option.
   input [1:0] clk_freq;  // The given operation frequency.

   begin
      clk_tb = 1'b0;
      rst_n_tb = 1'b0;
      timer_pause_tb = 1'b0;
      clk_freq_tb = clk_freq;
      double_wash_tb = double_wash;
   end
endtask

//____________________________ RESET TASK ______________________________
// Resets the whole system to start from a well known state.
task reset ;
 begin
  rst_n_tb = 1'b1   ;
      #(0.5*clk_period)
  rst_n_tb = 1'b0   ;
      #(0.5*clk_period)
  rst_n_tb = 1'b1   ;
 end
endtask
```

- The above piece of code represents some of the used tasks. “Reset” task resets the design, “Initialize” task initializes the test signals with initial values.


```
//___________________________ TIMER PAUSE TASK _________________________
// Simulates the assertion and de-assertion of the timer_pause flag.
task timer_pause;
      if(timer_pause_tb)
         timer_pause_tb = 1'b0;
      else
         timer_pause_tb = 1'b1;
endtask

//___________________________ COIN IN TASK _____________________________
// Makes the coin_in signal high for a while to start the operation.
task coin_in;
   begin
      coin_in_tb = 1'b1;
      #(5*clk_period)
      coin_in_tb = 1'b0;
   end
endtask

endmodule
```

- The rest of the tasks, “timer_pause” when used the “timer_pause_tb” signal is asserted if de-asserted and vice versa, “coin_in” used to set the “coin_in_tb” signal high so the design start to do the job.



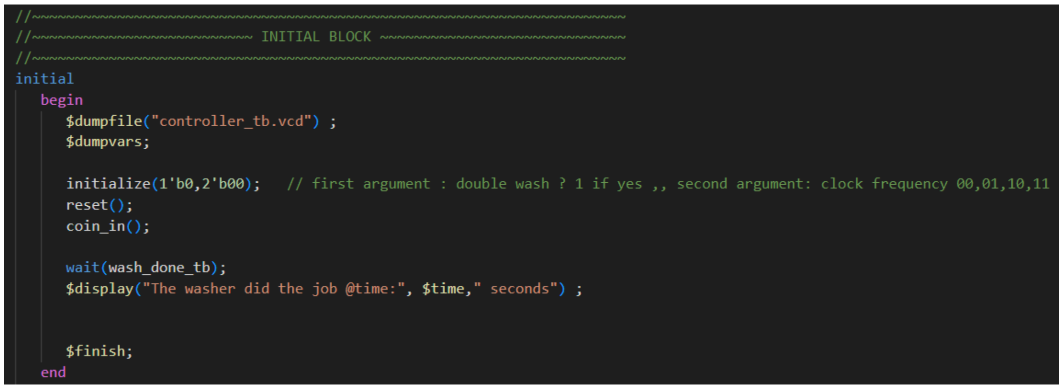
```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b1,2'b11);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      wait(wash_done_tb);
      $display("The First wash is completely done @time: \t", $time) ;

      //#(10*clk_period)
      //coin_in();
      //#clk_period
      //wait(wash_done_tb)
      //$display("The Washing 2 is done @time: \t", $time) ;


      $finish;
   end
```

- The above piece of code is the Initial procedural block, this is just a simple case, more complicated scenarios will be presented in the following section “Testing covered scenarios”. “$dumpfile” and “$dumpvars” used to save the results of the test. Initialize and reset tasks as mentioned before used to reset and initialize the test signals. More details in the following section.

• Testing covered scenarios:

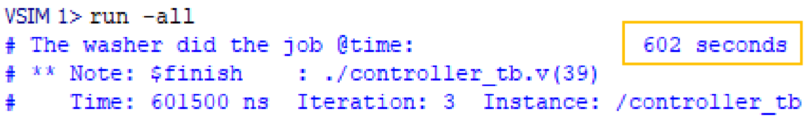
Note: to speed up the simulations I will make the counter counts for example (120 instead of 120M) to save time, and the time unit of the output will represent the seconds in real time (1 $\mu \sec in\ simulation == 1 \sec in\ real\ life$).

▪ Verifying the time of a single complete washing cycle at the four given frequencies:



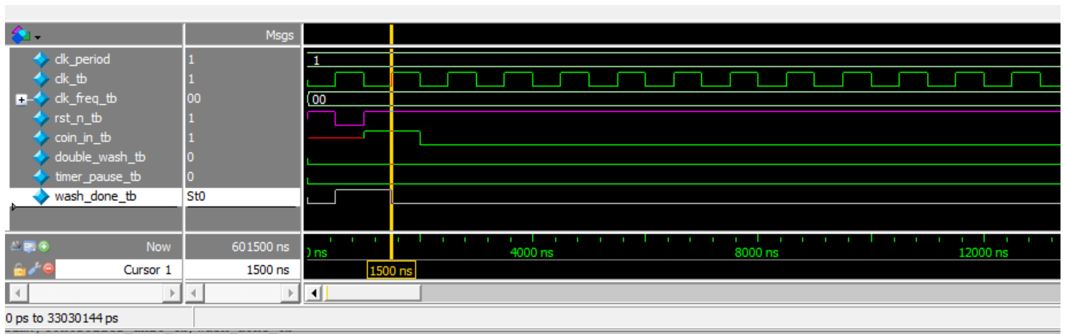
```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b0,2'b00);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      wait(wash_done_tb);
      $display("The washer did the job @time:", $time," seconds") ;


      $finish;
   end
```

<u>Result (@ 1MHZ freq.):</u>



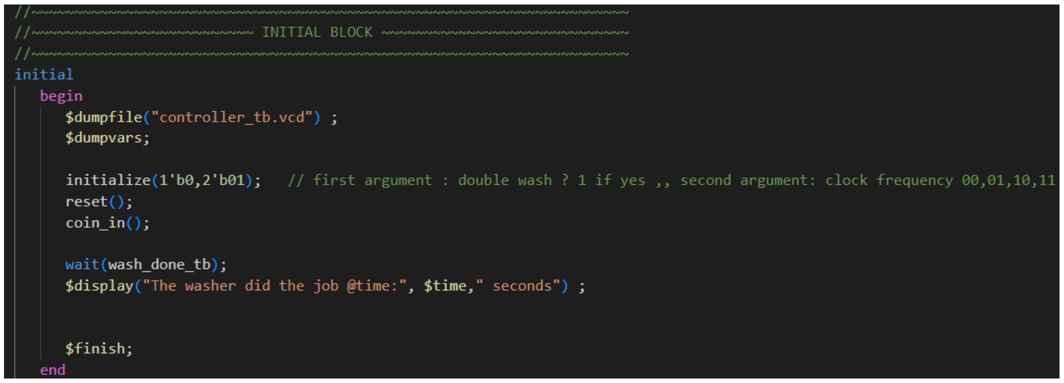
```
VSIM 1> run -all
# The washer did the job @time:               602 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 601500 ns  Iteration: 3  Instance: /controller_tb
```

- The previous figure shows the time of one complete washing operation at frequency 1MHZ. As mentioned, 600 microseconds in the simulation are equivalent to 600 second in real time. There is 1.5 microsecond which is the reference from which the design starts the operation.



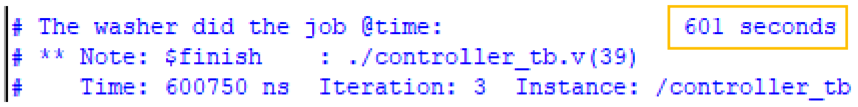
```
VSIM 2> run -all
# The Washing 1 is done @time:                    602
# ** Note: $finish    : ./controller_unit_tb.v(38)
#    Time: 601500 ns  Iteration: 3  Instance: /controller_unit_tb
```

- <u>The previous figure shows the time reference from which the simulation starts. The operation starts after 1.5 clock cycles (resetting and till the next +ve edge comes). Same for all the following cases.</u>



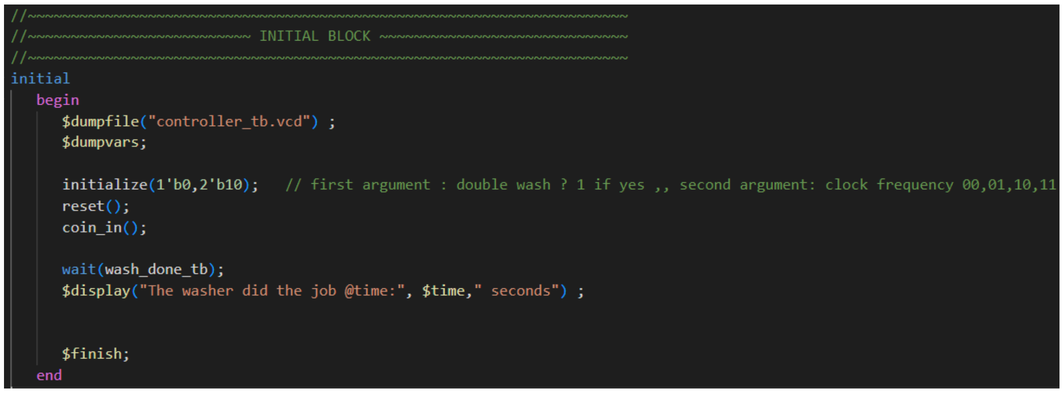
```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b0,2'b01);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      wait(wash_done_tb);
      $display("The washer did the job @time:", $time," seconds") ;


      $finish;
   end
```

<u>Result (@ 2 MHZ freq.):</u>



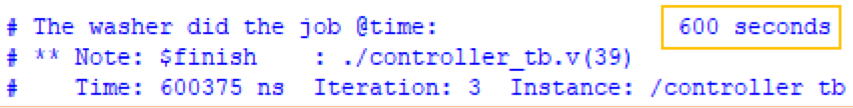
```
# The washer did the job @time:                 601 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 600750 ns  Iteration: 3  Instance: /controller_tb
```

- As explained in the previous case, time of complete operation is 600 seconds in real time. 600.75 = 600 + 1.5 clock cycles (time reference). The clock cycle in this case of 2MHZ freq. is 0.5 microseconds.

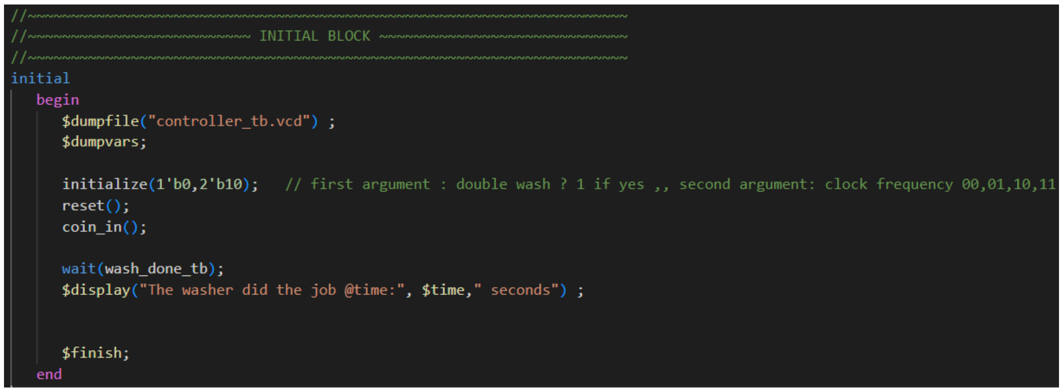


```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b0,2'b10);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      wait(wash_done_tb);
      $display("The washer did the job @time:", $time," seconds") ;


      $finish;
   end
```

<u>Result (@ 4 MHZ freq.):</u>

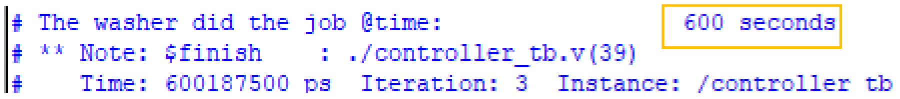


```
# The washer did the job @time:                 600 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 600375 ns  Iteration: 3  Instance: /controller tb
```



```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b0,2'b10);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      wait(wash_done_tb);
      $display("The washer did the job @time:", $time," seconds") ;


      $finish;
   end
```

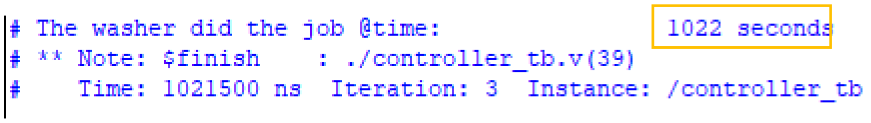
Result (@ 8 MHZ freq.):



```
# The washer did the job @time:                600 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 600187500 ps  Iteration: 3  Instance: /controller tb
```

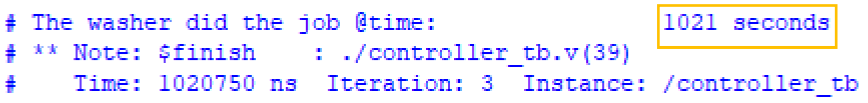
- Verifying the time of a double washing cycle at the four given frequencies

Result (@ 1 MHZ freq.):



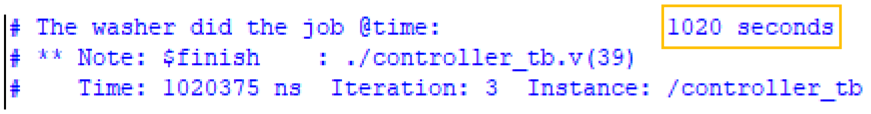
```
# The washer did the job @time:               1022 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 1021500 ns  Iteration: 3  Instance: /controller_tb
```

Result (@ 2 MHZ freq.):



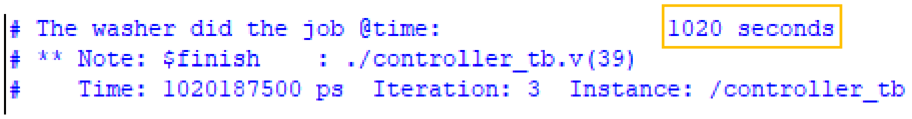
```
# The washer did the job @time:               1021 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 1020750 ns  Iteration: 3  Instance: /controller_tb
```

Result (@ 4 MHZ freq.):



```
# The washer did the job @time:               1020 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 1020375 ns  Iteration: 3  Instance: /controller_tb
```

Result (@ 8 MHZ freq.):



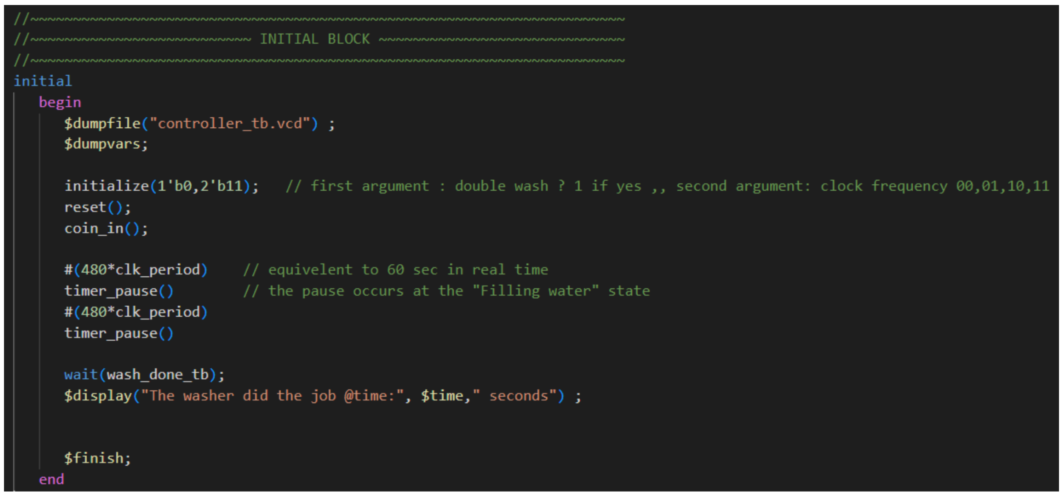
```
# The washer did the job @time:               1020 seconds
# ** Note: $finish    : ./controller_tb.v(39)
#    Time: 1020187500 ps  Iteration: 3  Instance: /controller_tb
```

- As expected, the time taken for a double wash is 17 mins (1020 seconds) from the reference starting time (1.5 clock periods at each frequency).

- Now its verified that the design is working correctly at the four different frequencies.
- The following part will test the “Timer_pause” input signal to verify that it only affects the “Spinning” state.
- I will use any of the given freq. as they all will give the same results so <u>I will use 8MHZ in all the following test cases.</u>

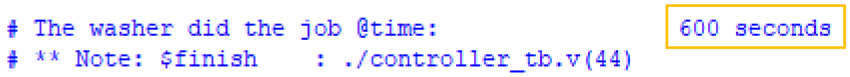
- **Testing the effectiveness of the “Timer_pause” input flag on the different states:**

- Working at 8MHZ frequency, single wash and raising the “timer_pause” flag at the “filling_water” state for 60 seconds. If the operation is completed at 600 seconds, then “timer_pause” input signal has no effect on the “filling_water” state.



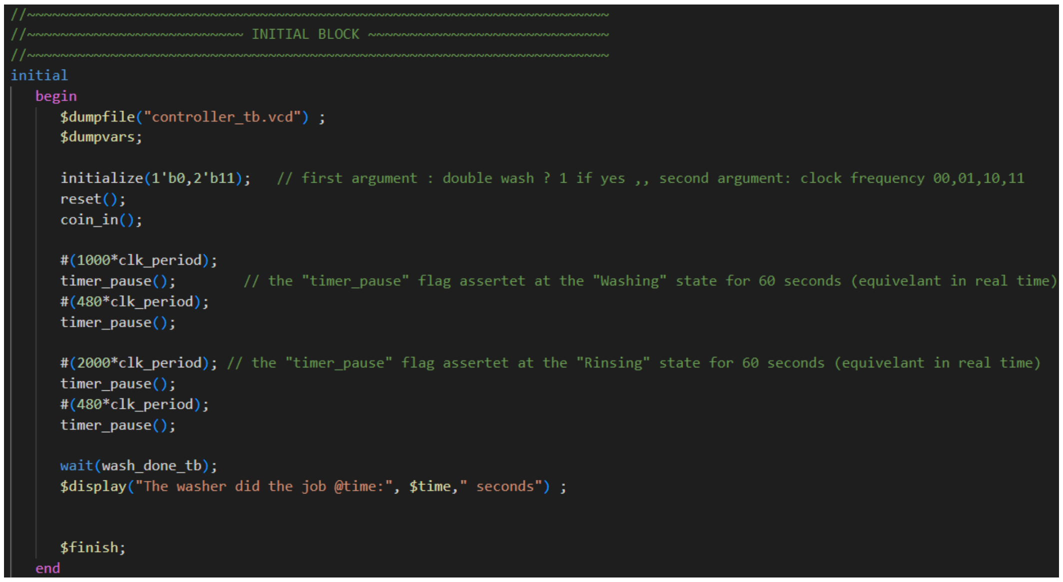
```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b0,2'b11);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      #(480*clk_period)    // equivelent to 60 sec in real time
      timer_pause()        // the pause occurs at the "Filling water" state
      #(480*clk_period)
      timer_pause()

      wait(wash_done_tb);
      $display("The washer did the job @time:", $time," seconds") ;


      $finish;
   end
```

Result:



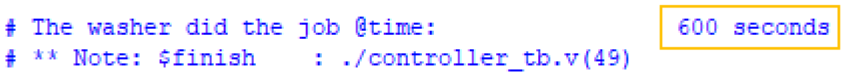
```
# The washer did the job @time:       600 seconds
# ** Note: $finish    : ./controller_tb.v(44)
```

- As expected, the “timer_pause” input flag has no effect on the “Filling_water” state.



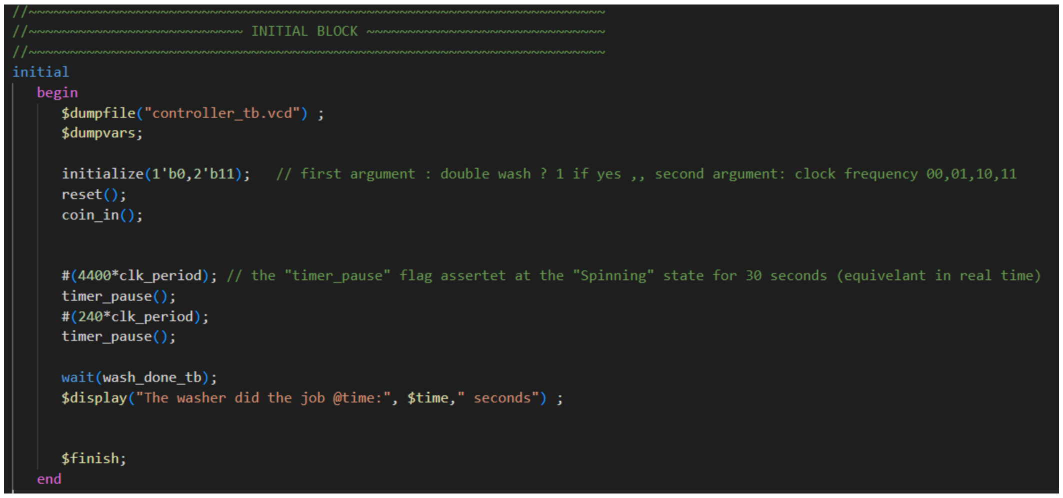
```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b0,2'b11);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();

      #(1000*clk_period);
      timer_pause();        // the "timer_pause" flag assertet at the "Washing" state for 60 seconds (equivelant in real time)
      #(480*clk_period);
      timer_pause();

      #(2000*clk_period); // the "timer_pause" flag assertet at the "Rinsing" state for 60 seconds (equivelant in real time)
      timer_pause();
      #(480*clk_period);
      timer_pause();

      wait(wash_done_tb);
      $display("The washer did the job @time:", $time," seconds") ;


      $finish;
   end
```

Result:



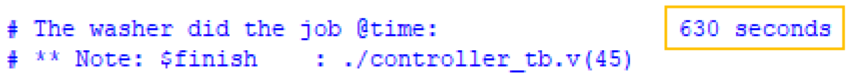
```
# The washer did the job @time:       600 seconds
# ** Note: $finish    : ./controller_tb.v(49)
```

In the previous case, “timer_pause” asserted for 60 seconds in “washing” and “Rinsing” states without any effect.



```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
    begin
        $dumpfile("controller_tb.vcd") ;
        $dumpvars;

        initialize(1'b0,2'b11);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
        reset();
        coin_in();


        #(4400*clk_period); // the "timer_pause" flag assertet at the "Spinning" state for 30 seconds (equivelant in real time)
        timer_pause();
        #(240*clk_period);
        timer_pause();

        wait(wash_done_tb);
        $display("The washer did the job @time:", $time," seconds") ;


        $finish;
    end
```

Result:



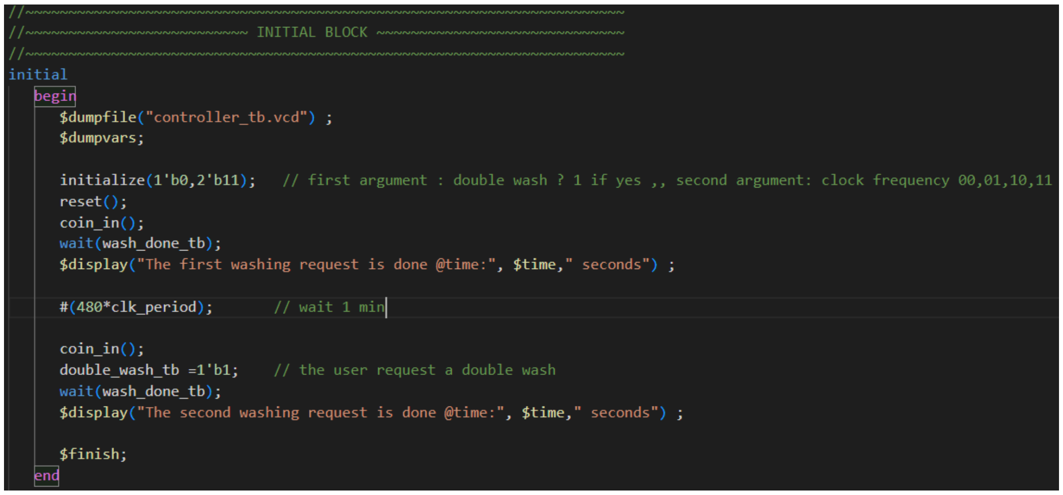
```
# The washer did the job @time:                 630 seconds
# ** Note: $finish    : ./controller_tb.v(45)
```

The previous case shows the effect of the “timer_pause” input flag on the “Spinning” state. I asserted the flag for 30 equivalent seconds at the “Spinning” state and the machine took 630 seconds to complete the job, which is expected and achieves the design requirements.

Now all the requirements of the design are achieved. The next test cases are more complex to represent more complex scenarios.

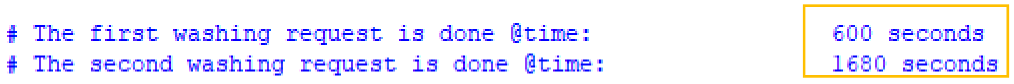
- **Testing more complex scenarios:**

- The first user requested a single wash and when the washing is done, another user came after 1 minute and requested a double wash, the expected time for the whole scenario is 10 mins + 1 min + 17 mins = 28 mins (1680 seconds).



```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
    begin
        $dumpfile("controller_tb.vcd") ;
        $dumpvars;

        initialize(1'b0,2'b11);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
        reset();
        coin_in();
        wait(wash_done_tb);
        $display("The first washing request is done @time:", $time," seconds") ;

        #(480*clk_period);       // wait 1 min

        coin_in();
        double_wash_tb =1'b1;    // the user request a double wash
        wait(wash_done_tb);
        $display("The second washing request is done @time:", $time," seconds") ;

        $finish;
    end
```

Result:

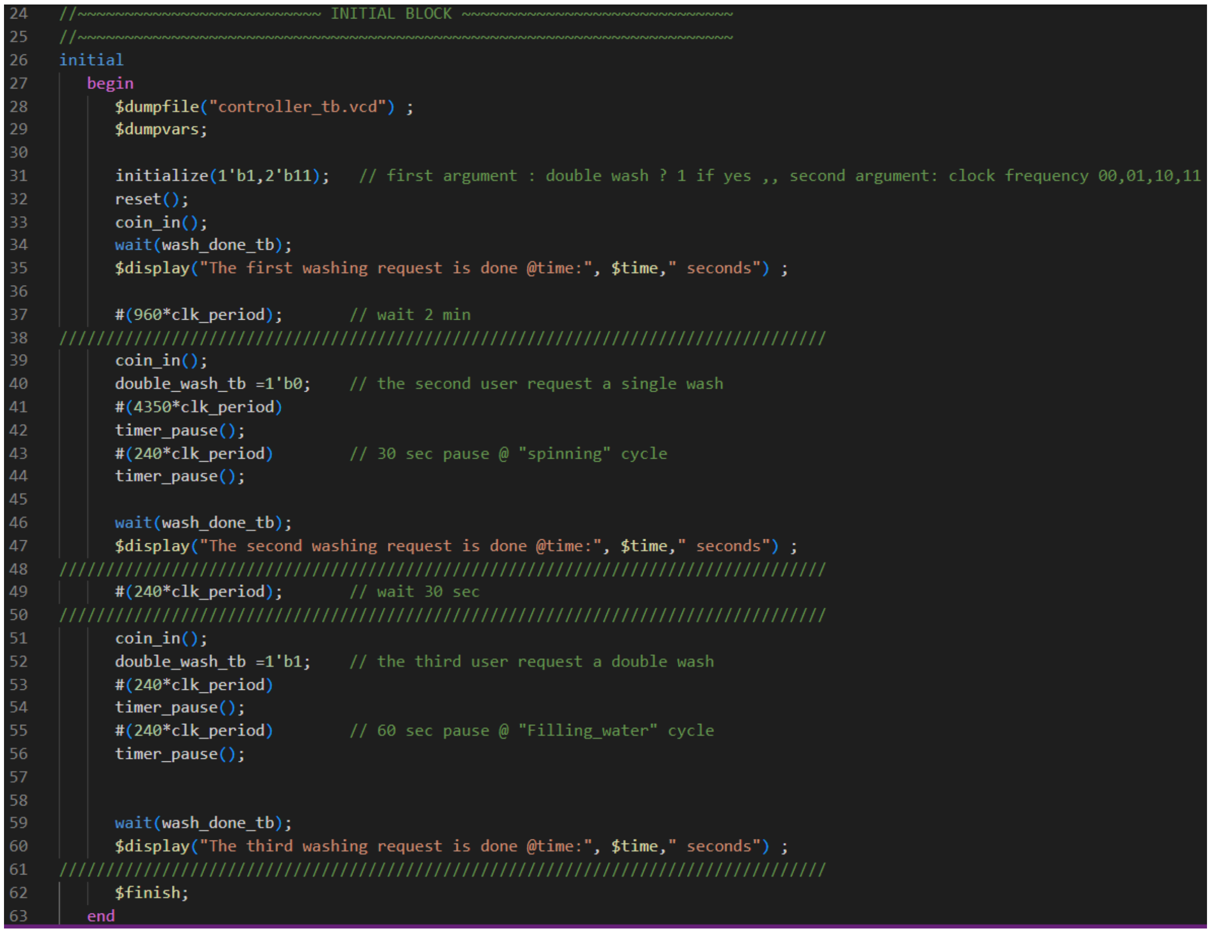


```
# The first washing request is done @time:              600 seconds
# The second washing request is done @time:             1680 seconds
```

- The following scenario is more complex, the first user requested a double wash and after 2 mins another user requested a single wash but pressed the timer pause button for 30sec at “Spinning” cycle and when the second request is done, a third user came after 30 seconds and requested a double wash. The third user pressed the timer pause button at “Filling_water” cycle for 1 min. The expected time for the whole scenario is:

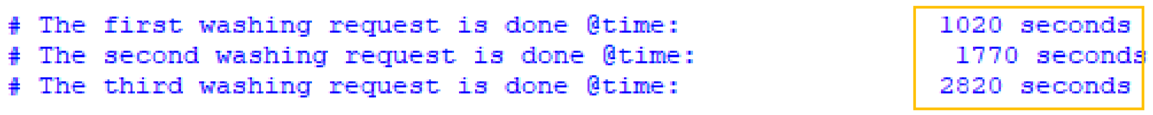
First user:	17min Double wash
IDLE:	2 min
Second user:	10min Single wash + 30sec pause
IDLE:	30sec
Third user:	17min Double wash + 1min pause (but ignored as the pause wasn’t at the spinning cycle)

Total: 47 min (2820 seconds)



```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~ INITIAL BLOCK ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
initial
   begin
      $dumpfile("controller_tb.vcd") ;
      $dumpvars;

      initialize(1'b1,2'b11);   // first argument : double wash ? 1 if yes ,, second argument: clock frequency 00,01,10,11
      reset();
      coin_in();
      wait(wash_done_tb);
      $display("The first washing request is done @time:", $time," seconds") ;

      #(960*clk_period);       // wait 2 min
////////////////////////////////////////////////////////////////////////////////////
      coin_in();
      double_wash_tb =1'b0;    // the second user request a single wash
      #(4350*clk_period)
      timer_pause();
      #(240*clk_period)        // 30 sec pause @ "spinning" cycle
      timer_pause();

      wait(wash_done_tb);
      $display("The second washing request is done @time:", $time," seconds") ;
////////////////////////////////////////////////////////////////////////////////////
      #(240*clk_period);       // wait 30 sec
////////////////////////////////////////////////////////////////////////////////////
      coin_in();
      double_wash_tb =1'b1;    // the third user request a double wash
      #(240*clk_period)
      timer_pause();
      #(240*clk_period)        // 60 sec pause @ "Filling_water" cycle
      timer_pause();


      wait(wash_done_tb);
      $display("The third washing request is done @time:", $time," seconds") ;
////////////////////////////////////////////////////////////////////////////////////
      $finish;
   end
```

Result:

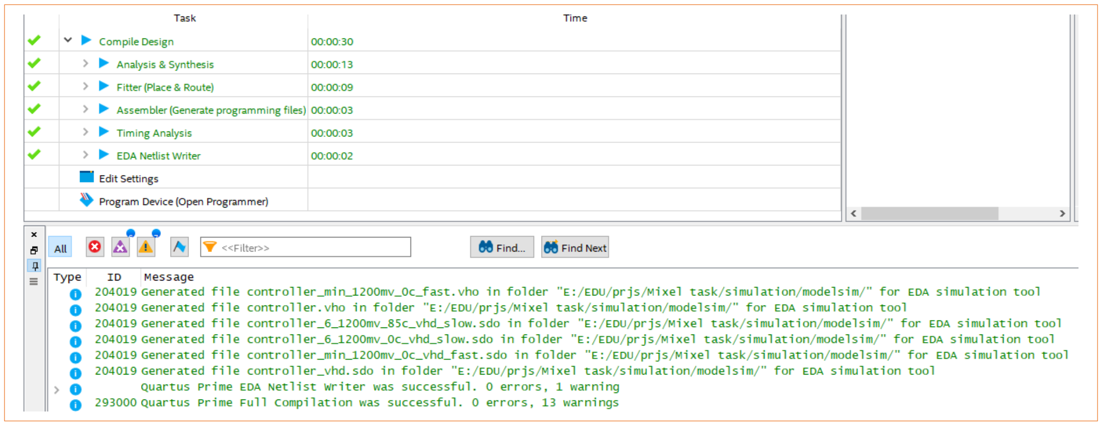


```
# The first washing request is done @time:          1020 seconds
# The second washing request is done @time:          1770 seconds
# The third washing request is done @time:          2820 seconds
```

- The machine completed the job at the expected time.

- Conclusion:

- The design successfully achieved the required specifications and behaves as designed and expected.
- The Verilog design code is synthesizable and free from errors.



Task	Time
✓ Compile Design	00:00:30
✓ Analysis & Synthesis	00:00:13
✓ Fitter (Place & Route)	00:00:09
✓ Assembler (Generate programming files)	00:00:03
✓ Timing Analysis	00:00:03
✓ EDA Netlist Writer	00:00:02
Edit Settings	
Program Device (Open Programmer)	

```
Type   ID     Message
       204019 Generated file controller_min_1200mv_0c_fast.vho in folder "E:/EDU/prjs/Mixel task/simulation/modelsim/" for EDA simulation tool
       204019 Generated file controller.vho in folder "E:/EDU/prjs/Mixel task/simulation/modelsim/" for EDA simulation tool
       204019 Generated file controller_6_1200mv_85c_vhd_slow.sdo in folder "E:/EDU/prjs/Mixel task/simulation/modelsim/" for EDA simulation tool
       204019 Generated file controller_6_1200mv_0c_vhd_slow.sdo in folder "E:/EDU/prjs/Mixel task/simulation/modelsim/" for EDA simulation tool
       204019 Generated file controller_min_1200mv_0c_vhd_fast.sdo in folder "E:/EDU/prjs/Mixel task/simulation/modelsim/" for EDA simulation tool
       204019 Generated file controller_vhd.sdo in folder "E:/EDU/prjs/Mixel task/simulation/modelsim/" for EDA simulation tool
              Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
       293000 Quartus Prime Full Compilation was successful. 0 errors, 13 warnings
```

Figure 4 Compiling the design using Quartus prime

*All the warnings are not related to the design.

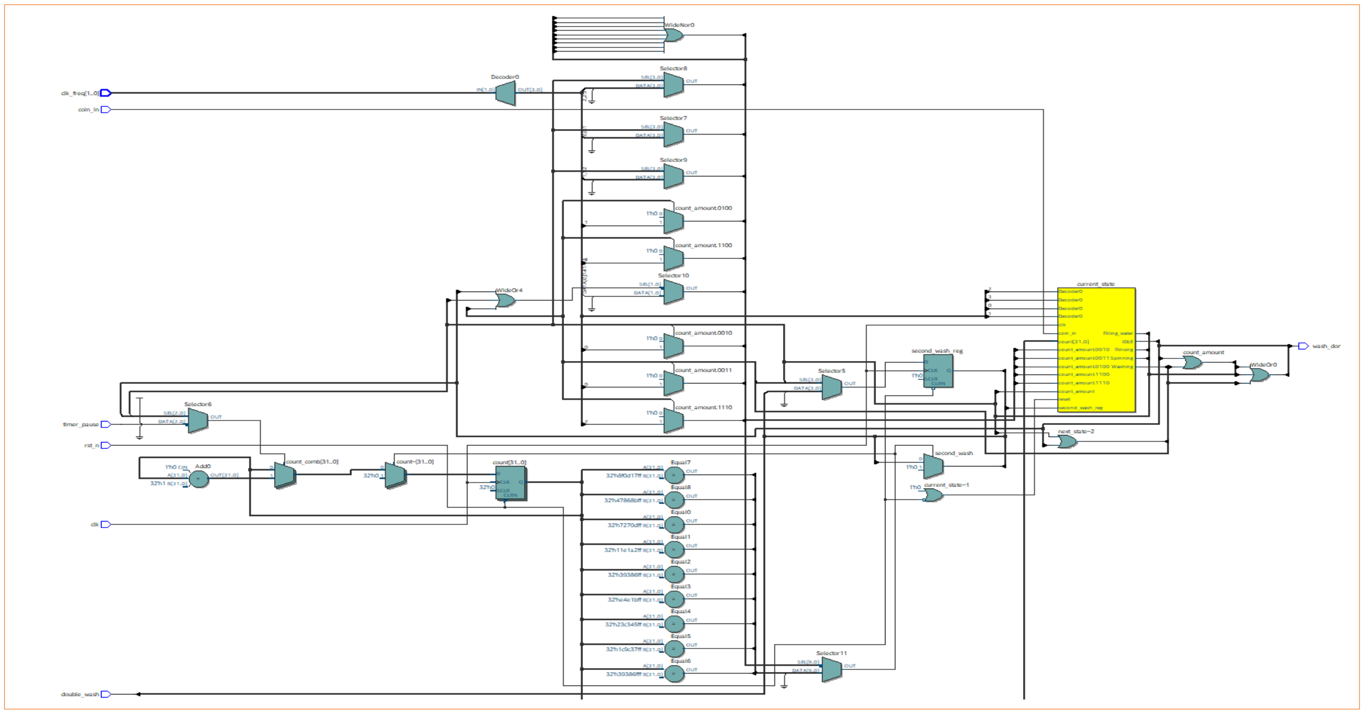


Figure 5 Gate level netlist of the design

- The design could be represented by two separate modules, FSM and counter and connecting them in the top module, but I chose to make them together.

The assignment was very interesting, I hope I did it as expected.

Thanks!