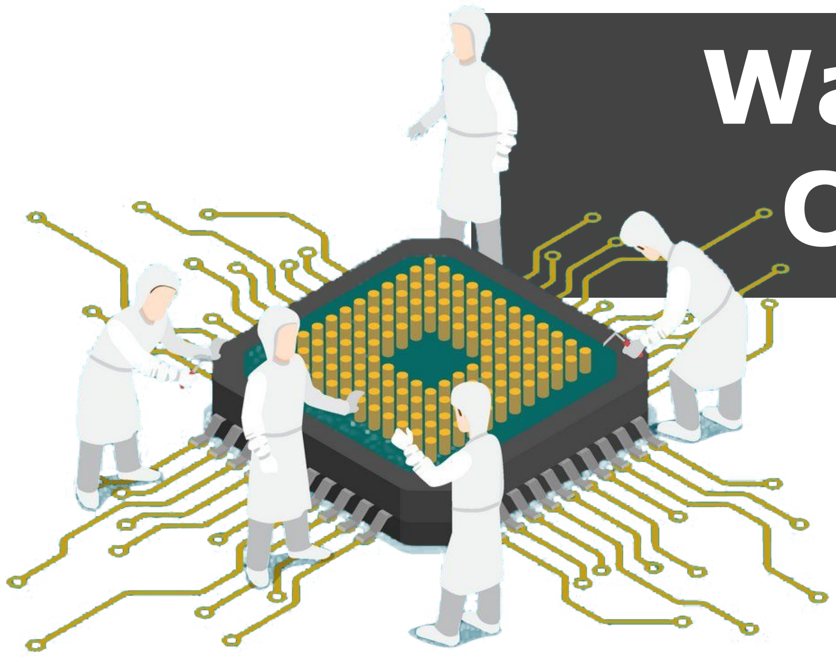


Washing Machine Controller Unit

Prepared By:

Shehab Bahaa

Outline

Problem Definition & Requirements

Problem Definition: Designing the controller unit for a washing machine

Requirements:

- The machine operates in a sequence of states as follows
  1) The machine rests in the **idle state** until ***coin_in input port*** is asserted which corresponds to a coin being deposited.
  2) The asserted coin_in transfers the machine into the **filling water state**. The duration of this state is **2 minutes**.
  3) Then the machine moves into the **washing state** that takes **5 minutes**.
  4) After that, it moves into the **rinsing state** that takes **2 minutes**.

Problem Definition & Requirements

Requirements:

5) Then depending on the value at the ***double_wash input port***: **Either** it moves into the **spinning state** if double_wash is deasserted **Or** it moves into the **washing state** for another wash and rinse round if double_wash is asserted. Note: Only one more round of wash and rinse is allowed when double_wash asserted even if it is still asserted after the second wash and rinse round.
6) After that, the **spinning state** starts. Its duration is **1 minute**. However, if the ***timer_pause input port*** is asserted during this state (only the spinning state), the machine moves into the idle state. When the timer_pause flag is deasserted again, then the spinning state continues the time remaining from before the pause.

Problem Definition & Requirements

Requirements:

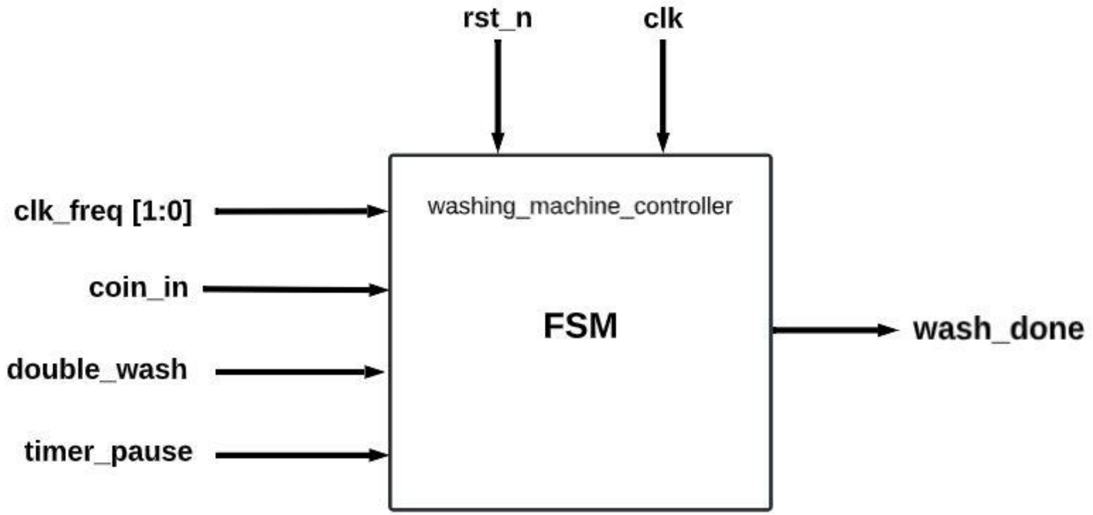
7) When the spinning state is done, the machine moves into the idle state, and the ***wash_done output port*** is asserted until another coin is deposited, then the sequence starts from the beginning again.

- Regarding the input signals:
  1) The double_wash is pressed pressed before depositing the coin and **stays pressed till the job completes**.
  2) The machine is designed to stop when the timer_pause flag is asserted **ONLY** during the spinning state.
  3) The input clock can have 1, 2, 4, or 8 MHz frequency. ***clk_freq [1:0]*** input port is used to encode the clock frequency used.

Overview of the Solution Flow

Step 1: defining the block diagram

The washing_machine_controller consists of a finite state machine that controls the state of the washing machine and the output.



Block Diagram

Overview of the Solution Flow

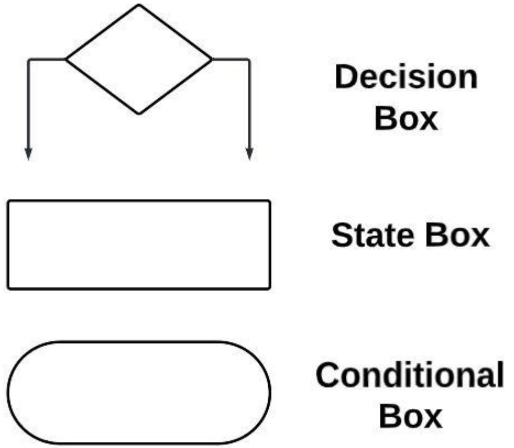
Step 2: developing Algorithmic State Machine (ASM) chart

- ASM chart is used for designing FSMs with three basic elements: the state box, the decision box, and the conditional box.
- The chart consists of ASM blocks that describe the state of the system during one clock-pulse interval.

Step 3: Convert the ASM chart into Verilog model

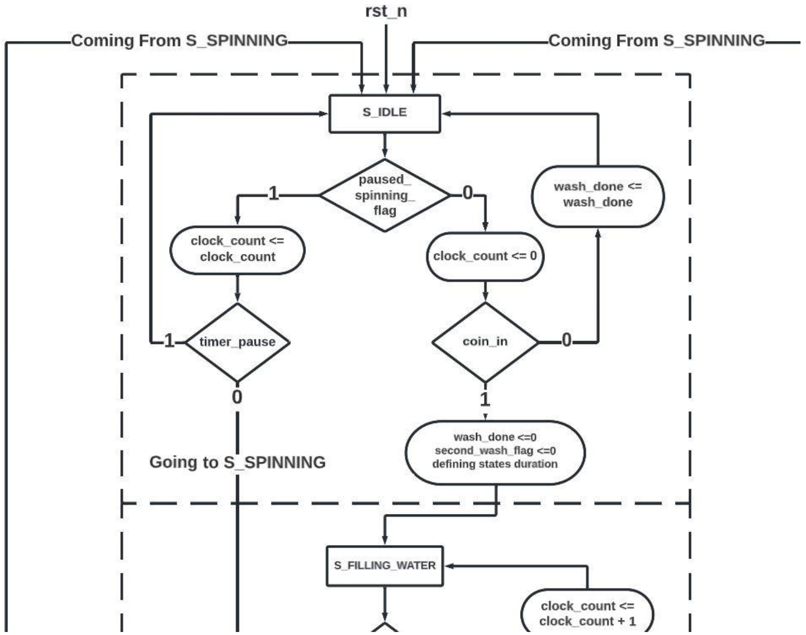
Step 4: Testbench development and simulation

Step 5: Synthesis

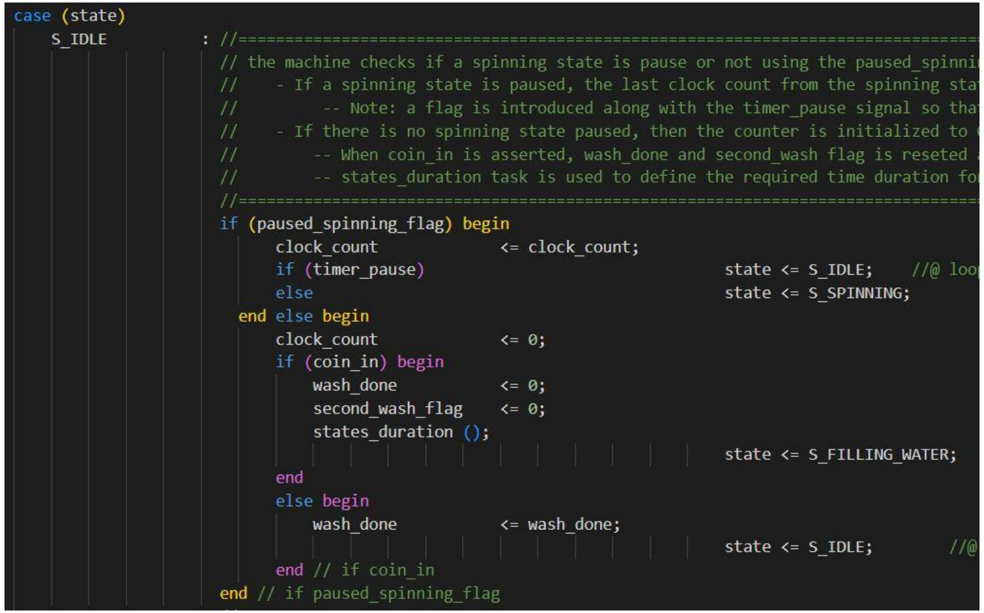


Basic Elements of ASM chart

ASM Block and RTL Code of Idle State



Idle state ASM block



```
case (state)
    S_IDLE              : //================================================================================
                          // the machine checks if a spinning state is pause or not using the paused_spinni
                          //    - If a spinning state is paused, the last clock count from the spinning sta
                          //         -- Note: a flag is introduced along with the timer_pause signal so tha
                          //    - If there is no spinning state paused, then the counter is initialized to
                          //        -- When coin_in is asserted, wash_done and second_wash flag is reseted
                          //        -- states_duration task is used to define the required time duration fo
                          //================================================================================
                          if (paused_spinning_flag) begin
                              clock_count           <= clock_count;
                              if (timer_pause)                                    state <= S_IDLE;    //@ loo
                              else                                                state <= S_SPINNING;
                            end else begin
                              clock_count           <= 0;
                              if (coin_in) begin
                                  wash_done         <= 0;
                                  second_wash_flag  <= 0;
                                  states_duration ();
                                                                                  state <= S_FILLING_WATER;
                              end
                              else begin
                                  wash_done         <= wash_done;
                                                                                  state <= S_IDLE;        //@
                              end // if coin_in
                          end // if paused_spinning_flag
```

Idle state RTL code

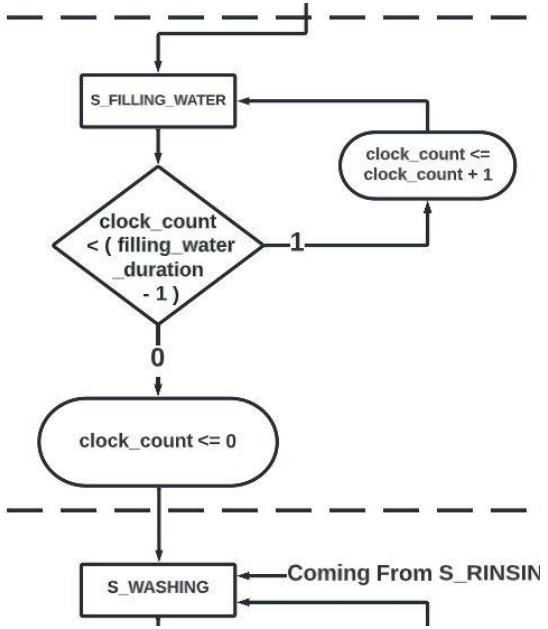
RTL Code Explanation of Idle State

- The machine enters the IDLE state in two cases:
  1) When the reset signal is 0 (active-low signal)
  2) When the spinning state is paused or done.
- In the IDLE state, the machine checks if a spinning state is pause or not using the paused_spinning_flag:
  - If a spinning state is paused, the last clock count from the spinning state is saved until the timer_pause input is deasserted, then it continues the spinning state. **Note:** a flag is introduced along with the timer_pause signal so that the timer_pause is considered only when the spinning state is paused.

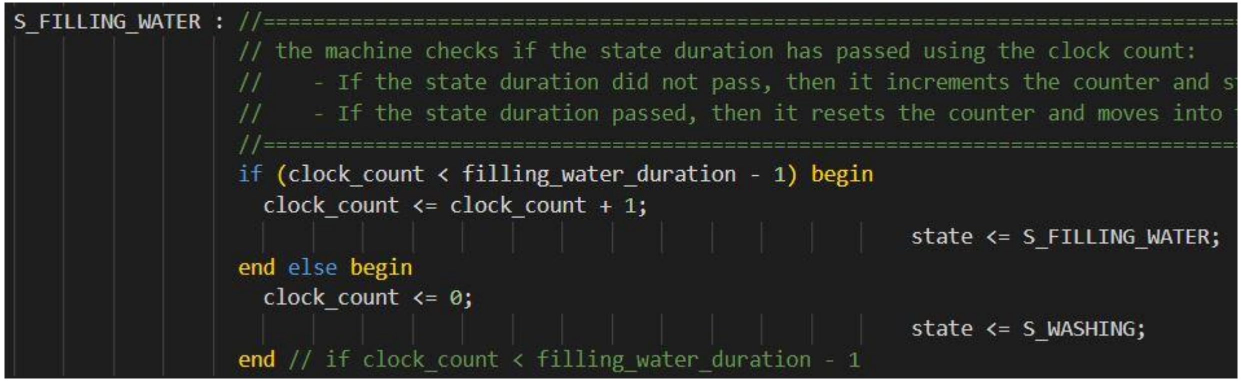
RTL Code Explanation of Idle State

- If there is no spinning state paused, then the counter is initialized to 0 and the value of wash_done signal is saved until the coin_in signal is asserted.
  - When coin_in is asserted, wash_done and second_wash flag is reseted and the state_duration task is called.
  - **states_duration task** is used to define the required time duration for each state. The time duration is defined using a counter for the number of clock cycles of each state. The number a clock cycles per state depends on the clock frequency used and the state itself. It is calculated as follows:
    # of clock cycles per state = (frequency, i.e., cycles per sec)*(state duration in minutes)*(60 s/min)

ASM Block and RTL Code of Filling Water State


S_FILLING_WATER
clock_count <= clock_count + 1
clock_count < (filling_water _duration - 1)
1
0
clock_count <= 0
S_WASHING
Coming From S_RINSIN

Filling water state ASM block



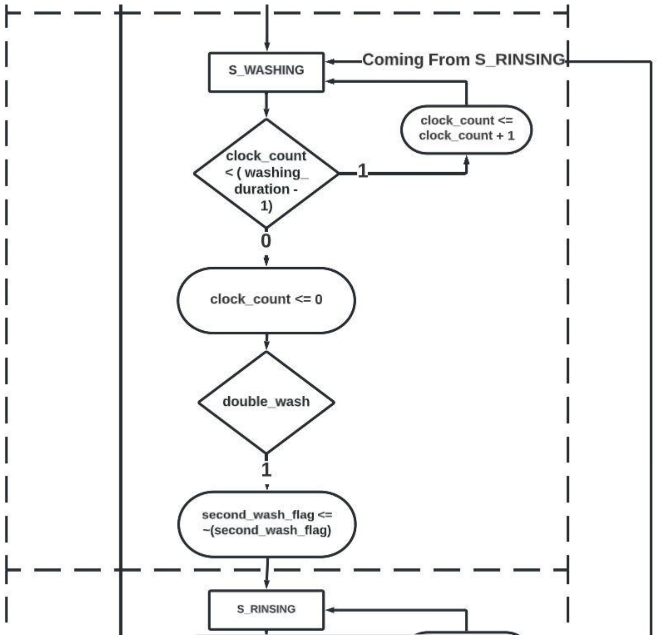
```
S_FILLING_WATER : //==========================================================================
                  // the machine checks if the state duration has passed using the clock count:
                  //    - If the state duration did not pass, then it increments the counter and s
                  //    - If the state duration passed, then it resets the counter and moves into
                  //==========================================================================
                  if (clock_count < filling_water_duration - 1) begin
                    clock_count <= clock_count + 1;
                                                                          state <= S_FILLING_WATER;
                  end else begin
                    clock_count <= 0;
                                                                          state <= S_WASHING;
                  end // if clock_count < filling_water_duration - 1
```

Filling water state RTL code

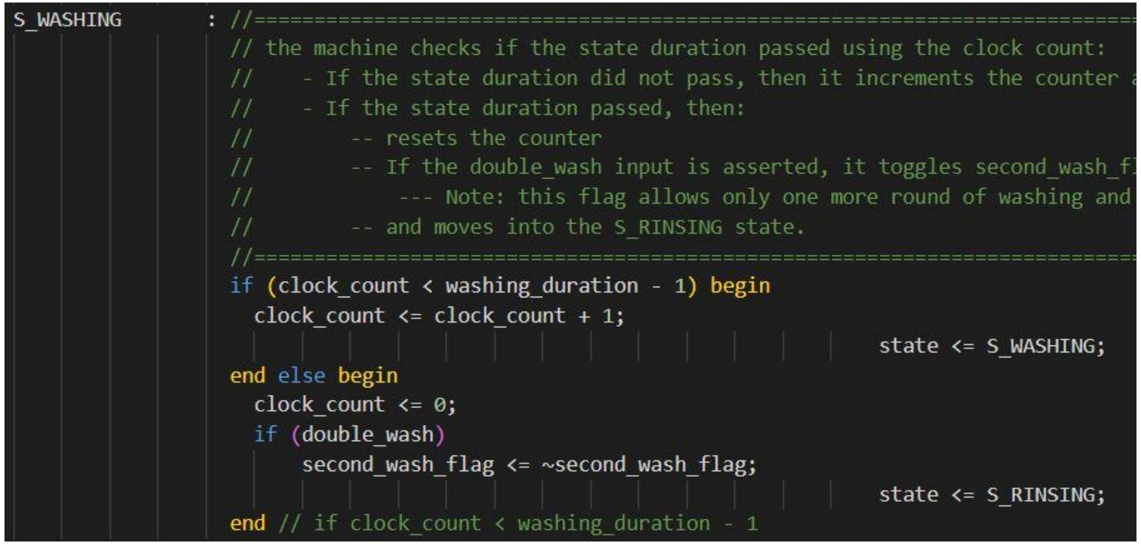
RTL Code Explanation of Filling Water State

- The S_FILLING_WATER state starts after the S_IDLE state
- In this state, the machine checks if the state duration passed using the clock count:
  - If the state duration did not pass, then it increments the counter and stays in this state.
  - If the state duration passed, then it resets the counter and moves into the S_WASHING state.

ASM Block and RTL Code of Washing State


S_WASHING
Coming From S_RINSING
clock_count <= clock_count + 1
clock_count < (washing_ duration - 1)
1
0
clock_count <= 0
double_wash
1
second_wash_flag <= ~(second_wash_flag)
S_RINSING

Washing state ASM block



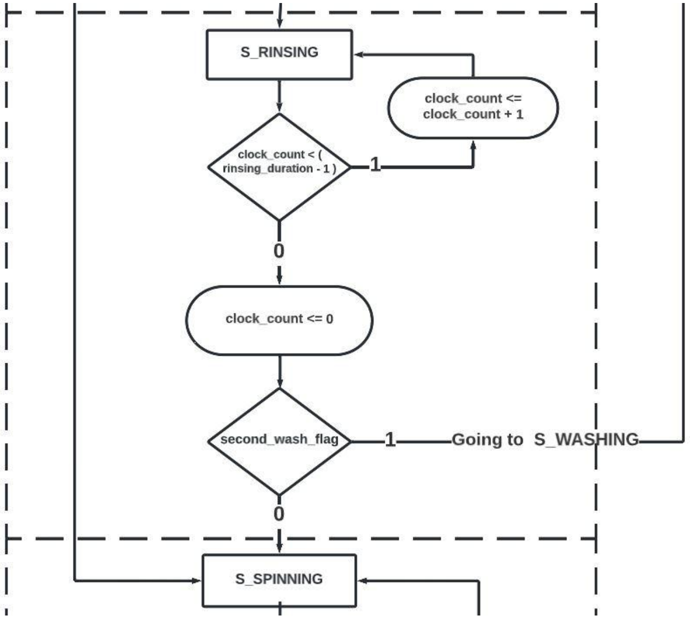
```
S_WASHING       : //==========================================================================
                  // the machine checks if the state duration passed using the clock count:
                  //    - If the state duration did not pass, then it increments the counter a
                  //    - If the state duration passed, then:
                  //        -- resets the counter
                  //        -- If the double_wash input is asserted, it toggles second_wash_fl
                  //            --- Note: this flag allows only one more round of washing and
                  //        -- and moves into the S_RINSING state.
                  //==========================================================================
                  if (clock_count < washing_duration - 1) begin
                    clock_count <= clock_count + 1;
                                                                  state <= S_WASHING;
                  end else begin
                    clock_count <= 0;
                    if (double_wash)
                        second_wash_flag <= ~second_wash_flag;
                                                                  state <= S_RINSING;
                  end // if clock_count < washing_duration - 1
```

Washing state RTL code

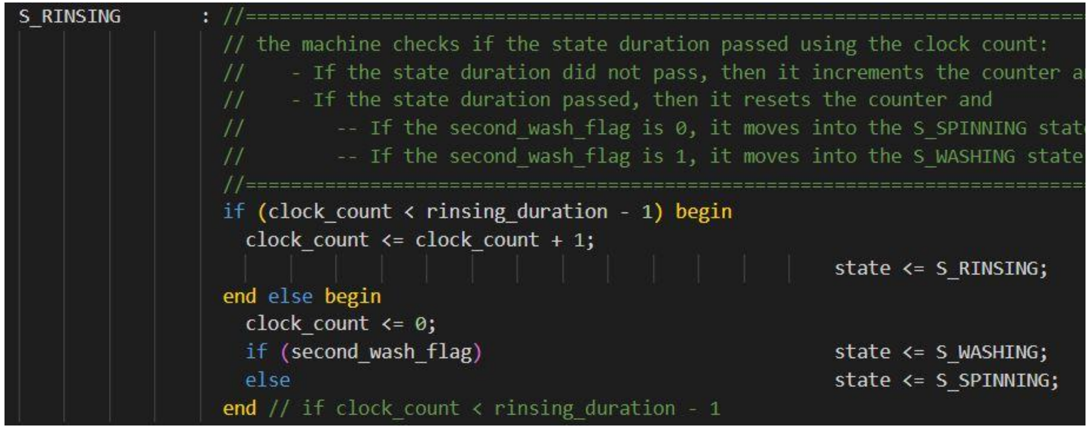
RTL Code Explanation of Washing State

- The S_WASHING state starts after the S_FILLING_WATER state or after S_SPINNING if double_wash is asserted.
- In this state, the machine checks if the state duration passed using the clock count:
  - If the state duration did not pass, then it increments the counter and stays in this state.
  - If the state duration passed, then:
    - resets the counter
    - If the double_wash input is asserted, it toggles second_wash_flag. **Note:** this flag allows only one more round of washing and rinsing even if double_wash is still asserted after the second round.
    - and moves into the S_RINSING state.

ASM Block and RTL Code of Rinsing State



Rinsing state ASM block



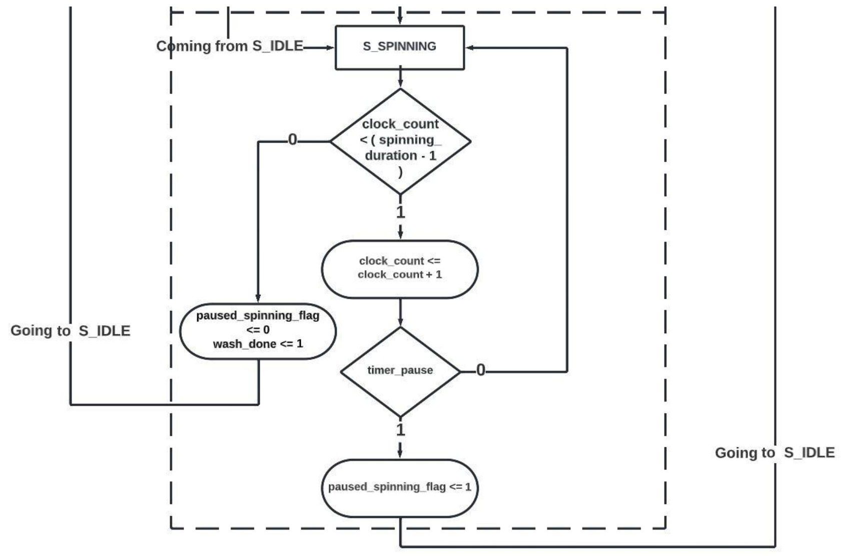
```
S_RINSING        : //=====================================================================
                   // the machine checks if the state duration passed using the clock count:
                   //    - If the state duration did not pass, then it increments the counter a
                   //    - If the state duration passed, then it resets the counter and
                   //        -- If the second_wash_flag is 0, it moves into the S_SPINNING stat
                   //        -- If the second_wash_flag is 1, it moves into the S_WASHING state
                   //=====================================================================
                   if (clock_count < rinsing_duration - 1) begin
                     clock_count <= clock_count + 1;
                                                               state <= S_RINSING;
                   end else begin
                     clock_count <= 0;
                     if (second_wash_flag)                     state <= S_WASHING;
                     else                                      state <= S_SPINNING;
                   end // if clock_count < rinsing_duration - 1
```

Rinsing state RTL code

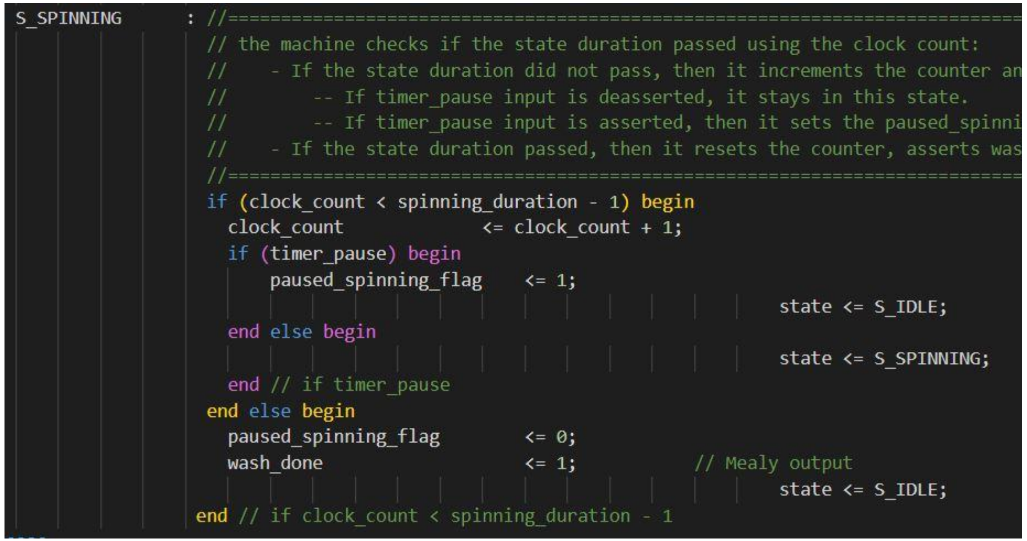
RTL Code Explanation of Rinsing State

- The S_RINSING state starts after the S_WASHING state
- In this state, the machine checks if the state duration passed using the clock count:
  - If the state duration did not pass, then it increments the counter and stays in this state.
  - If the state duration passed, then it resets the counter and
    - If the second_wash_flag is 0, it moves into the S_SPINNING state.
    - If the second_wash_flag is 1, it moves into the S_WASHING state.

ASM Block and RTL Code of Spinning State


Coming from S_IDLE
S_SPINNING
clock_count < (spinning_ duration - 1)
0
1
clock_count <= clock_count + 1
paused_spinning_flag <= 0 wash_done <= 1
Going to S_IDLE
timer_pause
0
1
paused_spinning_flag <= 1
Going to S_IDLE

Spinning state ASM block



```
S_SPINNING      : //=========================================================================
                  // the machine checks if the state duration passed using the clock count:
                  //    - If the state duration did not pass, then it increments the counter an
                  //        -- If timer_pause input is deasserted, it stays in this state.
                  //        -- If timer_pause input is asserted, then it sets the paused_spinni
                  //    - If the state duration passed, then it resets the counter, asserts was
                  //=========================================================================
                  if (clock_count < spinning_duration - 1) begin
                    clock_count              <= clock_count + 1;
                    if (timer_pause) begin
                      paused_spinning_flag    <= 1;
                                                                  state <= S_IDLE;
                    end else begin
                                                                  state <= S_SPINNING;
                    end // if timer_pause
                  end else begin
                    paused_spinning_flag        <= 0;
                    wash_done                   <= 1;          // Mealy output
                                                                  state <= S_IDLE;
                end // if clock_count < spinning_duration - 1
```

Spinning state RTL code

RTL Code Explanation of Spinning State

- The S_SPINNING state starts after the S_RINSING state or after S_IDLE state if it was paused.
- In this state, the machine checks if the state duration passed using the clock count:
    - If the state duration did not pass, then it increments the counter and checks:
        - If timer_pause input is deasserted, it stays in this state.
        - If timer_pause input is asserted, then it sets the paused_spinning_flag and moves into S_IDLE state.
    - If the state duration passed, then it resets the counter, asserts wash_done and moves into the S_IDLE state.

Verification

Verification Methodology

Grey-Box methodology was used in the verification. This approach involves:

- Controlling and observing the design through the top-level interface
- Verifying implementation-specific features. In our case, the state registers and the duration of each state were checked.

Testing Covered Scenarios

1) Test 1: complete washing cycle with no double wash and no timer pause.
2) Test 2: complete washing cycle with double wash and timer pause.
3) Test 3: two consecutive washing cycles (running test 1 then test 2)

Verification

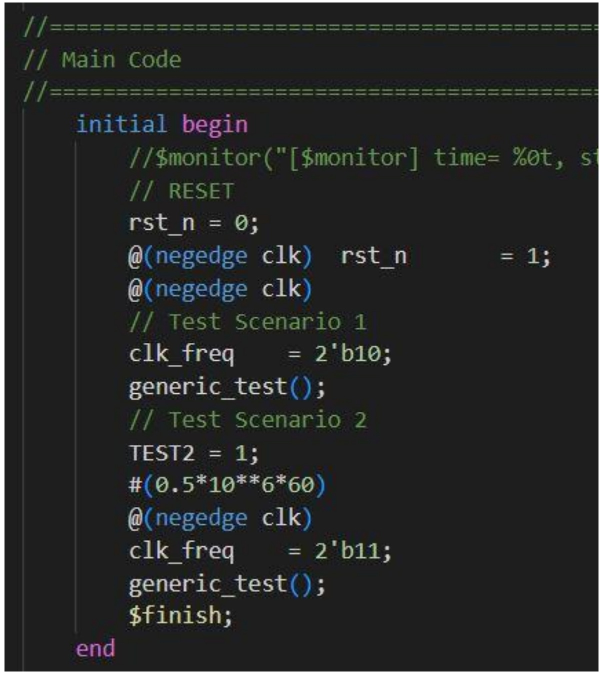
Checkers

A self-checking testbench was developed that checks the following in each test scenario:

- The sequence of state transitions and duration of each state
- The value of the wash_done output signal

Verification: Testbench Explanation

- The testbench implements the test scenarios using a **generic_test task**.
- The content of the generic_test is modified using flags to select the required test scenario.
- Figure shows that the DUT is initially reseted, then test 1 runs. After 0.5 minute in the idle state, the second test is applied.
- Both tests are applied consecutively which constitutes the third test scenario.
- Also, the clk frequency changes for each test scenario.

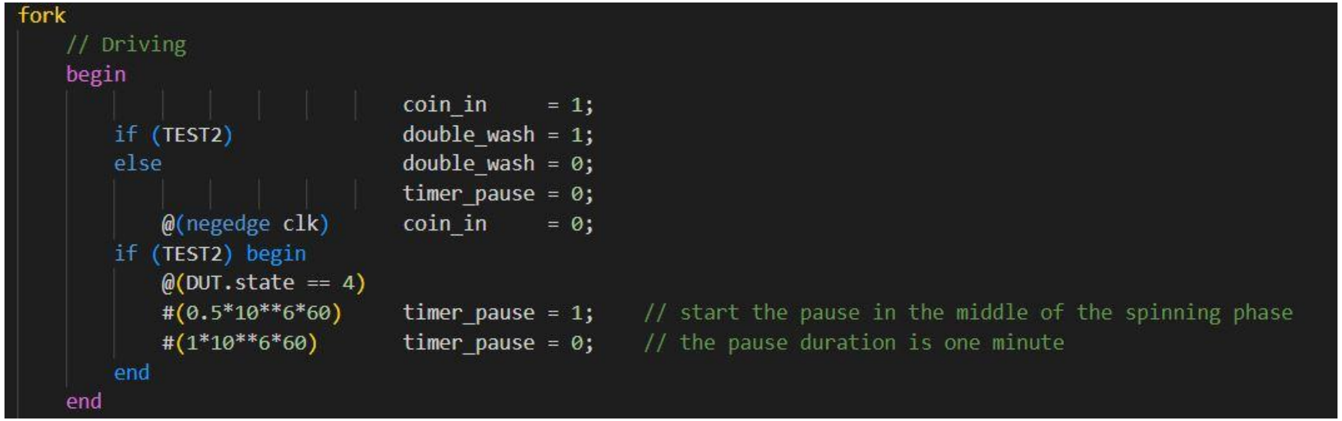


```
//========================================
// Main Code
//========================================
    initial begin
        //$monitor("[$monitor] time= %0t, s
        // RESET
        rst_n = 0;
        @(negedge clk)  rst_n       = 1;
        @(negedge clk)
        // Test Scenario 1
        clk_freq    = 2'b10;
        generic_test();
        // Test Scenario 2
        TEST2 = 1;
        #(0.5*10**6*60)
        @(negedge clk)
        clk_freq    = 2'b11;
        generic_test();
        $finish;
    end
```

Sequence of test scenarios

Verification: Testbench Explanation

- The **generic_test task** consists of two part:
  - Driver drives the input signals
  - Monitor detects the output signal and current state.
- The figure below shows the driving part of the generic_test task.

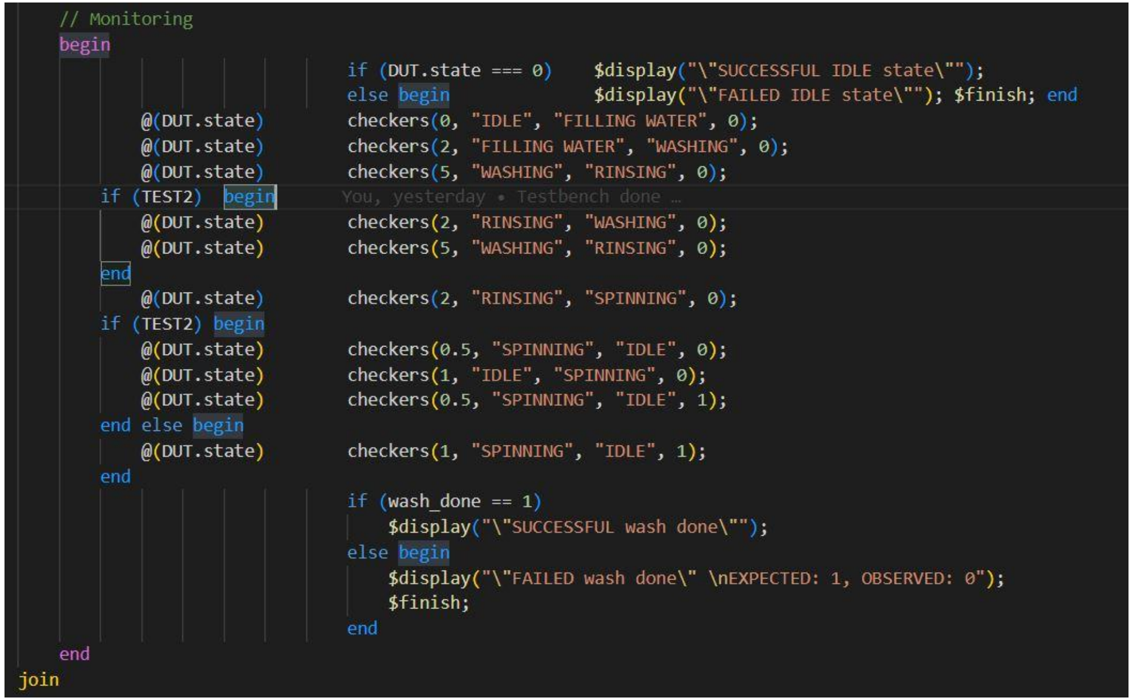


```
fork
    // Driving
    begin
                                coin_in     = 1;
        if (TEST2)              double_wash = 1;
        else                    double_wash = 0;
                                timer_pause = 0;
            @(negedge clk)      coin_in     = 0;
        if (TEST2) begin
            @(DUT.state == 4)
            #(0.5*10**6*60)     timer_pause = 1;    // start the pause in the middle of the spinning phase
            #(1*10**6*60)       timer_pause = 0;    // the pause duration is one minute
        end
    end
```

Driving part of generic_test

Verification: Testbench Explanation

- The monitor part of the generic_test performs multiple checks whenever the state of the DUT changes.



```
    // Monitoring
    begin
                        if (DUT.state === 0)    $display("\"SUCCESSFUL IDLE state\"");
                        else begin              $display("\"FAILED IDLE state\""); $finish; end
            @(DUT.state)        checkers(0, "IDLE", "FILLING WATER", 0);
            @(DUT.state)        checkers(2, "FILLING WATER", "WASHING", 0);
            @(DUT.state)        checkers(5, "WASHING", "RINSING", 0);
        if (TEST2) begin        You, yesterday • Testbench done …
            @(DUT.state)        checkers(2, "RINSING", "WASHING", 0);
            @(DUT.state)        checkers(5, "WASHING", "RINSING", 0);
        end
            @(DUT.state)        checkers(2, "RINSING", "SPINNING", 0);
        if (TEST2) begin
            @(DUT.state)        checkers(0.5, "SPINNING", "IDLE", 0);
            @(DUT.state)        checkers(1, "IDLE", "SPINNING", 0);
            @(DUT.state)        checkers(0.5, "SPINNING", "IDLE", 1);
        end else begin
            @(DUT.state)        checkers(1, "SPINNING", "IDLE", 1);
        end
                        if (wash_done == 1)
                            $display("\"SUCCESSFUL wash done\"");
                        else begin
                            $display("\"FAILED wash done\" \nEXPECTED: 1, OBSERVED: 0");
                            $finish;
                        end
    end
join
```

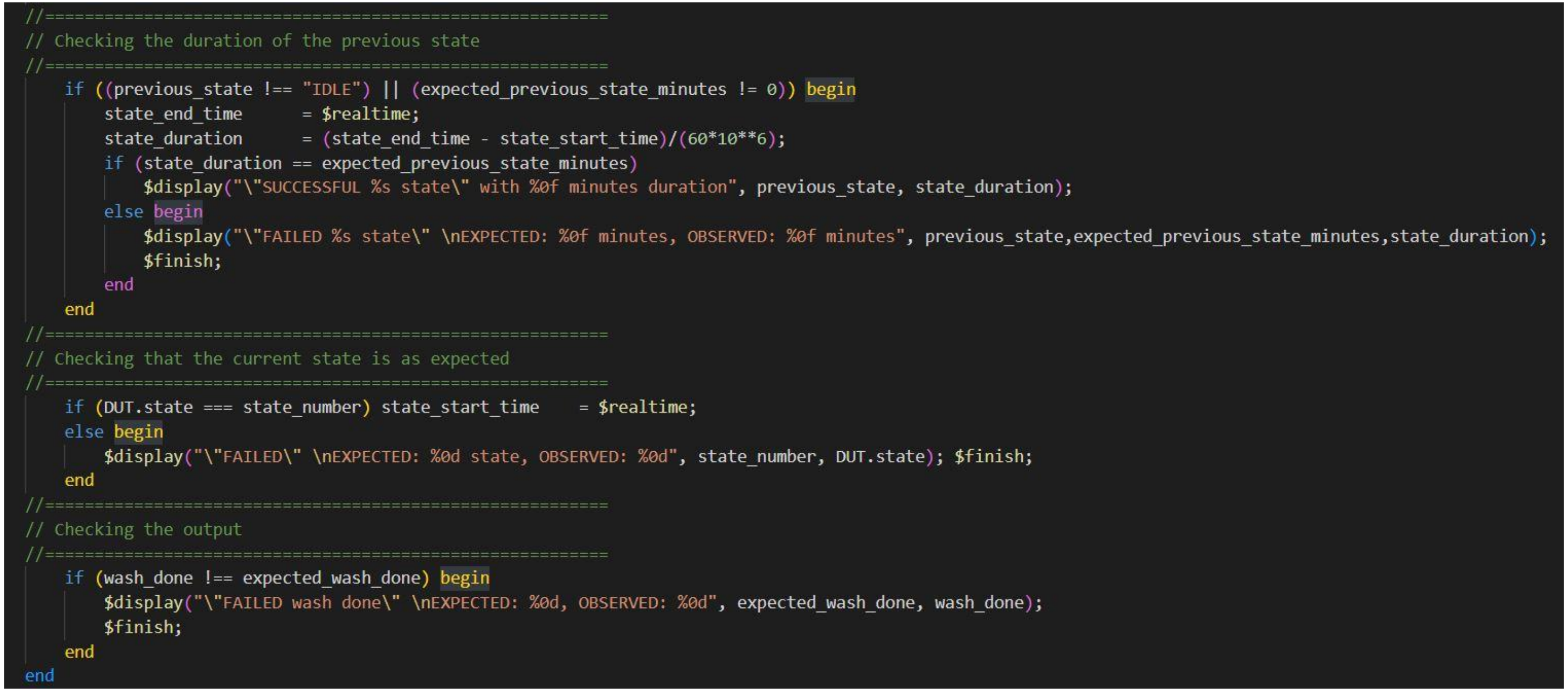
Monitoring part of generic_test

Verification: Testbench Explanation

Checkers Implementation

- One checker calculates the duration of the previous state by storing the start and end time using **$realtime** system task.
- Another checker verifies that the current state is the correct one in the operation sequence.
- The final checker verifies that the wash_done signal is as expected in each state.

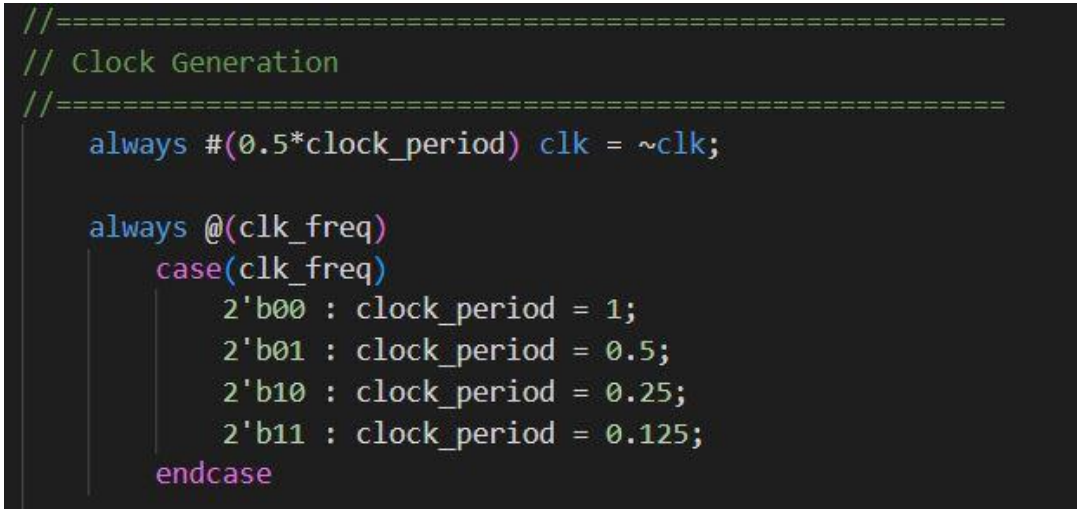
Verification: Testbench Explanation



```
//==========================================================
// Checking the duration of the previous state
//==========================================================
    if ((previous_state !== "IDLE") || (expected_previous_state_minutes != 0)) begin
        state_end_time      = $realtime;
        state_duration      = (state_end_time - state_start_time)/(60*10**6);
        if (state_duration == expected_previous_state_minutes)
            $display("\"SUCCESSFUL %s state\" with %0f minutes duration", previous_state, state_duration);
        else begin
            $display("\"FAILED %s state\" \nEXPECTED: %0f minutes, OBSERVED: %0f minutes", previous_state,expected_previous_state_minutes,state_duration);
            $finish;
        end
    end
//==========================================================
// Checking that the current state is as expected
//==========================================================
    if (DUT.state === state_number) state_start_time    = $realtime;
    else begin
        $display("\"FAILED\" \nEXPECTED: %0d state, OBSERVED: %0d", state_number, DUT.state); $finish;
    end
//==========================================================
// Checking the output
//==========================================================
    if (wash_done !== expected_wash_done) begin
        $display("\"FAILED wash done\" \nEXPECTED: %0d, OBSERVED: %0d", expected_wash_done, wash_done);
        $finish;
    end
end
```

Checkers Implementation

Verification: Testbench Explanation



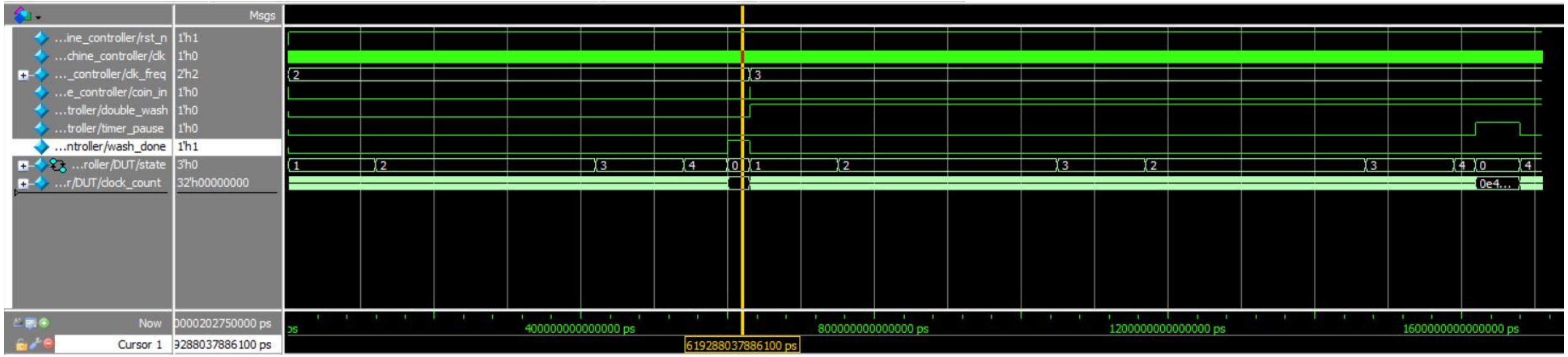
```
//==========================================================
// Clock Generation
//==========================================================
    always #(0.5*clock_period) clk = ~clk;

    always @(clk_freq)
        case(clk_freq)
            2'b00 : clock_period = 1;
            2'b01 : clock_period = 0.5;
            2'b10 : clock_period = 0.25;
            2'b11 : clock_period = 0.125;
        endcase
```

Input Clock Implementation

Verification: Simulation Results

Simulation Waveform

- The first test is shown before the yellow cursor, the second test is shown after the yellow cursor, and test are executed in series, so both tests combined form the third test.



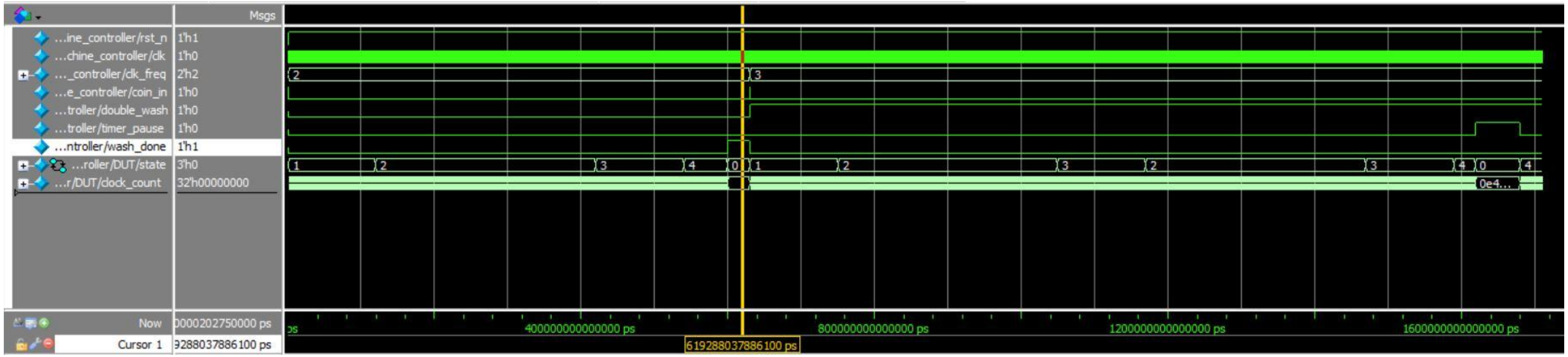
```
                          Msgs
...ine_controller/rst_n    1'h1
...chine_controller/clk    1'h0
..._controller/clk_freq    2'h2
...e_controller/coin_in    1'h0
...troller/double_wash     1'h0
...troller/timer_pause     1'h0
...ntroller/wash_done      1'h1
...roller/DUT/state        3'h0
...r/DUT/clock_count       32'h00000000

Now        0000202750000 ps
Cursor 1   9288037886100 ps
```

Simulation Waveform

Verification: Simulation Results

Simulation Waveform

- The waveform shows the sequence of state transitions and the duration of each state.
- It also shows the waveform of the output in each state



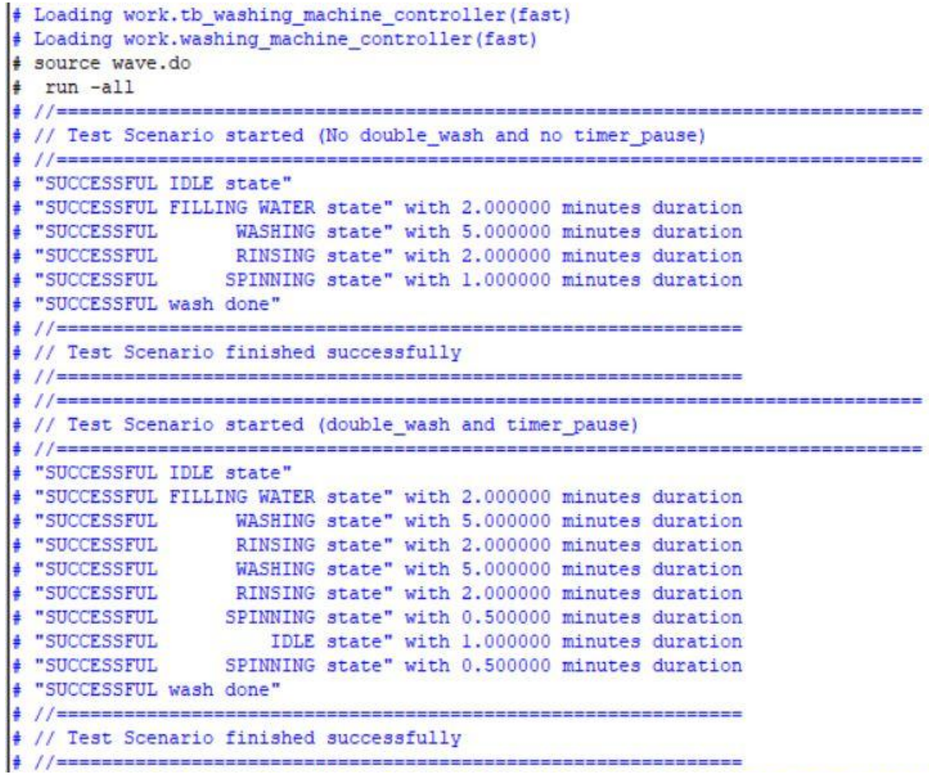
	Msgs
...ine_controller/rst_n	1'h1
...chine_controller/clk	1'h0
..._controller/clk_freq	2'h2
...e_controller/coin_in	1'h0
...troller/double_wash	1'h0
...troller/timer_pause	1'h0
...ntroller/wash_done	1'h1
...roller/DUT/state	3'h0
...r/DUT/clock_count	32'h00000000
Now	000202750000 ps
Cursor 1	9288037886100 ps

Simulation Waveform

Verification: Simulation Results

Testbench output

The figure shows the result of the checkers implemented in the testbench.



```
# Loading work.tb_washing_machine_controller(fast)
# Loading work.washing_machine_controller(fast)
# source wave.do
#  run -all
# //===============================================================================
# // Test Scenario started (No double_wash and no timer_pause)
# //===============================================================================
# "SUCCESSFUL IDLE state"
# "SUCCESSFUL FILLING WATER state" with 2.000000 minutes duration
# "SUCCESSFUL       WASHING state" with 5.000000 minutes duration
# "SUCCESSFUL       RINSING state" with 2.000000 minutes duration
# "SUCCESSFUL      SPINNING state" with 1.000000 minutes duration
# "SUCCESSFUL wash done"
# //=============================================================
# // Test Scenario finished successfully
# //=============================================================
# //===============================================================================
# // Test Scenario started (double_wash and timer_pause)
# //===============================================================================
# "SUCCESSFUL IDLE state"
# "SUCCESSFUL FILLING WATER state" with 2.000000 minutes duration
# "SUCCESSFUL       WASHING state" with 5.000000 minutes duration
# "SUCCESSFUL       RINSING state" with 2.000000 minutes duration
# "SUCCESSFUL       WASHING state" with 5.000000 minutes duration
# "SUCCESSFUL       RINSING state" with 2.000000 minutes duration
# "SUCCESSFUL      SPINNING state" with 0.500000 minutes duration
# "SUCCESSFUL          IDLE state" with 1.000000 minutes duration
# "SUCCESSFUL      SPINNING state" with 0.500000 minutes duration
# "SUCCESSFUL wash done"
# //=============================================================
# // Test Scenario finished successfully
# //=============================================================
```

Verification

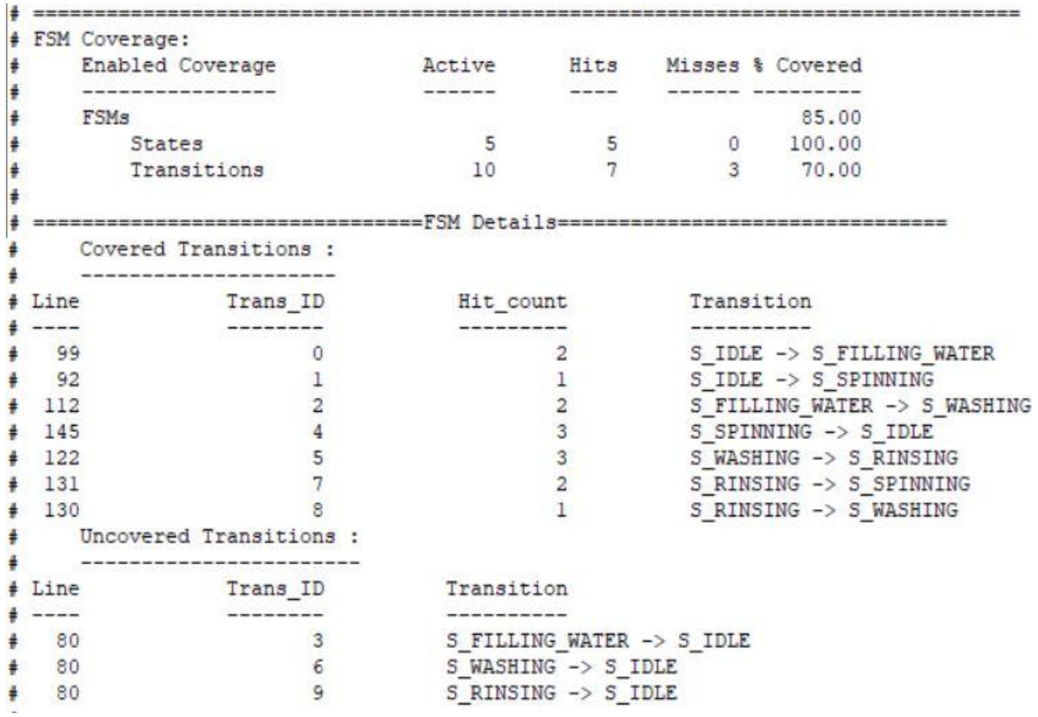
Traceability Matrix

- It shows the possible combinations covered by the test scenarios

Test Name	rst_n		double_wash		timer_pause		clk_freq				coin_in
	reset	No reset	Enabled	Disabled	Enabled	Disabled	1 MHz	2 MHz	4 MHz	8 MHz	enabled
Test01	✔			✔		✔	✔		✔		✔
Test02		✔	✔		✔			✔		✔	✔

Note: each test was used twice with two different frequencies.

Verification: FSM Coverage Report



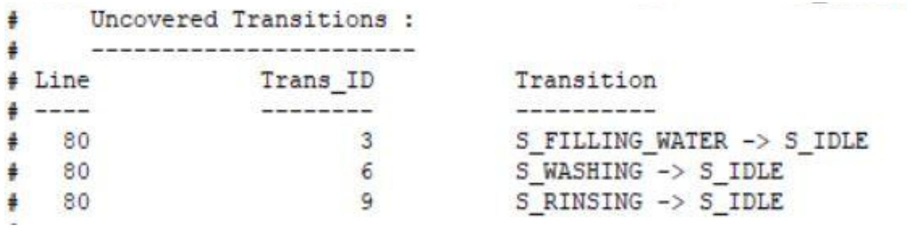
```
# ================================================================================
# FSM Coverage:
#     Enabled Coverage              Active      Hits    Misses % Covered
#     ----------------              ------      ----    ------ ---------
#     FSMs                                                         85.00
#         States                         5         5         0    100.00
#         Transitions                   10         7         3     70.00
#
# ================================FSM Details================================
#     Covered Transitions :
#     ---------------------
# Line             Trans_ID           Hit_count           Transition
# ----             --------           ---------           ----------
#   99                    0                   2           S_IDLE -> S_FILLING_WATER
#   92                    1                   1           S_IDLE -> S_SPINNING
#  112                    2                   2           S_FILLING_WATER -> S_WASHING
#  145                    4                   3           S_SPINNING -> S_IDLE
#  122                    5                   3           S_WASHING -> S_RINSING
#  131                    7                   2           S_RINSING -> S_SPINNING
#  130                    8                   1           S_RINSING -> S_WASHING
#     Uncovered Transitions :
#     -----------------------
# Line             Trans_ID          Transition
# ----             --------          ----------
#   80                    3          S_FILLING_WATER -> S_IDLE
#   80                    6          S_WASHING -> S_IDLE
#   80                    9          S_RINSING -> S_IDLE
```

FSM Coverage Report

Verification: FSM Coverage Report

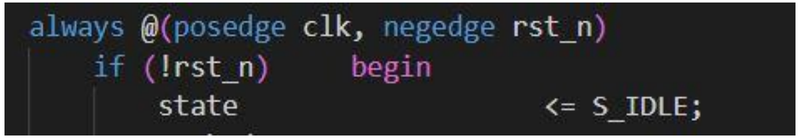
FSM Coverage Analysis

- All the possible state transitions during the normal operation was covered by the test scenarios.
- However, the reason for the three uncovered state transitions is that the reset signal puts the machine into the idle state from any state
- so a test that resets at S_FILLING_WATER, S_WASHING, S_RINSING states will cover the remaining transitions.



```
#     Uncovered Transitions :
#     ------------------------
# Line            Trans_ID          Transition
# ----            --------          ----------
#   80                   3          S_FILLING_WATER -> S_IDLE
#   80                   6          S_WASHING -> S_IDLE
#   80                   9          S_RINSING -> S_IDLE
```

Uncovered State Transitions

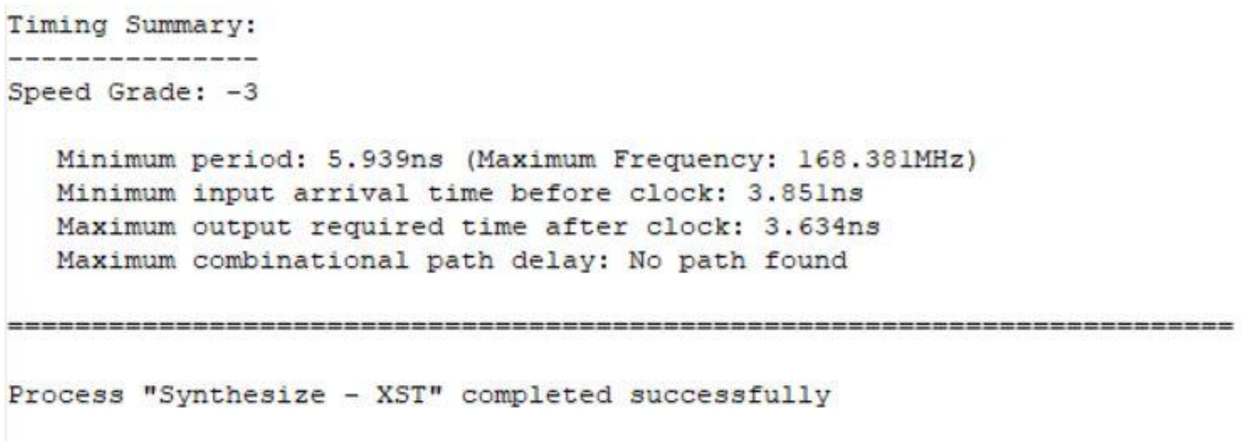


```
always @(posedge clk, negedge rst_n)
    if (!rst_n)     begin
        state                   <= S_IDLE;
```

The Source of uncovered state transitions

Synthesis Result

- The RTL code was synthesized successfully using Xilinx ISE and Spartan 6 FPGA.



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 5.939ns (Maximum Frequency: 168.381MHz)
   Minimum input arrival time before clock: 3.851ns
   Maximum output required time after clock: 3.634ns
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully
```

Synthesis Results

Thank You!