Washing Machine Controller

Designed by: Yousef Sherif

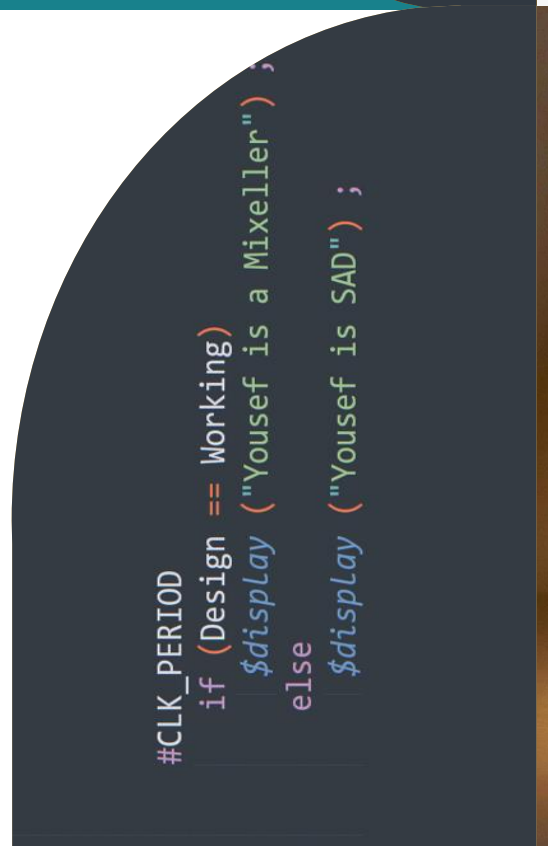
Content

1) Requirements

2) Solution and Design Flow

3) Detailed Explanation of My Code

4) The Testing Covered Scenarios

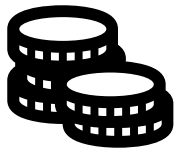


Requirements

Requirements

What is the washing machine supposed to do?

Requirements



- The machine starts when a coin is deposited.
- There is a double wash button. When it is pressed, a second wash and rinse to occur after completing the first rinse.
- There is a pause button. When it is pressed, during the spin cycle, the machine stops spinning until the button is pressed again.

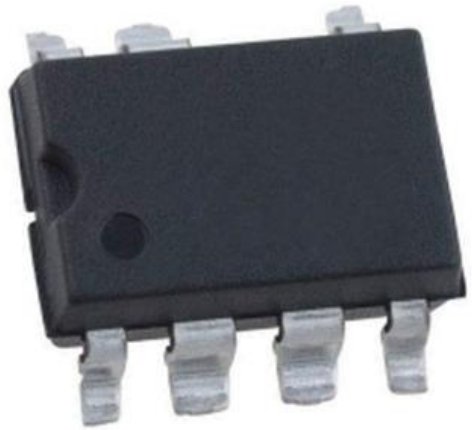
Requirements

Each state of this machine takes a time period as stated below

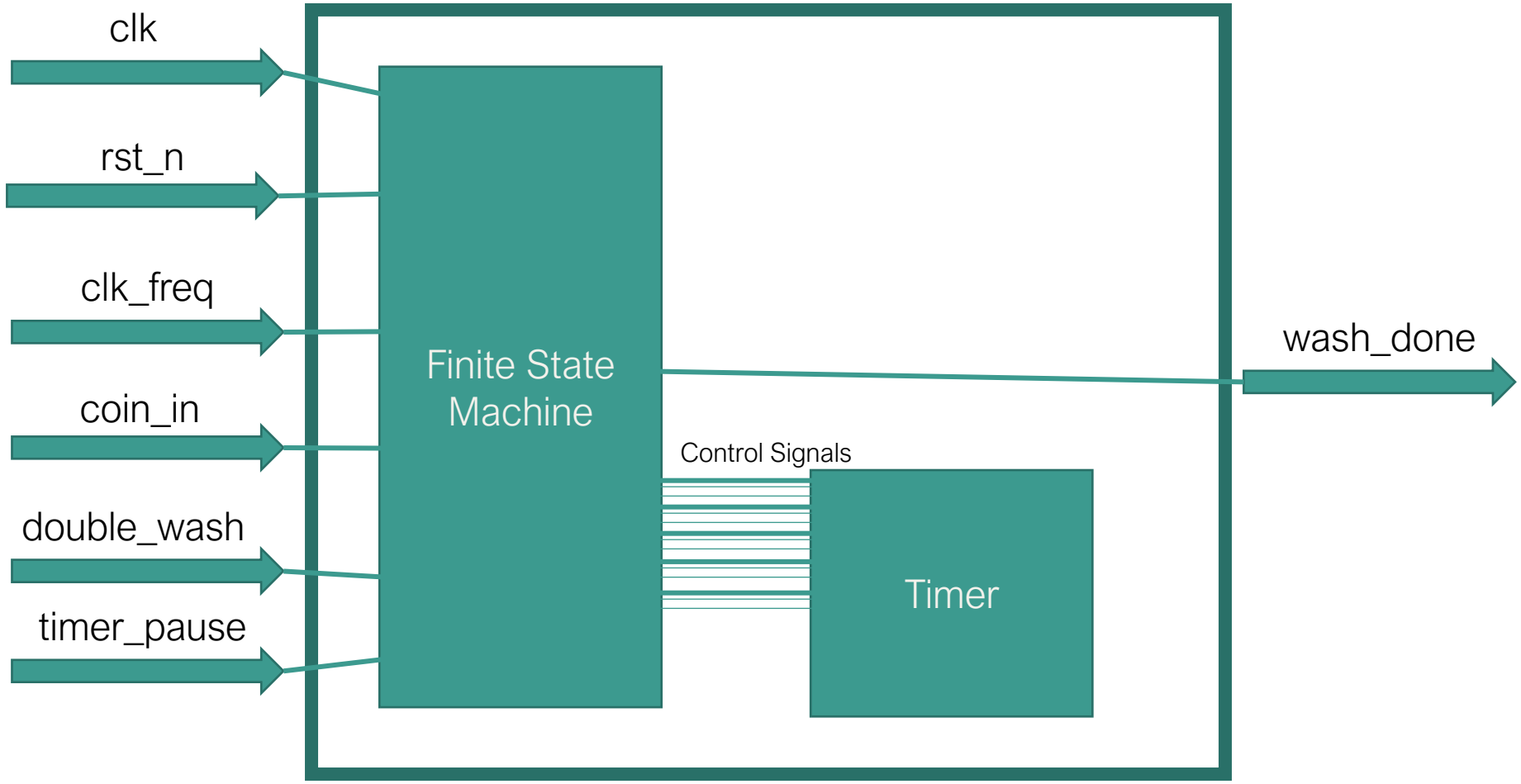
Solution and Design flow

Solution and Design flow

The solution is to design a controller for the washing machine to operate as intended.



Solution and Design flow

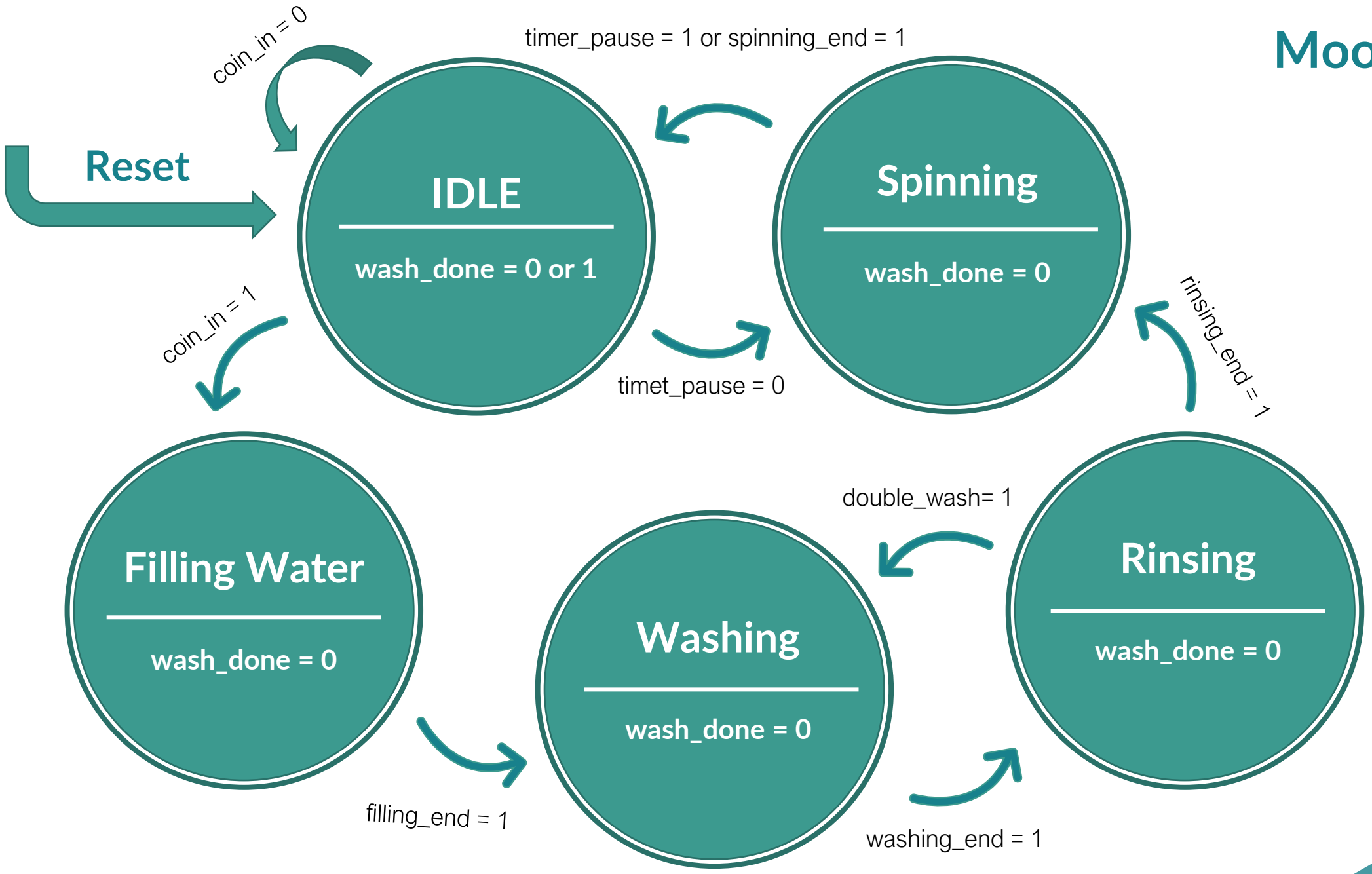


Solution and Design flow

Control signals are some flags to indicate the flow of the operation and the start and end of each state:

State	Control Signals
Filling	filling_start
	filling_end
Washing	washing_start
	washing_end
Rinsing	rinsing_start
	rinsing_end
	double_wash_done
Spinning	spinning_start
	spinning_end
	timer_pause_start

Moore FSM


Reset
IDLE
wash_done = 0 or 1
Spinning
wash_done = 0
Rinsing
wash_done = 0
Washing
wash_done = 0
Filling Water
wash_done = 0
coin_in = 0
timer_pause = 1 or spinning_end = 1
coin_in = 1
timet_pause = 0
rinsing_end = 1
double_wash= 1
filling_end = 1
washing_end = 1

Detailed explanation of my code

```
g machine, asynchronous clear
-----------------------------
ll rights reserved.
-----------------------------

////////////////////////////////////////
/// Detailed explanation of my code ////
////////////////////////////////////////
module Mixel (

 input      wire                    clk
 input      wire                    rst_n
 input      wire      [1:0]         clk_fre
 input      wire                    coin_in
 input      wire                    double_
 input      wire                    timer_p
 output     reg                     wash_do
);

//////////////////////////
/////////////////////////
//////////////////////////
    3'b000 ,
    3'b001 ,
```

File Header

The headers include the name of the file, author, and author's email address, a purpose section explaining the functionality of the module.

```
1  // FILE NAME: washing_machine.v
2  // TYPE: module
3  // AUTHOR: Yousef Sherif
4  // AUTHOR'S EMAIL: shirefy49@gmail.com
5  //----------------------------------------------------------------------
6  // PURPOSE: Digital Design Assignment
7  //----------------------------------------------------------------------
8  // KEYWORDS: Controller unit for a washing machine, asynchronous clear
9  //----------------------------------------------------------------------
10 // Copyright 2022, Yousef Sherif, All rights reserved.
11 //----------------------------------------------------------------------
12
```

Module Definition

Each module must have a module_name, which is the identifier for the module, and a port list, which describes the input and output terminals of the module.

```
13 ////////////////////////////////////////////////////////
14 ////////////////// Module Difinition ///////////////////
15 ////////////////////////////////////////////////////////
16 module washing_machine (
17
18  input       wire                clk         ,
19  input       wire                rst_n       ,
20  input       wire      [1:0]     clk_freq    ,
21  input       wire                coin_in     ,
22  input       wire                double_wash ,
23  input       wire                timer_pause ,
24  output      reg                 wash_done
25 );
```

States Encoded in Gray Encoding

This block of code creates the state variables. This way each state can be referenced by name.

localparam prevents the values to be overwritten (directly) from outside the module. And, it gives descriptive names to the states.

Gray code consists of a sequence where only one bit changes between one value and the next. In addition to also using the minimum number of bits, this encoding minimizes dynamic power consumption if the sequence of states is followed optimally.

```
26
27  ////////////////////////////////////////////////////////
28  /////////// States Encoded in Gray Encoding ////////////
29  ////////////////////////////////////////////////////////
30   localparam          IDLE          = 3'b000 ,
31                       FILLING_WATER = 3'b001 ,
32                       WASHING       = 3'b011 ,
33                       RINSING       = 3'b010 ,
34                       SPINNING      = 3'b110 ;
35
```

Control Signals

This block of code creates the state registers that hold the next_state and present_state variables. This is which circle you are on the state diagram.

States control signals are some flags to indicate the flow of the operation and the start and end of each state.

They are defined as reg because they are assigned variables inside an always block.

```
36  ////////////////////////////////////
37  ////////////////// Control Signals ///
38  ////////////////////////////////////
39   reg  [2:0]  current_state ,
40               next_state    ;
```

```
42  // States Control Signals
43   reg filling_start    ;
44   reg filling_end      ;
45   reg washing_start    ;
46   reg washing_end      ;
47   reg rinsing_start    ;
48   reg rinsing_end      ;
49   reg spinning_start   ;
50   reg spinning_end     ;
51   reg double_wash_done ;
52
53  // For Timer
54   reg [31:0] counter     ;
55   reg timer_pause_start  ;
```

State Transition

This is the state register. It moves the next_state to the current_state at the positive clock edge. This is what makes movement between the states in the state diagram happen.

Using an active low asynchronous reset signal, we can reset all the control signals, and make the current state IDLE at the negative edge of the clock.

```
57  ////////////////////////////////////////////////////////////////
58  ////////////////// State Transition ////////////////////////////
59  ////////////////////////////////////////////////////////////////
60  always @(posedge clk or negedge rst_n) begin
61    if(!rst_n) begin
62          current_state <= IDLE ;
63          filling_start <= 1'b0  ;
64          filling_end   <= 1'b0  ;
65          washing_start <= 1'b0  ;
66          washing_end   <= 1'b0  ;
67          rinsing_start <= 1'b0  ;
68          rinsing_end   <= 1'b0  ;
69          spinning_start     <= 1'b0 ;
70          spinning_end       <= 1'b0 ;
71          timer_pause_start  <= 1'b0 ;
72          double_wash_done   <= 1'b0 ;
73          counter    <= 32'b0 ;
74      end
75    else begin
76        current_state <= next_state ;
77      end
78  end
```

Next State Logic

Here, I am using a combinational always block to move between different states.

```
69  ////////////////////////////////////////////////////////////////
70  ////////////////// Next State Logic ////////////////////////////
71  ////////////////////////////////////////////////////////////////
72  always @(*) begin
73
74   case (current_state)
75     IDLE : begin
```

Next State Logic

This is the first chunk of the state logic. It shows, at first, if you are in the IDLE state, then the spinning_end signal (which is a signal that is set to 1 by the timer at the end of the spinning state) is zero, and the timer_pause_start signal (which is a signal that is set to 1 when the timer_pause button is pressed during the spinning state) is zero, then if there is a coin deposited in the machine, it will move to the filling water state, else, it will stay at the IDLE state waiting for a coin to be deposited. The part of code that contain the timer_pause_start, and spinning_end signals is related to other states, and I will talk more about them is the following slides.

```
75    IDLE : begin
76        if (spinning_end) begin
77            next_state = IDLE ;
78        end
79        else begin
80          if(timer_pause_start) begin
81            if(timer_pause) begin
82              next_state = IDLE ;
83            end
84          else begin
85              next_state = SPINNING ;
86            end
87        end
88      else begin
89        if(coin_in) begin
90            filling_start = 1'b1 ;
91            next_state = FILLING_WATER ;
92          end
93        else begin
94            next_state = IDLE ;
95          end
96        end
97        end
98      end
```

Next State Logic

This is the second chunk of the state logic. It shows, if you are in the FILLING_WATER state, and the filling is done, so, the filling_end signal is set to one by the timer, then the next state is the WASHING state, and the washing_start signal is set to one, so that the timer can start counting the time for this operation.

If the filling is not finished yet (filling_end signal is zero), the machine will stay in the FILLING_WATER state.

```
 99      FILLING_WATER : begin
100       if(filling_end) begin
101           washing_start = 1'b1 ;
102           washing_end   = 1'b0 ;
103           next_state = WASHING ;
104         end
105       else begin
106         filling_start = 1'b1 ;
107         next_state = FILLING_WATER ;
108         end
109       end
```

Next State Logic

This is the third chunk of the state logic. It shows, if you are in the WASHING state, and the washing is finished (washing_end signal is set to one by the timer), then the next state is the RINSING state, and the rinsing_start signal is set to one so that the timer starts counting the time for this operation.

If the washing is not finished yet (washing_end signal is zero), the machine will stay in the WASHING state.

```
110     WASHING : begin
111       if(washing_end) begin
112           rinsing_start = 1'b1 ;
113           rinsing_end   = 1'b0 ;
114           next_state = RINSING ;
115         end
116       else begin
117         washing_start = 1'b1 ;
118         next_state = WASHING ;
119         end
120       end
```

Next State Logic

This is the fourth chunk of the state logic. It shows, if you are in the RINSING state, and the rinsing is finished (rinsing_end signal is set to one by the timer), then the next state will depend on whether the double_wash button is pressed or not, if it is pressed, and the double_wash_done is initially set to 0, then the next state will be the washing state again, and the double_wash_done signal is set to 1 to ensure that the double whashing process will not accur when returning from the double wash to the rinsing. After the double wash is done, the rinsing_end is set to 1 by the timer, and the double_wash_done is set to 1, then the next state will be the spinning state, else, it will stay in the rinsing state.

```
119     RINSING : begin
120       if(rinsing_end) begin
121         if(double_wash && !double_wash_done) begin
122             double_wash_done = 1'b1 ;
123             next_state = FILLING_WATER ;
124           end
125         else begin
126             spinning_start = 1'b1;
127             next_state = SPINNING ;
128           end
129         end
130       else begin
131           rinsing_start = 1'b1 ;
132           next_state = RINSING ;
133         end
134     end
```

Next State Logic

This is the fifth chunk of the state logic. It shows, if you are in the SPINNING state, and the timer_pause button is not pressed, then the timer_pause signal is set to 0, and the next state will be the IDLE state, and the machine will stay in the IDLE state until the reset is done. But, if the timer pause button is pressed during spinning, then the timer_pause_start signal is set to 1, and the machine will top spinning by entering the IDLE state.

```
135     SPINNING : begin
136       if(timer_pause) begin
137             timer_pause_start = 1'b1 ;
138             next_state = IDLE ;
139         end
140       else begin
141           if(spinning_end) begin
142               next_state = IDLE ;
143           end
144         end
145       end
```

```
75     IDLE : begin
76         if (spinning_end) begin
77             next_state = IDLE ;
78         end
79         else begin
80           if(timer_pause_start) begin
81             if(timer_pause) begin
82               next_state = IDLE ;
83             end
84           else begin
85               next_state = SPINNING ;
86             end
87         end
88       else begin
89         if(coin_in) begin
90             filling_start = 1'b1 ;
91             next_state = FILLING_WATER ;
92           end
93         else begin
94             next_state = IDLE ;
95           end
96         end
97         end
98       end
```

Next State Logic

In the IDLE state, the spinning_end signal is not set to 1 yet by the timer, as the spinning state is not finished yet, and the timer_pause_start and the timer_pause signals are both set to 1, then the machine will stay in the IDLE state until the pause button is left, and the timer_pause signal is 0. This way we ensure that the timer_pause signal has no effect on the machine at the beginning until the spinning state is reached.

```
135     SPINNING : begin
136       if(timer_pause) begin
137             timer_pause_start = 1'b1 ;
138             next_state = IDLE ;
139         end
140       else begin
141           if(spinning_end) begin
142               next_state = IDLE ;
143           end
144         end
145       end
```

```
75     IDLE : begin
76         if (spinning_end) begin
77             next_state = IDLE ;
78         end
79         else begin
80           if(timer_pause_start) begin
81             if(timer_pause) begin
82               next_state = IDLE ;
83             end
84           else begin
85               next_state = SPINNING ;
86             end
87         end
88       else begin
89         if(coin_in) begin
90             filling_start = 1'b1 ;
91             next_state = FILLING_WATER ;
92           end
93         else begin
94             next_state = IDLE ;
95           end
96         end
97         end
98       end
```

Next State Logic

This is the default case of the case statement, and it is used so that there will be no unintentional latches in the design.

```
158     default : begin
159         filling_start = 1'b0 ;
160         filling_end   = 1'b0 ;
161         washing_start = 1'b0 ;
162         washing_end   = 1'b0 ;
163         rinsing_start = 1'b0 ;
164         rinsing_end   = 1'b0 ;
165         spinning_start     = 1'b0 ;
166         spinning_end       = 1'b0 ;
167         timer_pause_start  = 1'b0 ;
168         double_wash_done   = 1'b0 ;
169         counter    = 32'b0 ;
170         next_state = IDLE ;
171       end
172     endcase
173 end
```

Timer

clk = 1MHz (T period = 1 us)	
Duration	Equivalent Clock Cycles
1 second	1,000,000 clock cycles
1 minute	60,000,000 clock cycles
2 minutes	120,000,000 clock cycles
5 minutes	300,000,000 clock cycles
clk = 2MHz (T period = 1/2 us)	
Duration	Equivalent Clock Cycles
1 second	2,000,000 clock cycles
1 minute	120,000,000 clock cycles
2 minutes	240,000,000 clock cycles
5 minutes	600,000,000 clock cycles

Timer

clk = 4MHz (T period = 1/4 us)	
Duration	Equivalent Clock Cycles
1 second	4,000,000 clock cycles
1 minute	240,000,000 clock cycles
2 minutes	480,000,000 clock cycles
5 minutes	1200,000,000 clock cycles
clk = 8MHz (T period = 1/8 us)	
Duration	Equivalent Clock Cycles
1 second	8,000,000 clock cycles
1 minute	480,000,000 clock cycles
2 minutes	960,000,000 clock cycles
5 minutes	2400,000,000 clock cycles

Timer

The largest number of clock cycles will be counted by the counter signal will be 2400,000,000 clock cycles (5 minutes) in case of operating at a frequency of 8 MHz.

So, the size of the counter will be 32 bits so that it can hold this large number.

Log2(2400,000,000) = 31.160 = 32 bits approximately.

```
42  // States Control Signals
43   reg filling_start    ;
44   reg filling_end      ;
45   reg washing_start    ;
46   reg washing_end      ;
47   reg rinsing_start    ;
48   reg rinsing_end      ;
49   reg spinning_start   ;
50   reg spinning_end     ;
51   reg double_wash_done ;
52
53  // For Timer
54   reg [31:0] counter     ;
55   reg timer_pause_start  ;
```

Timer

The timer is an always block that is sequential because I want to count the number of clock cycles equivalent to some duration of time. Inside this always block is a case statement which depends on the clk_freq. here is the first case which is operating at a frequency of 1 MHz. The first if statement will work at the starting of the filling state, and it will count 120,000,000 clock cycles (2 minutes) before setting the filling_start signal to zero, and the filling_end signal to one, which means that the next state will be the washing state according to the next state logic always block. The counter is set to zero before continuing to the next state.

```
175 ////////////////////////////////////////////////////////
176 ////////////////////// Timer ///////////////////////////
177 ////////////////////////////////////////////////////////
178 always @(posedge clk) begin
179  case (clk_freq)
180    2'b00 : begin // clk = 1 Mhz
181        if(filling_start && !wash_done) begin
182            counter <= counter + 1'b1;
183            if(counter == 32'd120000000) begin // 2 minutes = 120,000,000 * 1 us
184                filling_start <= 1'b0;
185                filling_end <= 1'b1;
186                counter <= 32'd0;
187              end
188          end
```

Timer

Here, is the same idea for the washing state, and the rinsing state. The difference is that each state will count different number of clock cycles than the other states.

```
190         else if(washing_start && !wash_done) begin
191             counter <= counter + 1'b1;
192             if(counter == 32'd300000000) begin // 5 minutes = 300,000,000 * 1 us
193                 washing_start <= 1'b0;
194                 washing_end <= 1'b1;
195                 counter <= 32'd0;
196               end
197           end
198 
199         else if(rinsing_start && !wash_done) begin
200             counter <= counter + 1'b1;
201             if(counter == 32'd120000000) begin // 2 minutes = 120,000,000 * 1 us
202                 rinsing_start <= 1'b0;
203                 rinsing_end <= 1'b1;
204                 counter <= 32'd0;
205               end
206           end
```

Timer

Same idea here. But, the difference is that the condition of the if statement contains the timer_pause signal as shown below, as I want to stop the timer when the timer_pause signal is set to one, then continuing the working of the timer after the timer_pause signal is set to 0.

```
208         else if(spinning_start && !timer_pause && !wash_done) begin
209             counter <= counter + 1'b1;
210             if(counter == 32'd60000000) begin // 1 minutes = 60,000,000 * 1 us
211                 spinning_start <= 1'b0;
212                 spinning_end <= 1'b1;
213                 counter <= 32'd0;
214               end
215           end
216 
217         else if(spinning_start && timer_pause && !wash_done) begin
218             counter <= counter + 1'b0;
219           end
220       end
```

Timer

The rest of the timer code is the same as before, the only difference is that I will work on the rest of the frequencies 2MHz, 4MHz, and 8MHz.

```
222     2'b01 : begin // clk = 2 Mhz
223         if(filling_start && !wash_done) begin
224             counter <= counter + 1'b1;
225             if(counter == 32'd240000000) begin // 2 minutes = 240,000,000 * 1/2 us
226                 filling_start <= 1'b0;
227                 filling_end <= 1'b1;
228                 counter <= 32'd0;
229               end
230           end
```

```
264     2'b10 : begin // clk = 4 Mhz
265         if(filling_start && !wash_done) begin
266             counter <= counter + 1'b1;
267             if(counter == 32'd480000000) begin // 2 minutes = 480,000,000 * 1/4 us
268                 filling_start <= 1'b0;
269                 filling_end <= 1'b1;
270                 counter <= 32'd0;
271               end
272           end
```

```
306     2'b11 : begin // clk = 8 Mhz
307         if(filling_start && !wash_done) begin
308             counter <= counter + 1'b1;
309             if(counter == 32'd960000000) begin // 2 minutes = 960,000,000 * 1/8 us
310                 filling_start <= 1'b0;
311                 filling_end <= 1'b1;
312                 counter <= 32'd0;
313               end
314           end
```

Output Logic

The output logic is a simple combinational circuit that will make the wash_done signal equal to 1 when the spinning operation is done, and the timer_pause signal is 0.

```
356  ////////////////////////////////////////////////////////
357  ////////////////// Output Logic ////////////////////////
358  ////////////////////////////////////////////////////////
359  always @(*) begin
360    wash_done  = 1'b0 ;
361
362  if(spinning_end && !timer_pause) begin
363      wash_done = 1'b1 ;
364    end
365  else begin
366      wash_done = 1'b0 ;
367    end
368  end
369
370  endmodule
```

```
#CLK_PERIOD
    if (Design == Working)
      $display ("Yousef is a Mixeller")
    else
      $display ("Yousef is SAD") ;
```

The testing covered scenarios

Testbench

Testing is done using a testbench Verilog code. It contains:

- File Header
- DUT Signals
- DUT Instantiation
- Clock Generation
- Tasks
- Initial Block

File Header

The header includes the name of the file, author, and author's email address, a purpose section explaining the functionality of the module.

```
1  // FILE NAME: washing_machine_tb.v
2  // TYPE: module
3  // AUTHOR: Yousef Sherif
4  // AUTHOR'S EMAIL: shirefy49@gmail.com
5  //------------------------------------------------------------------------
6  // PURPOSE: Digital Design Assignment Testbench
7  //------------------------------------------------------------------------
8  // KEYWORDS: Controller unit for a washing machine, asynchronous clear. Testbench
9  //------------------------------------------------------------------------
10 // Copyright 2022, Yousef Sherif, All rights reserved.
11 //------------------------------------------------------------------------
```

DUT Signals

Here, I define the testbench signals which will be connected to the DUT.

```
12  `timescale 1us/1ns
13  module washing_machine_tb() ;
14
15  ////////////////////////////////////////////////////////
16  ////////////////// DUT Signals ////////////////////////
17  ////////////////////////////////////////////////////////
18   reg                clk_tb         ;
19   reg                rst_n_tb       ;
20   reg      [1:0]     clk_freq_tb    ;
21   reg                coin_in_tb     ;
22   reg                double_wash_tb ;
23   reg                timer_pause_tb ;
24   wire               wash_done_tb   ;
25
```

DUT Instantiation

Here, I make an instance of the main design, and I am naming it DUT. Then, I am connecting the testbench signals to the DUT signals using port mapping by name to ensure that all signals are connected correctly.

```
105  ////////////////////////////////////////////////////////////////
106  ////////////////////// DUT Instantation ////////////////////////
107  ////////////////////////////////////////////////////////////////
108
109  washing_machine DUT (
110  .clk(clk_tb ),
111  .rst_n(rst_n_tb),
112  .clk_freq(clk_freq_tb),
113  .coin_in(coin_in_tb),
114  .double_wash(double_wash_tb),
115  .timer_pause(timer_pause_tb),
116  .wash_done(wash_done_tb));
```

Clock Generation

Here, I define different clocks to use in my simulation. Using the `timescale compiler directive means that all numbers after the (#) will be multiplied by 1 us. For example, in the first always statement, #0.5 means that 0.5 is half the T period, multiply the T period by 1 us from the timescale gives a T period of 1 us.

```
12  `timescale 1us/1ns
13  module washing_machine_tb() ;
```

```
110 ////////////////////////////////////////////////////////
111 ////////////////// Clock Generator  ////////////////////
112 ////////////////////////////////////////////////////////
113
114 always #0.5    clk_tb = ~clk_tb ;   // period = 1 us       (1 MHz)
115 //always #0.25   clk_tb = ~clk_tb ;   // period = 0.5 us     (2 MHz)
116 //always #0.125  clk_tb = ~clk_tb ;   // period = 0.25 us    (4 MHz)
117 //always #0.0625 clk_tb = ~clk_tb ;   // period = 0.125 us   (8 MHz)
118
```

Tasks

Here, I define different tasks to be uses in the initial block. The first task is Signals Initialization, and it is used to initialize the clk, clk_freq, coin_in, double_wash, and timer_pause to a known value, here, ther are all initialized to zero. The second task in called Reset, and it is used to reset the controller using the negative edge active low reset. #1 delay means 1 us delay in the waveform of the simulation.

```
66  ////////////////////////////////////////////////////////
67  ////////////////////// TASKS ///////////////////////////
68  ////////////////////////////////////////////////////////
69
70  /////////////// Signals Initialization /////////////////
71  task initialize;
72   begin
73    clk_tb         = 1'b0  ;
74    clk_freq_tb    = 2'b00 ;      // frequency is 1MHz
75    coin_in_tb     = 1'b0  ;      // initially coin is not deposited
76    double_wash_tb = 1'b0  ;      // initially double_wash button is not pressed
77    timer_pause_tb = 1'b0  ;      // initially timer_pause button is not pressed
78   end
79  endtask
80
81  ////////////////////// RESET ///////////////////////////
82  task reset;
83   begin
84   rst_n_tb =  'b1;
85   #1
86   rst_n_tb  = 'b0;
87   #1
88   rst_n_tb  = 'b1;
89   end
90  endtask
```

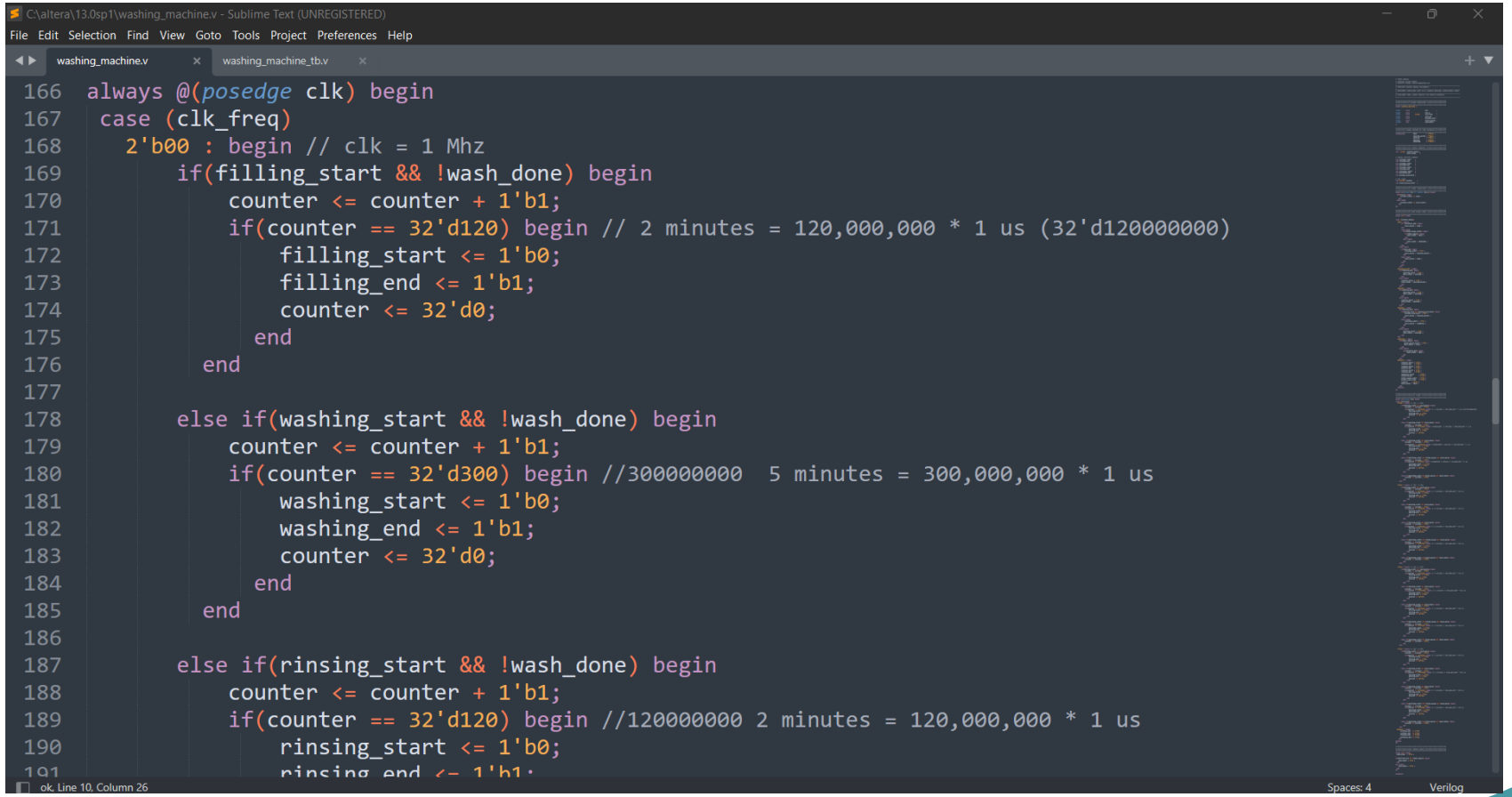
Tasks

The third task is double_wash, and it is used to initialize the clk, clk_freq, coin_in, double_wash, and timer_pause to a known value, here, ther are all initialized to zero. The second task in called Reset, and it is used to reset the controller using the negative edge active low reset.

```
92  ////////////////////// double_wash ////////////////////////
93  task double_wash;
94   begin
95   #1
96    double_wash_tb = 1'b1  ;      // double_wash button is pressed
97   end
98  endtask
99
100 ////////////////////// Timer_Pause ////////////////////////
101 task Timer_Pause;
102  begin
103  #1
104     timer_pause_tb = 1'b1  ;      // timer_pause button is pressed
105  #1
106     timer_pause_tb = 1'b0  ;      // timer_pause button is released
107  end
108 endtask
```

Simulation Note

Before Doing the simulation, and for the sake of making the simulation time shorter, I scaled down the time of each operation, for example, the time for the filling state is scaled down to 120 from 120,000,000.



Test Cases

I am going to make 6 different test simulation scenarios.

Test Case 1:
Here, we are testing the design knowing that the double wash and timer pause buttons are not pressed. Next are the simulation results from ModelSim software. (1 MHz)

Test Case 2:
Here, we are testing the design knowing that the double wash button is pressed. But, the timer pause button is not pressed. (1 MHz)

Test Case 3:
Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed. Next are the simulation results from ModelSim software. (1 MHz)

Test Cases

Test Case 4:
Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed.
Next are the simulation results from ModelSim software. (2 MHz)

Test Case 5:
Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed.
Next are the simulation results from ModelSim software. (4 MHz)

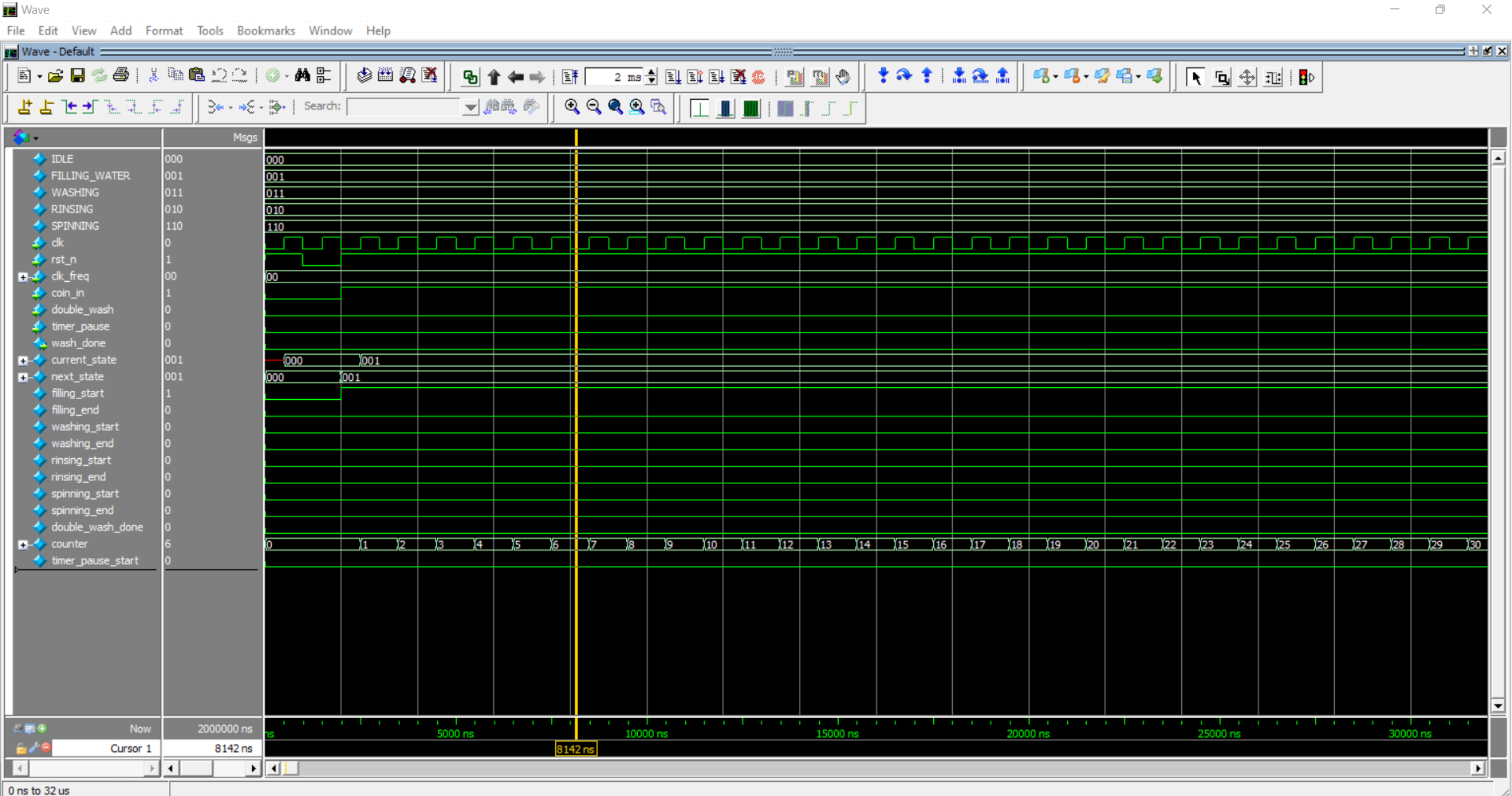
Test Case 6:
Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed.
Next are the simulation results from ModelSim software. (8 MHz)

Initial Block (Test Case 1)

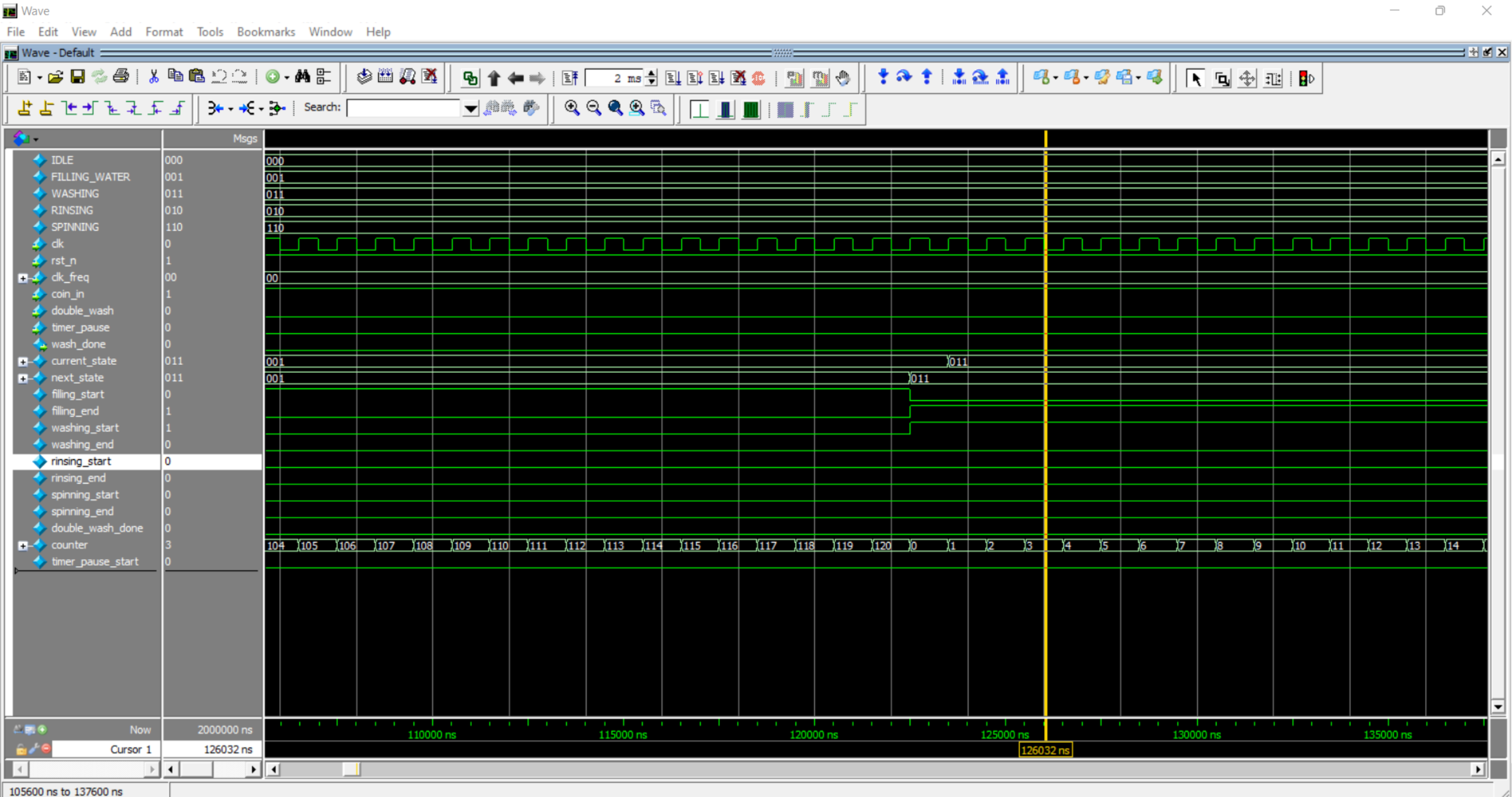
Here, we are testing the design knowing that the double wash and timer pause buttons are not pressed. Next are the simulation results from ModelSim software. (1 MHz)

```
26  ////////////////////////////////////////////////////////
27  ////////////////// initial block //////////////////////
28  ////////////////////////////////////////////////////////
29  initial begin
30
31   // Save Waveform
32     $dumpfile("washing_machine.vcd") ;
33     $dumpvars;
34
35   // initialization
36     initialize() ;
37
38   // Reset
39     reset() ;
40
41     coin_in_tb     = 1'b1  ;      // coin is deposited
42
43  #3000
44  $finish ;
45  end
```

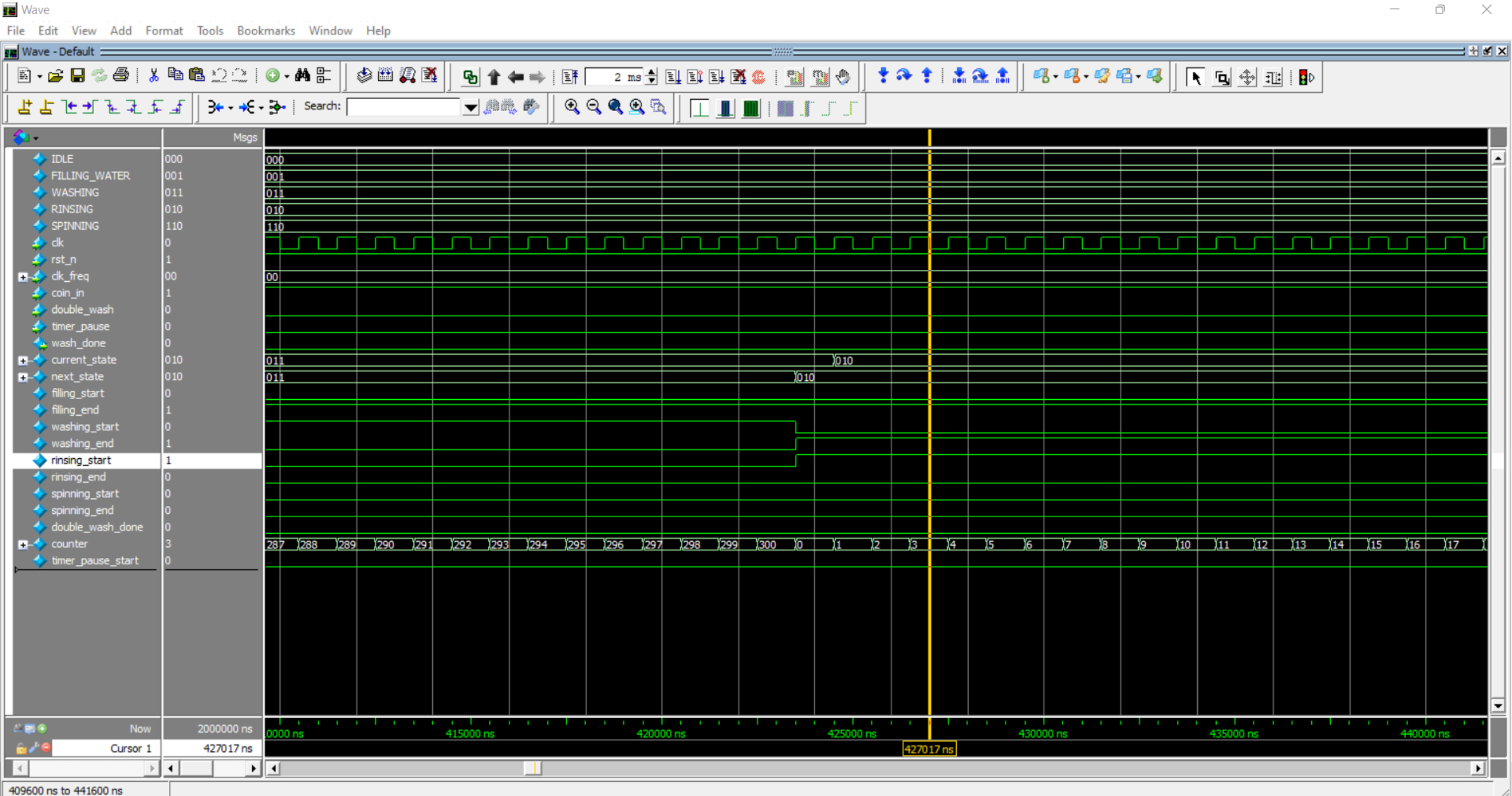
At the beginning.



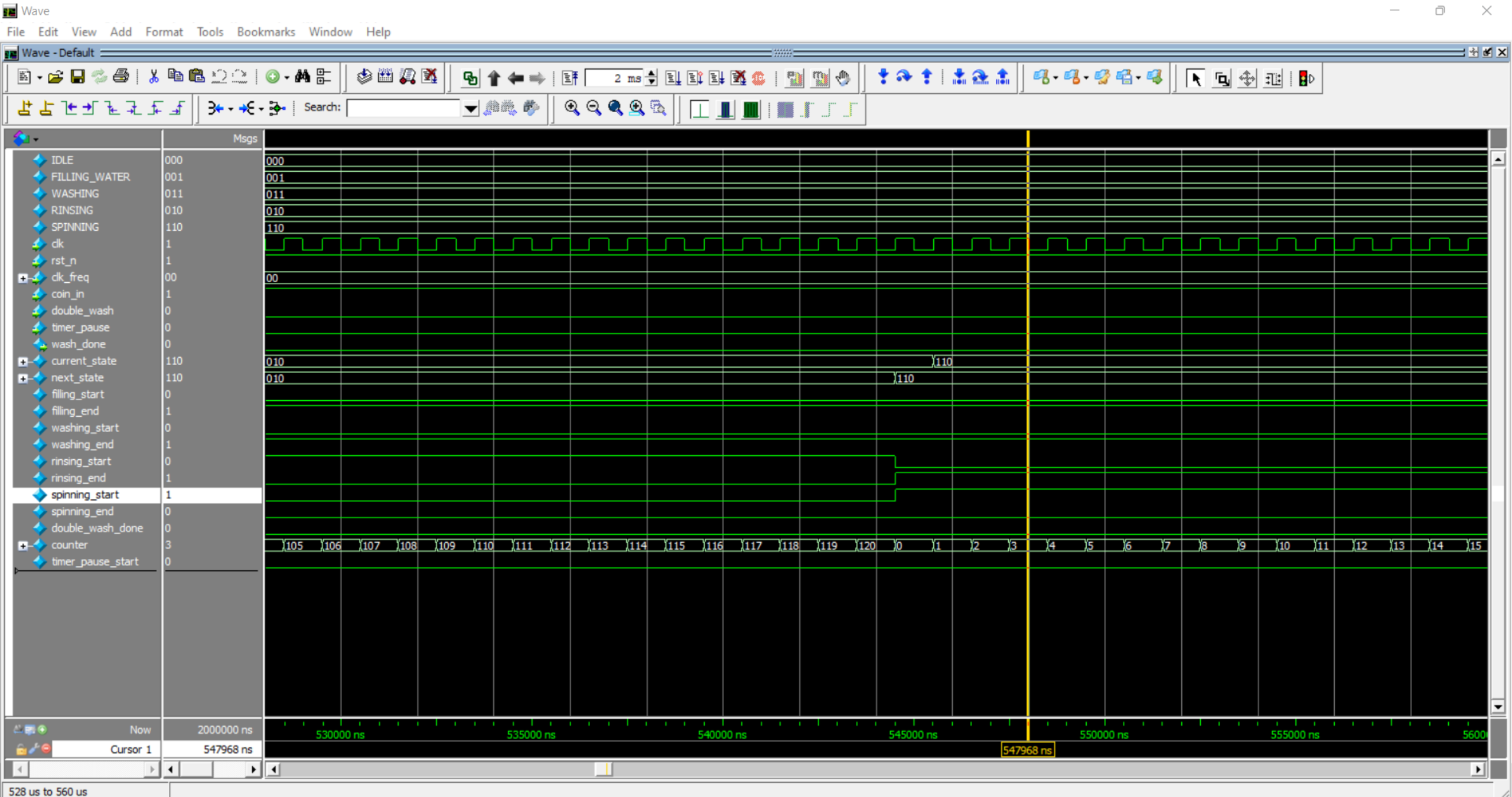
After the filling state is done. (2 Minutes = 120 clock cycles after scaling down)



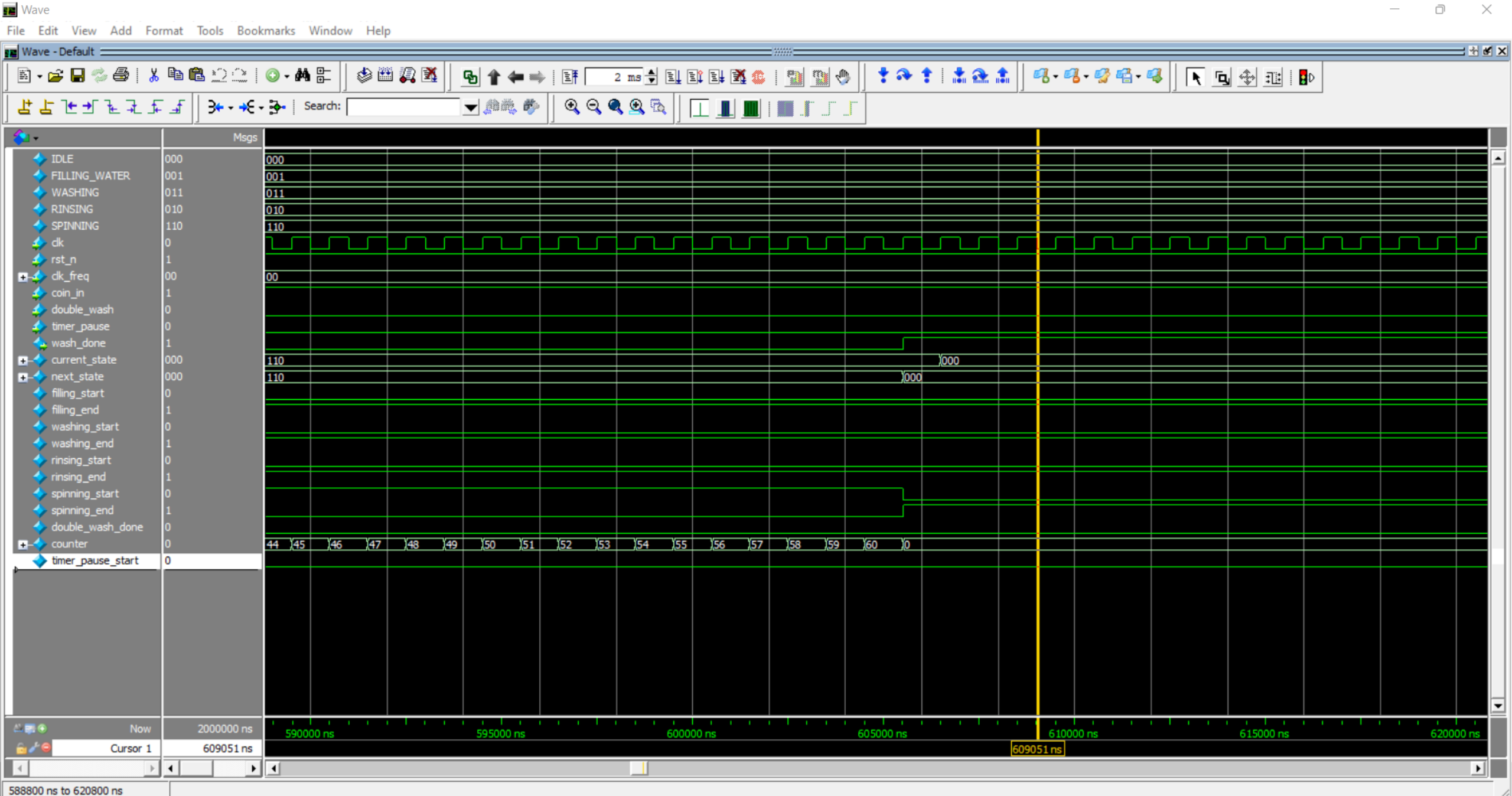
After the washing state is done. (5 Minutes = 300 clock cycles after scaling down)



After the rinsing state is done. (2 Minutes = 120 clock cycles after scaling down)



After the spinning state is done. (1 Minute = 60 clock cycles after scaling down)

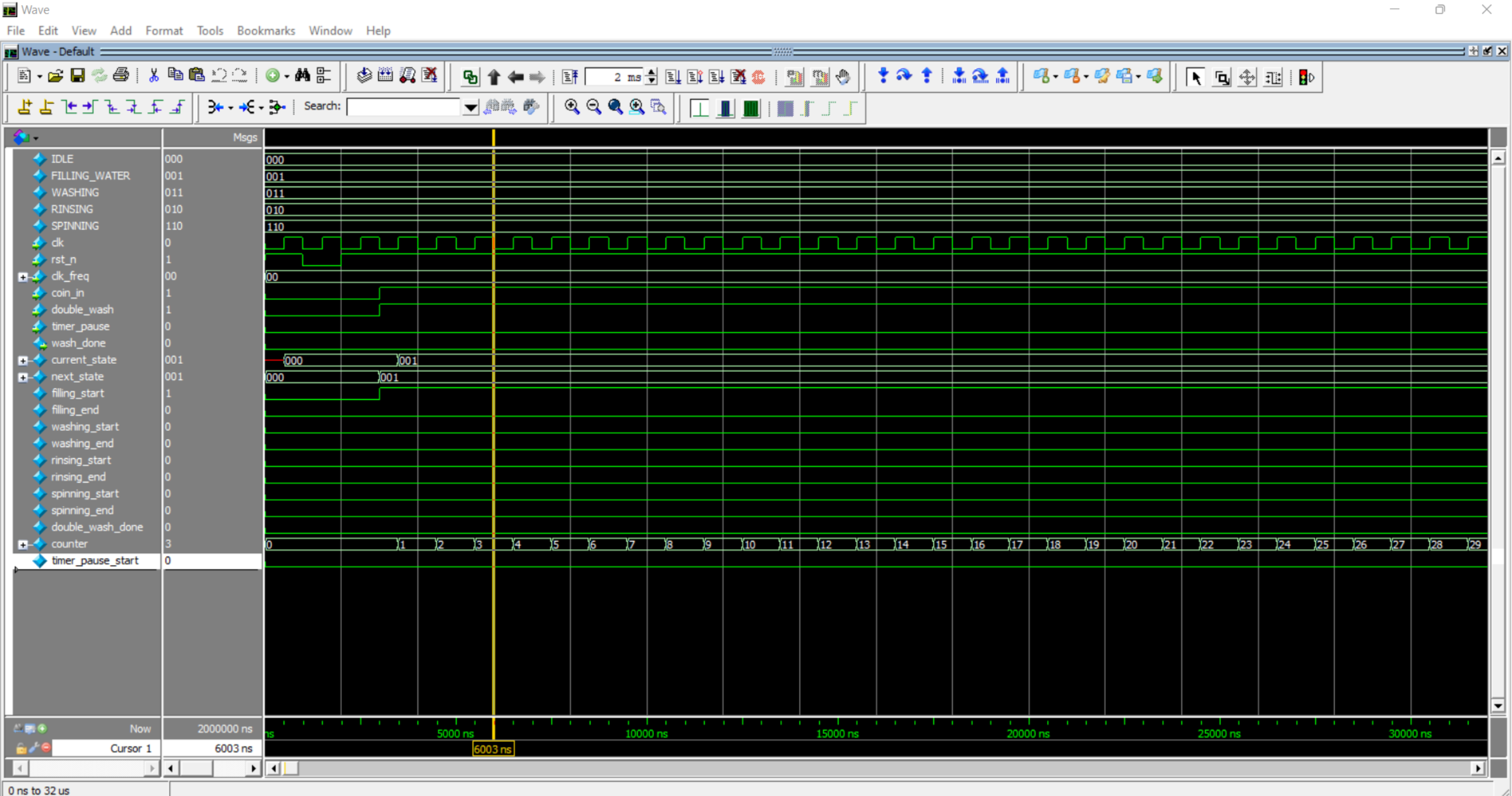


Initial Block (Test Case 2)

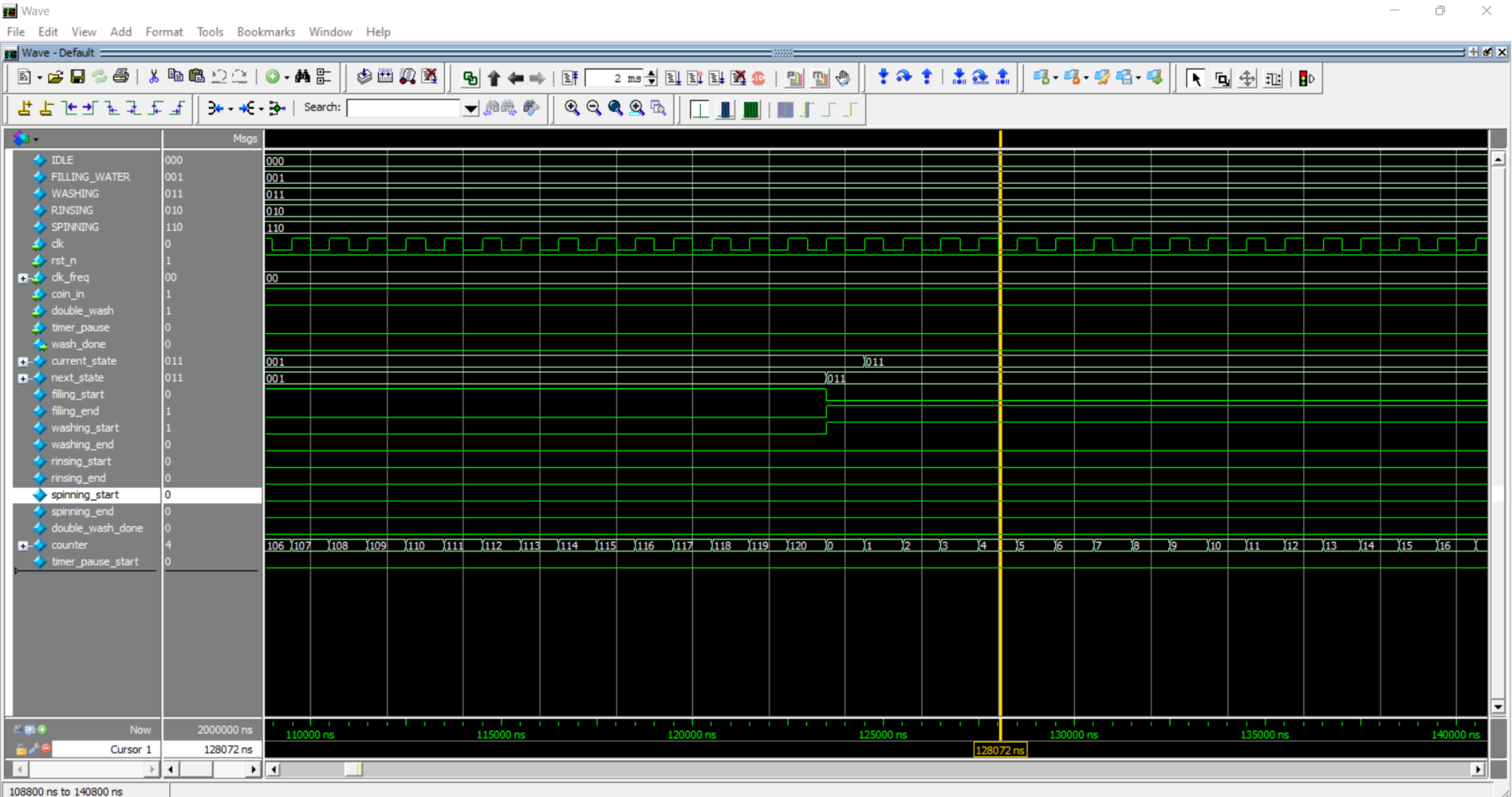
Here, we are testing the design knowing that the double wash button is pressed. But, the timer pause button is not pressed. (1 MHz)

```
26 ////////////////////////////////////////////////////////
27 ////////////////// initial block //////////////////////
28 ////////////////////////////////////////////////////////
29 initial begin
30
31  // Save Waveform
32    $dumpfile("washing_machine.vcd") ;
33    $dumpvars;
34
35  // initialization
36    initialize() ;
37
38  // Reset
39    reset() ;
40
41  // double_wash
42    double_wash() ;
43
44    coin_in_tb     = 1'b1  ;      // coin is deposited
45
46 #3000
47 $finish ;
48 end
```

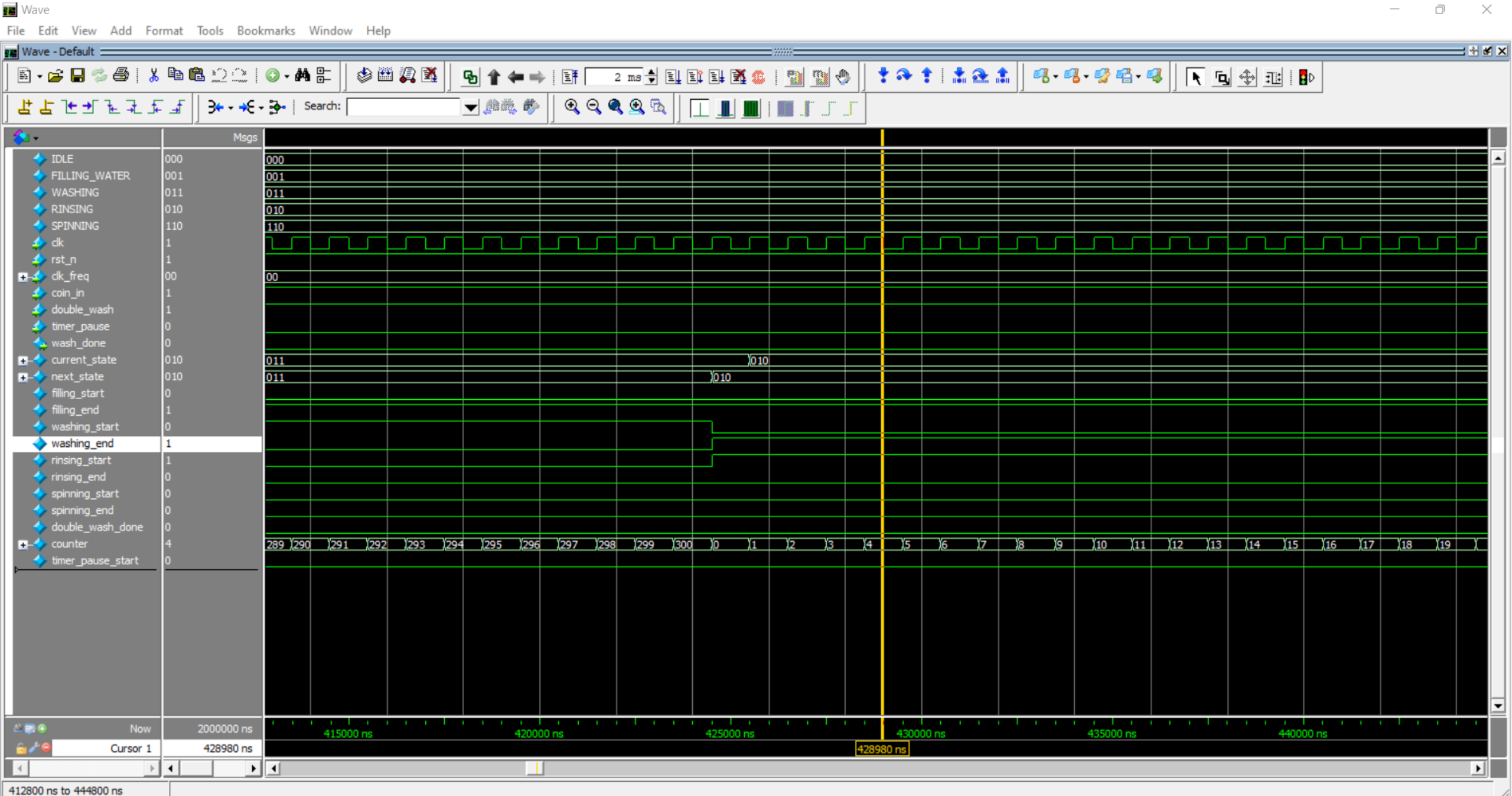
At the beginning.



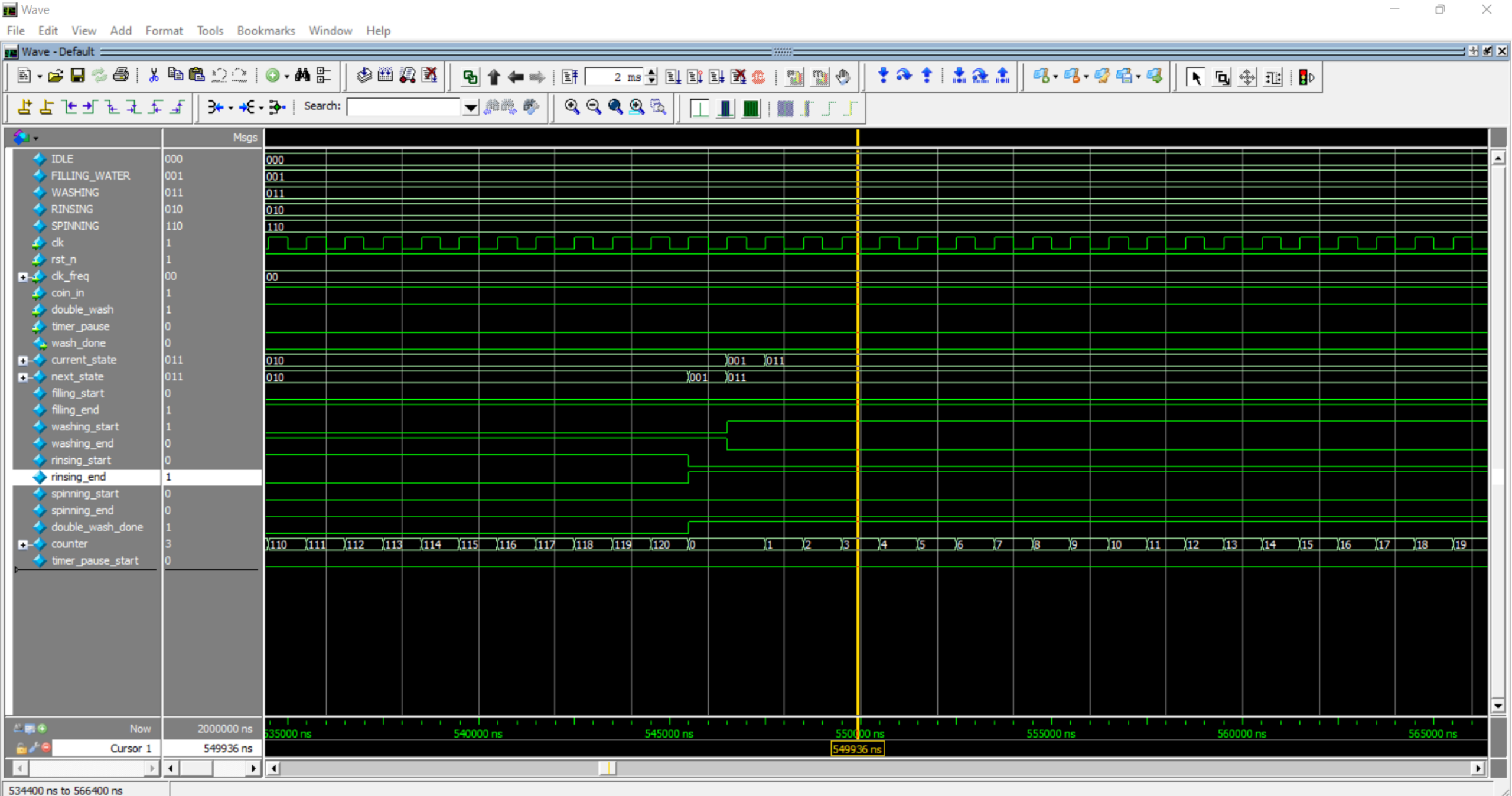
After the filling state is done. (2 Minutes = 120 clock cycles after scaling down)



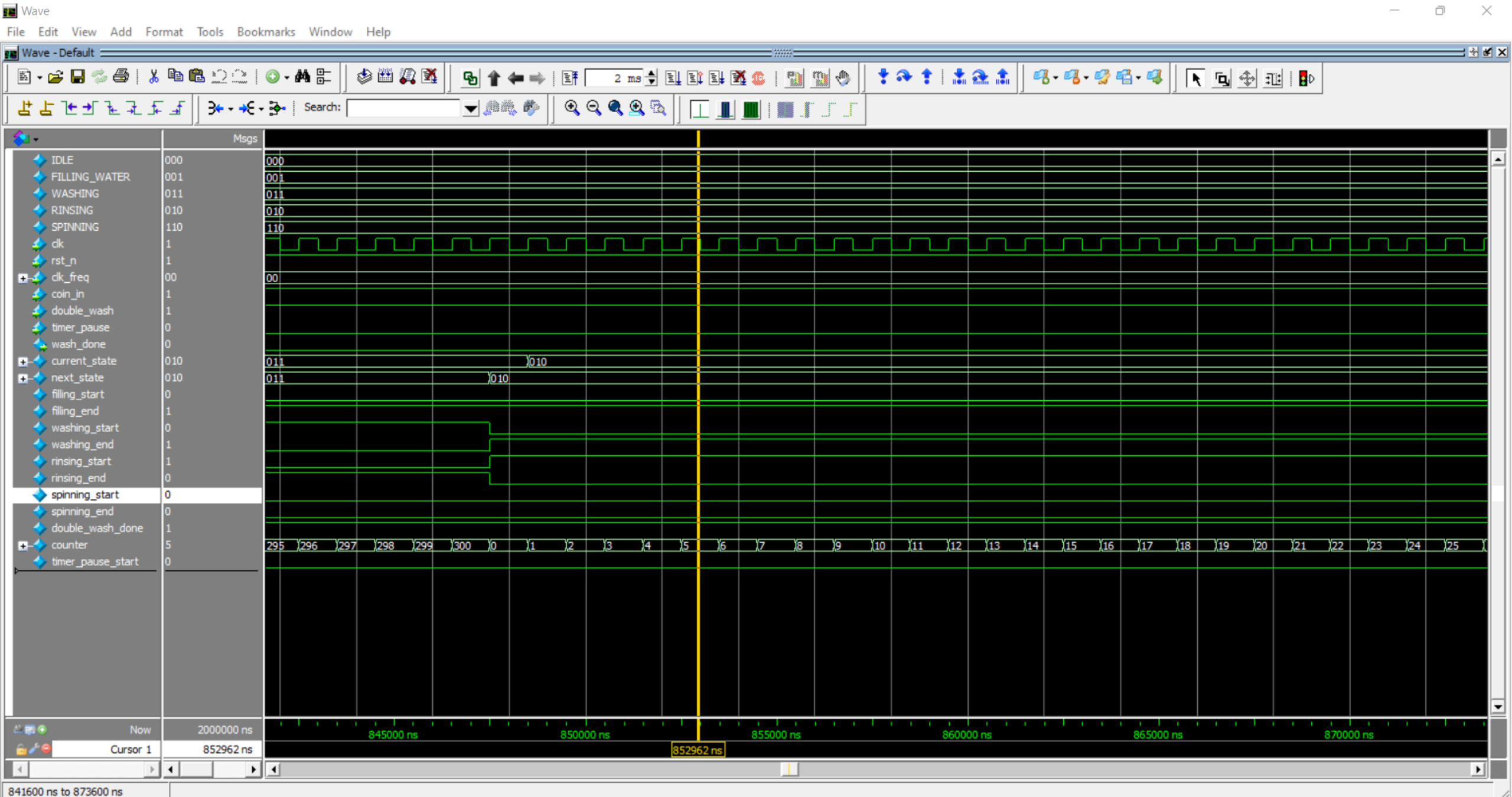
After the first washing state is done. (5 Minutes = 300 clock cycles after scaling down)



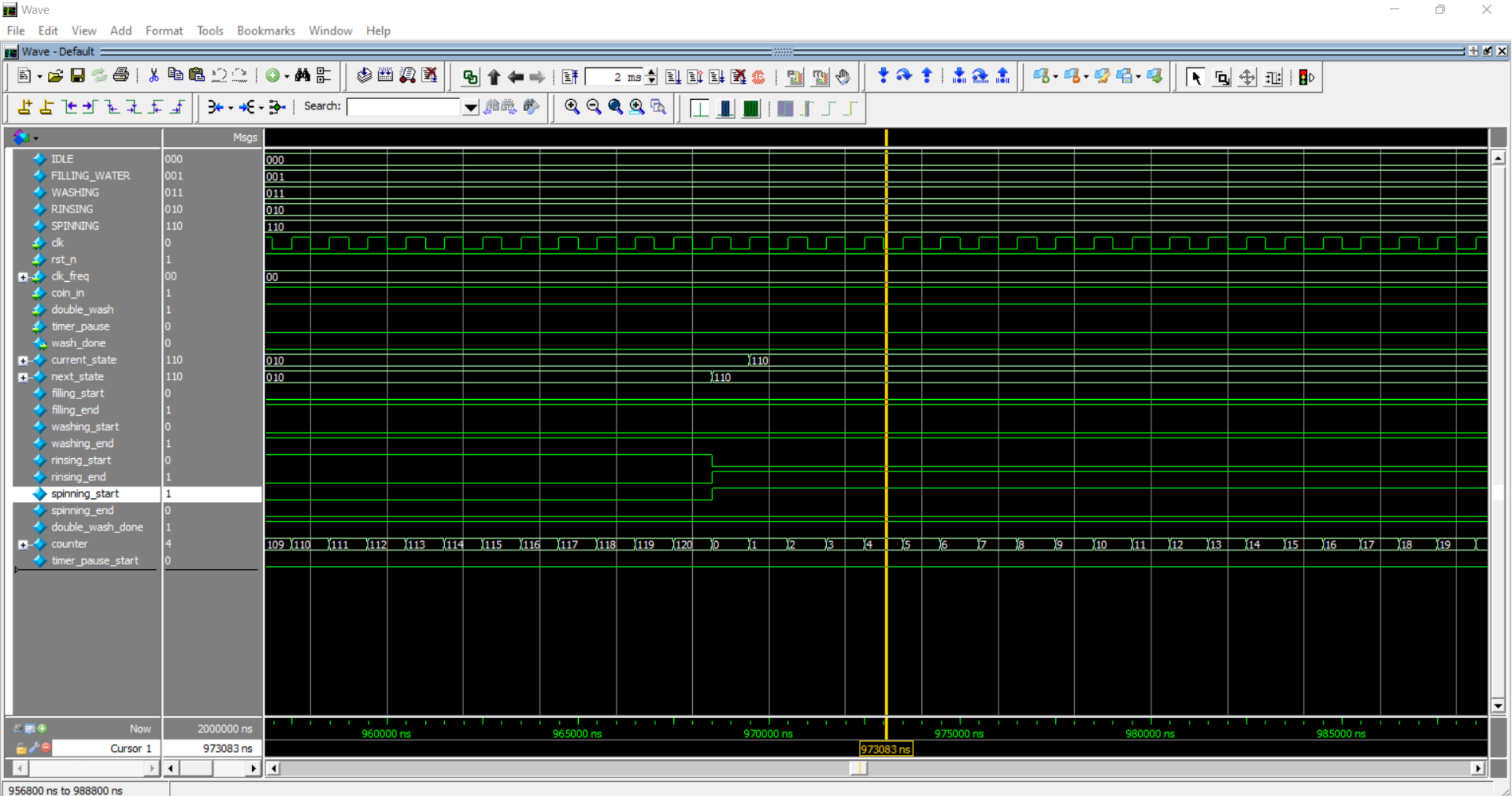
After the first rinsing state is done. (2 Minutes = 120 clock cycles after scaling down)



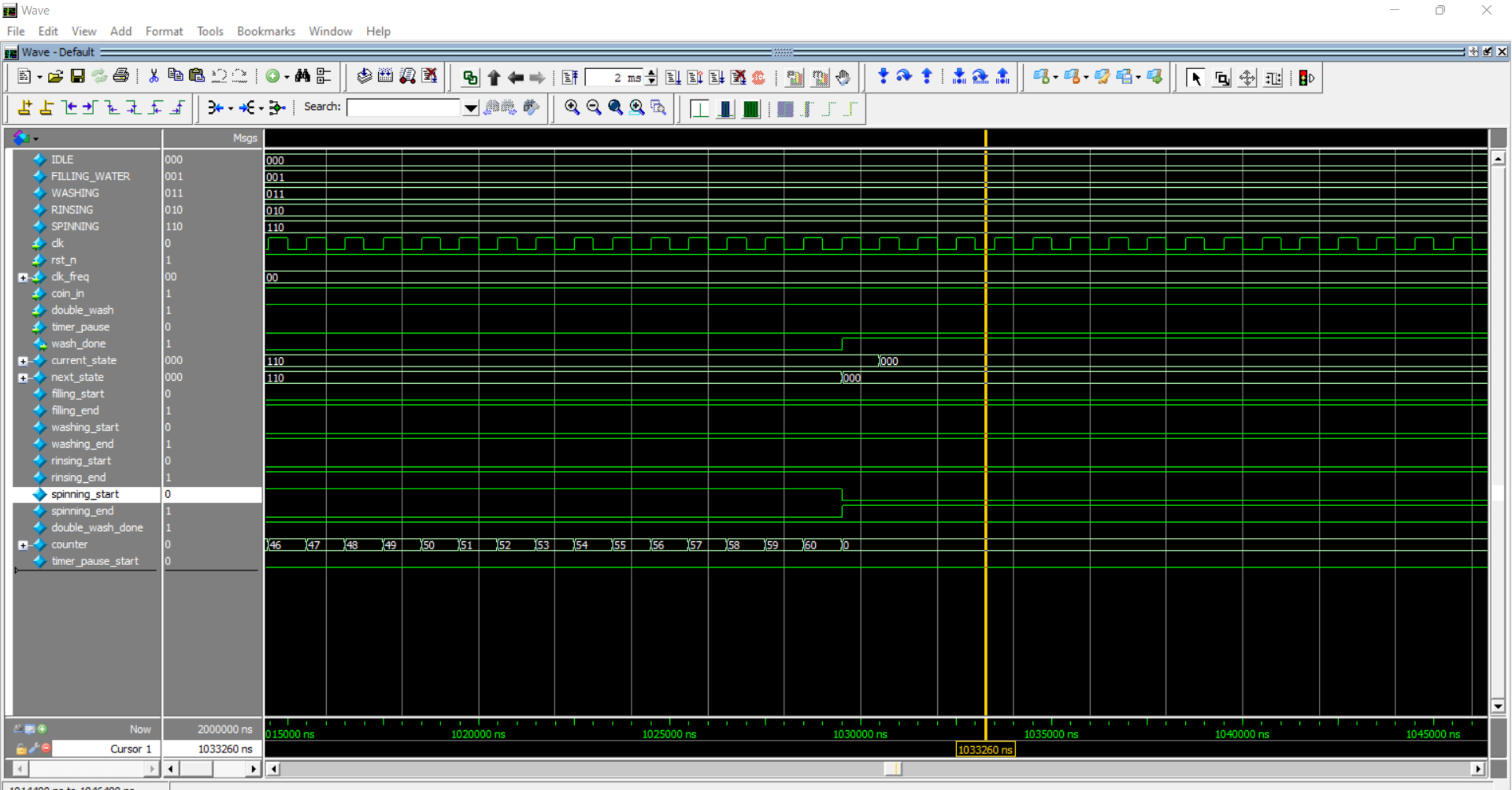
After the second washing state is done. (5 Minutes = 300 clock cycles after scaling down)



After the second rinsing state is done. (2 Minutes = 120 clock cycles after scaling down)



After the spinning state is done. (1 Minute = 60 clock cycles after scaling down)

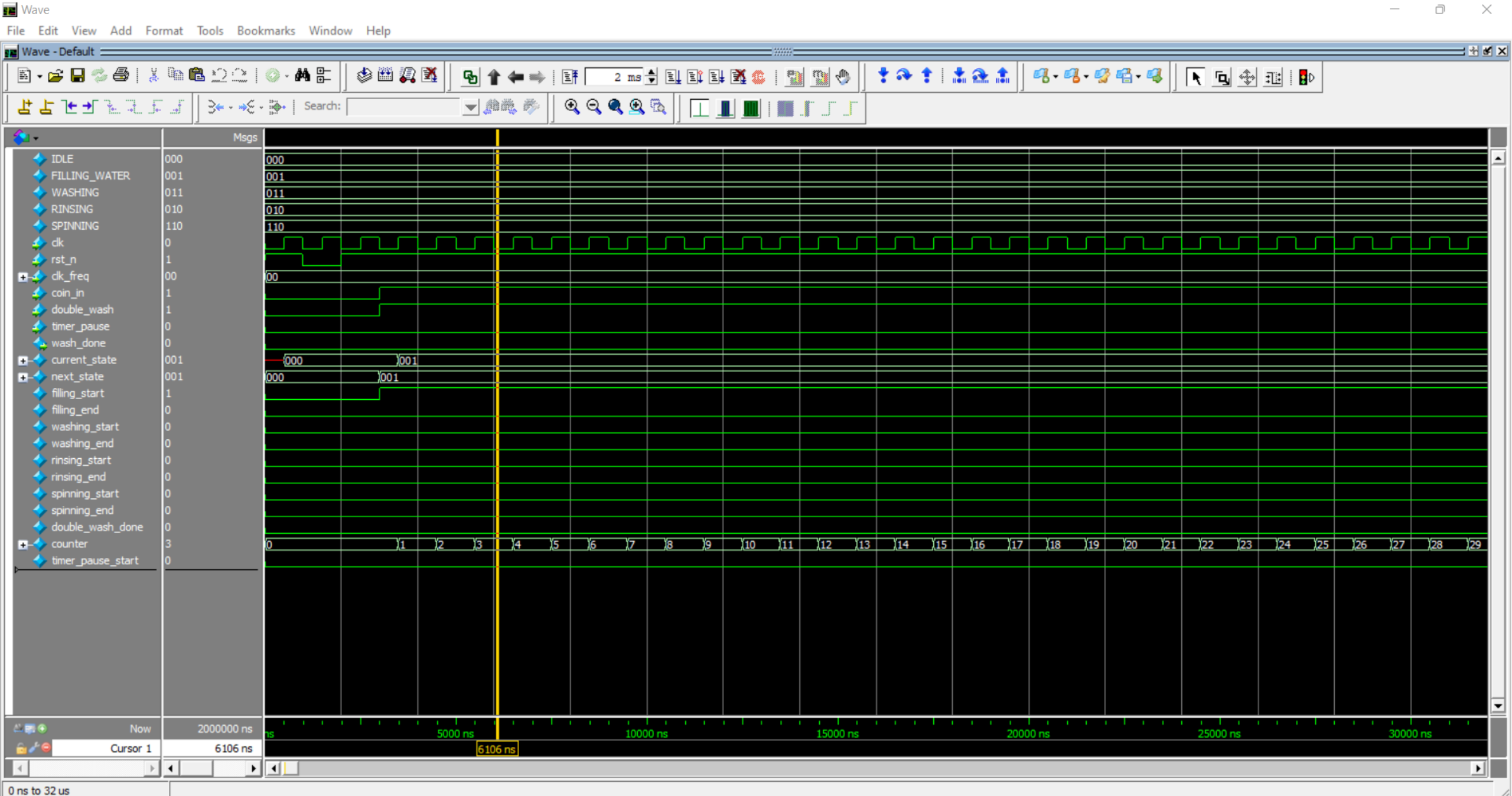


Initial Block (Test Case 3)

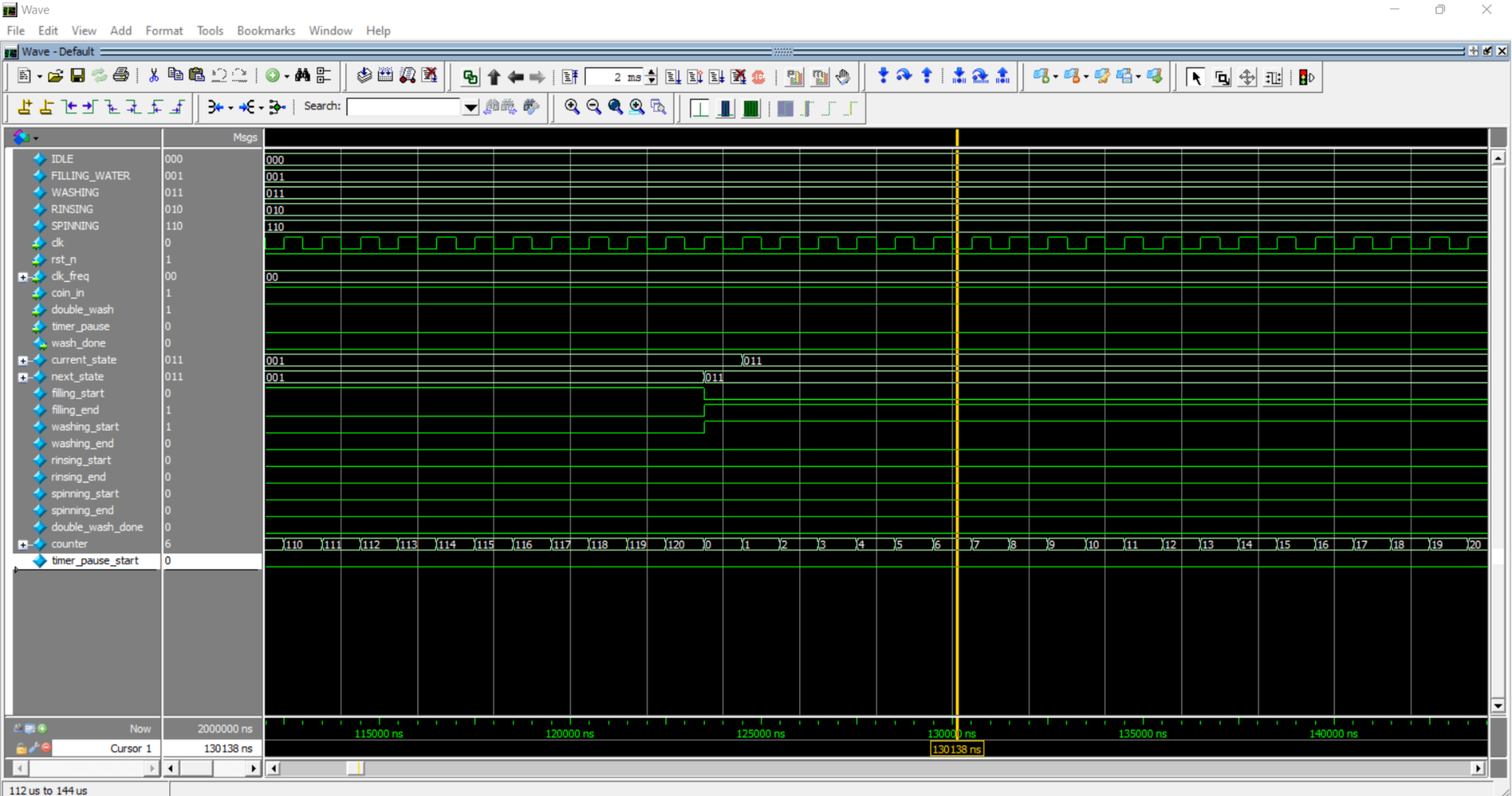
Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed. Next are the simulation results from ModelSim software. (1 MHz)

```
29  initial begin
30
31   // Save Waveform
32     $dumpfile("washing_machine.vcd") ;
33     $dumpvars;
34
35   // initialization
36     initialize() ;
37
38   // Reset
39     reset() ;
40
41   // double_wash
42     double_wash() ;
43
44     coin_in_tb     = 1'b1  ;      // coin is deposited
45
46  #975
47   // Timer_Pause
48     Timer_Pause() ;
49
50  #3000
51  $finish ;
52  end
```

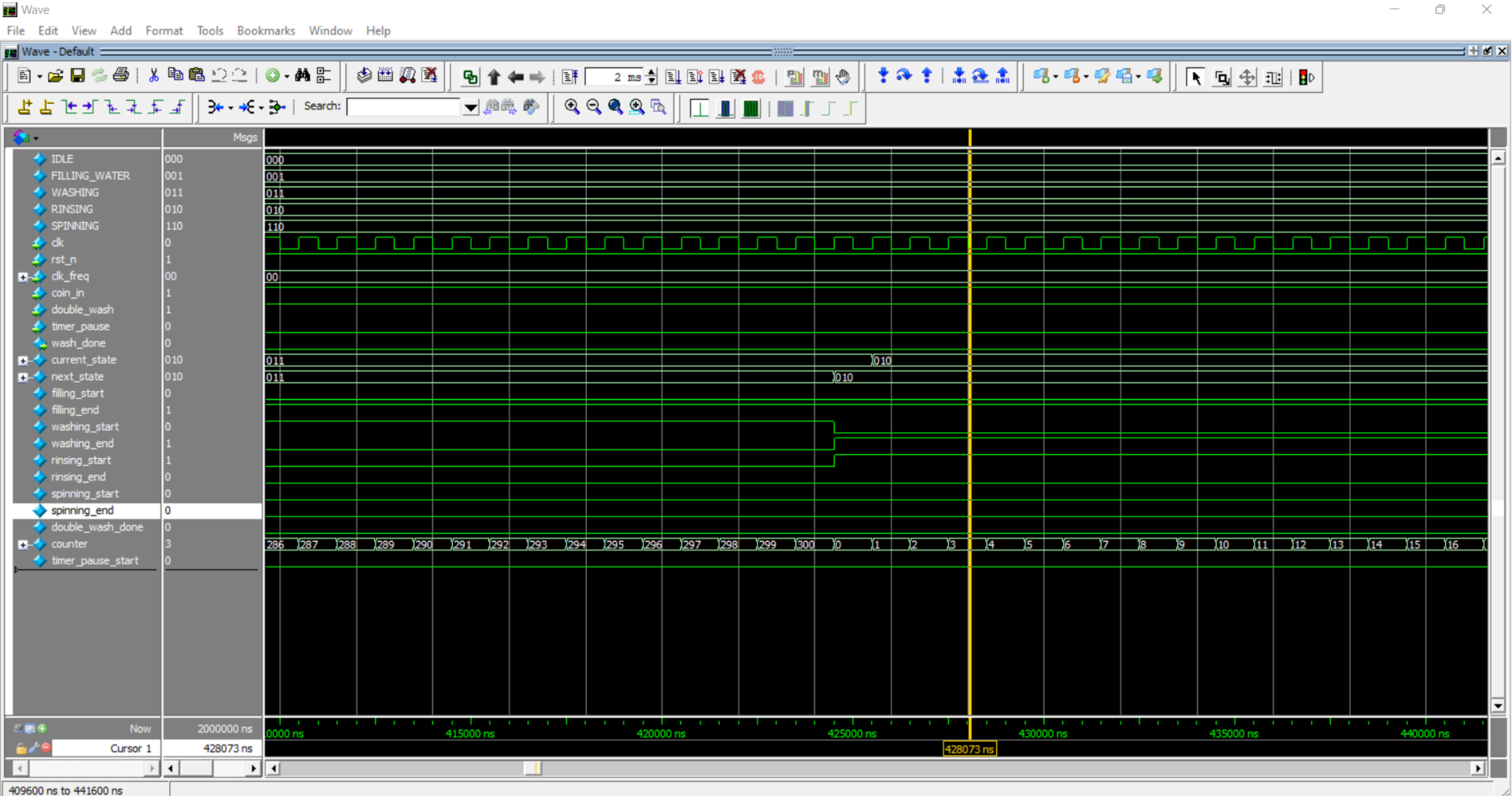
At the beginning.



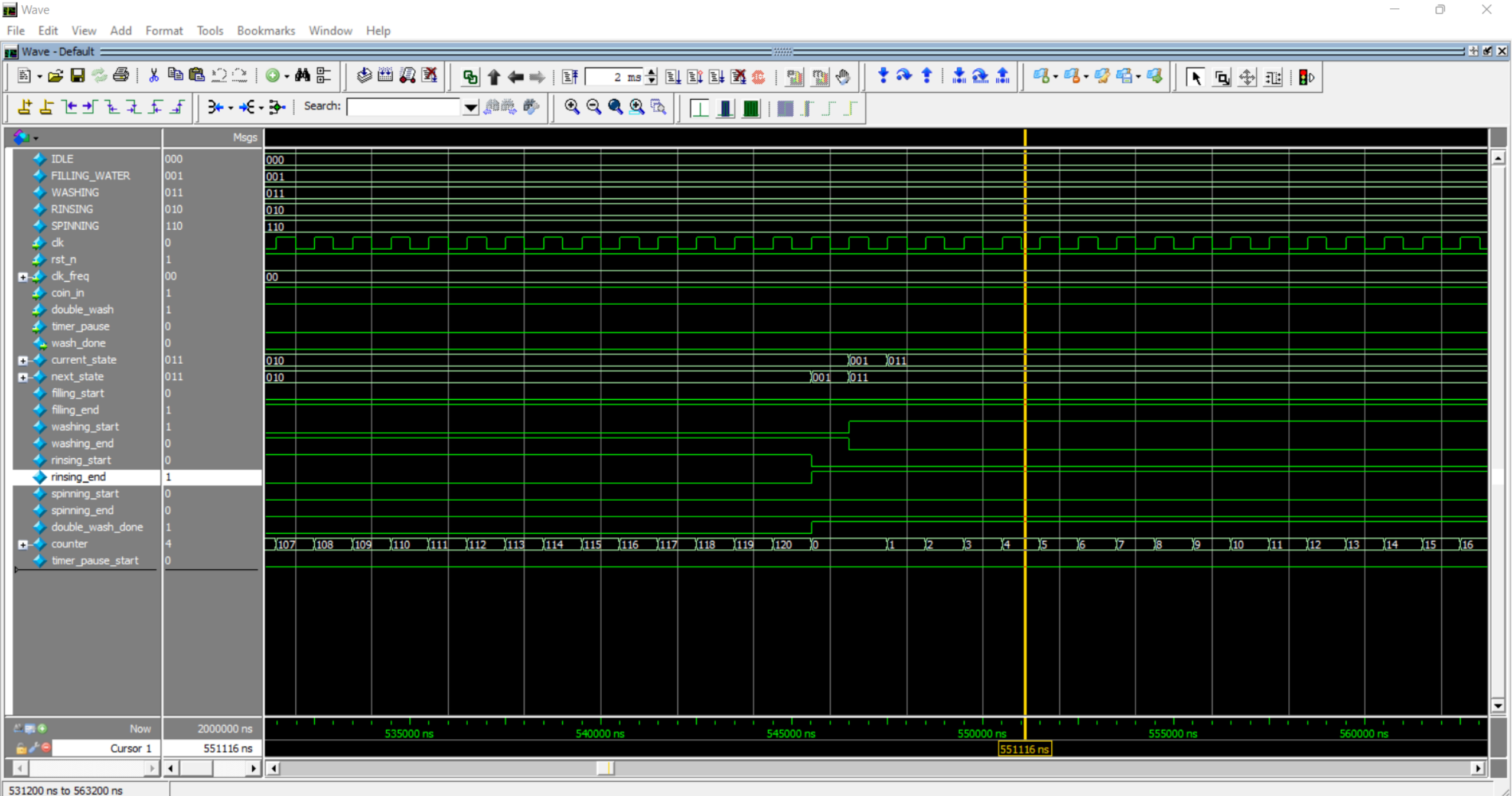
After the filling state is done. (2 Minutes = 120 clock cycles after scaling down)



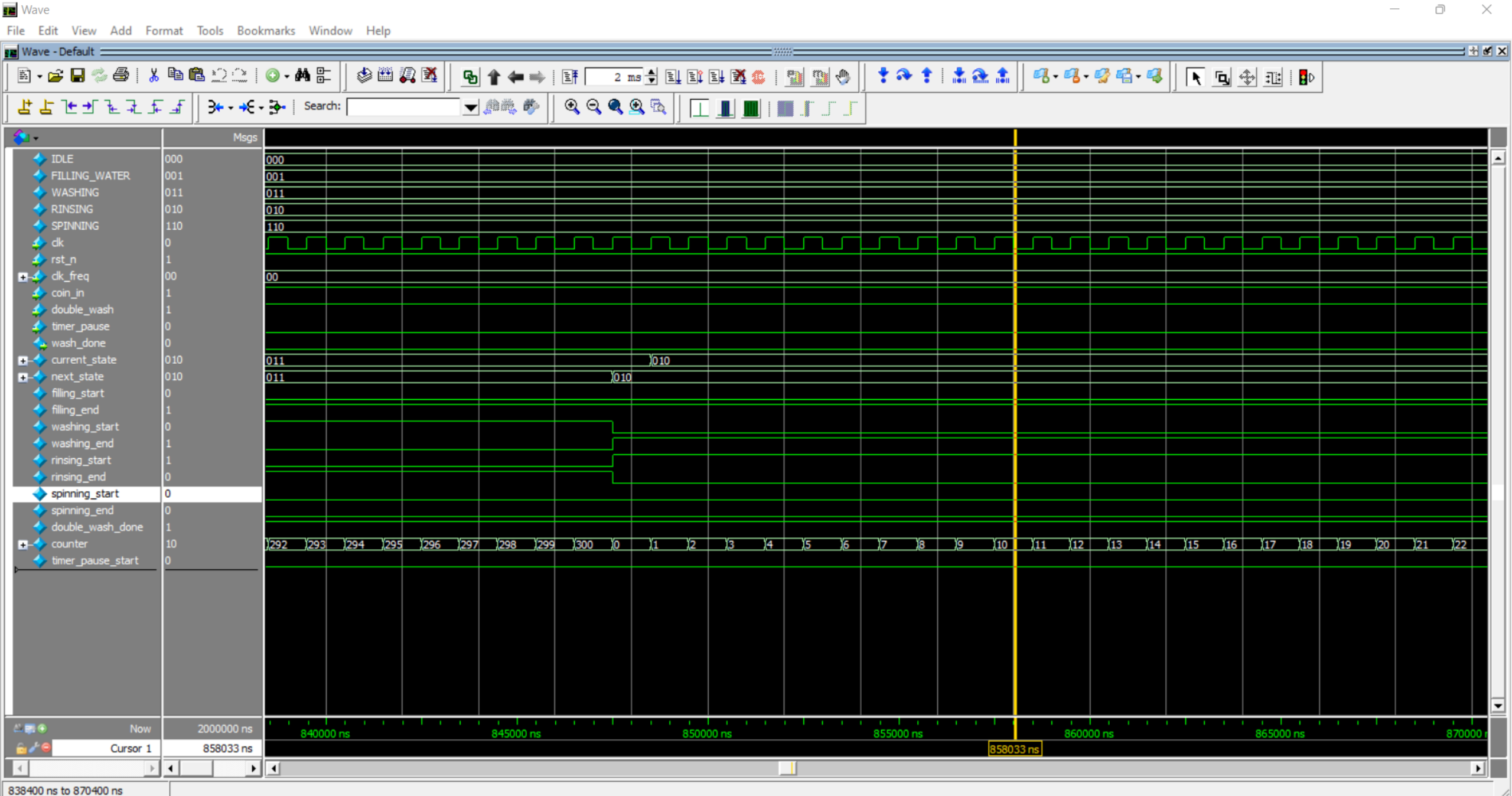
After the first washing state is done. (5 Minutes = 300 clock cycles after scaling down)



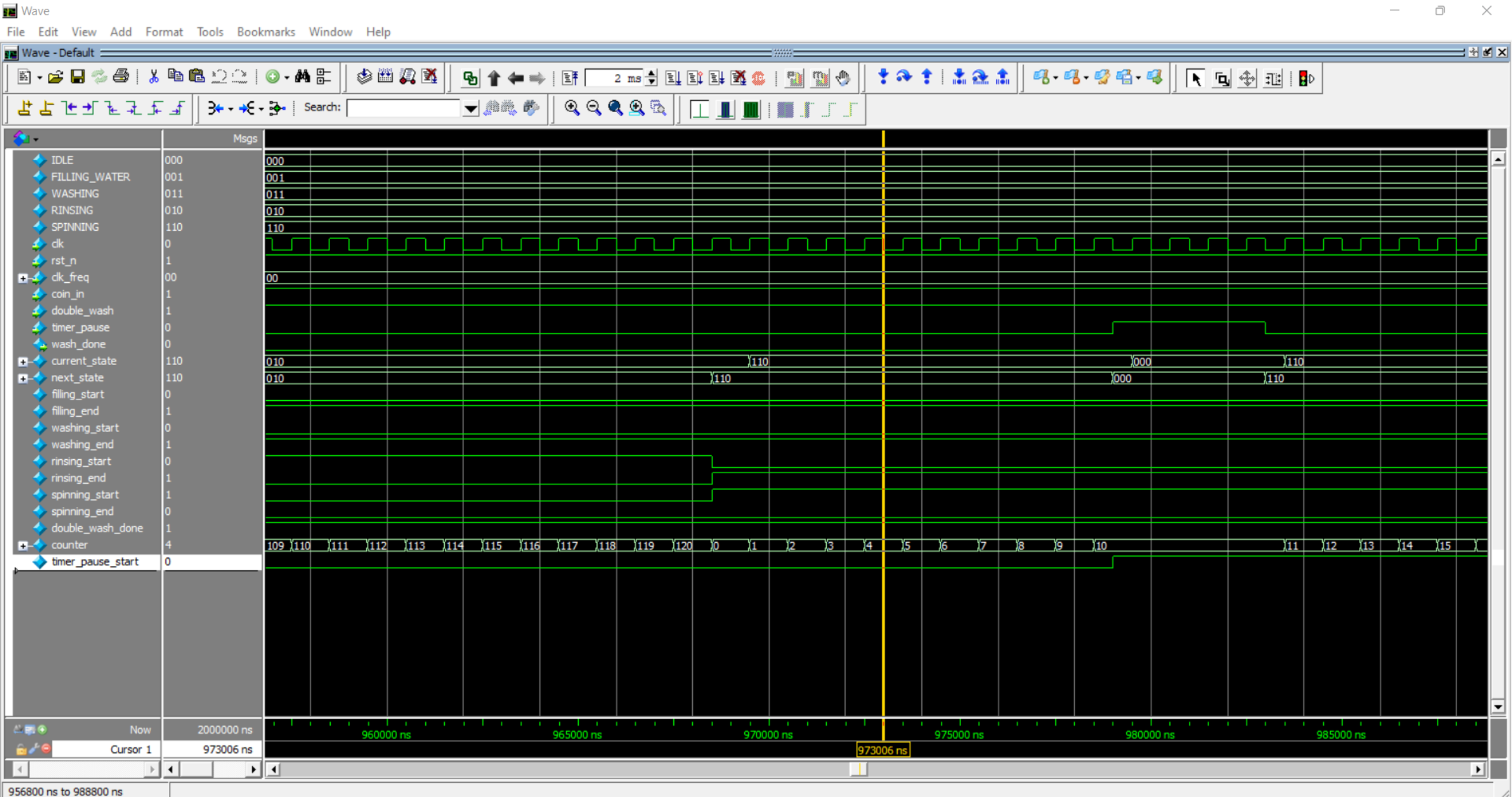
After the first rinsing state is done. (2 Minutes = 120 clock cycles after scaling down)



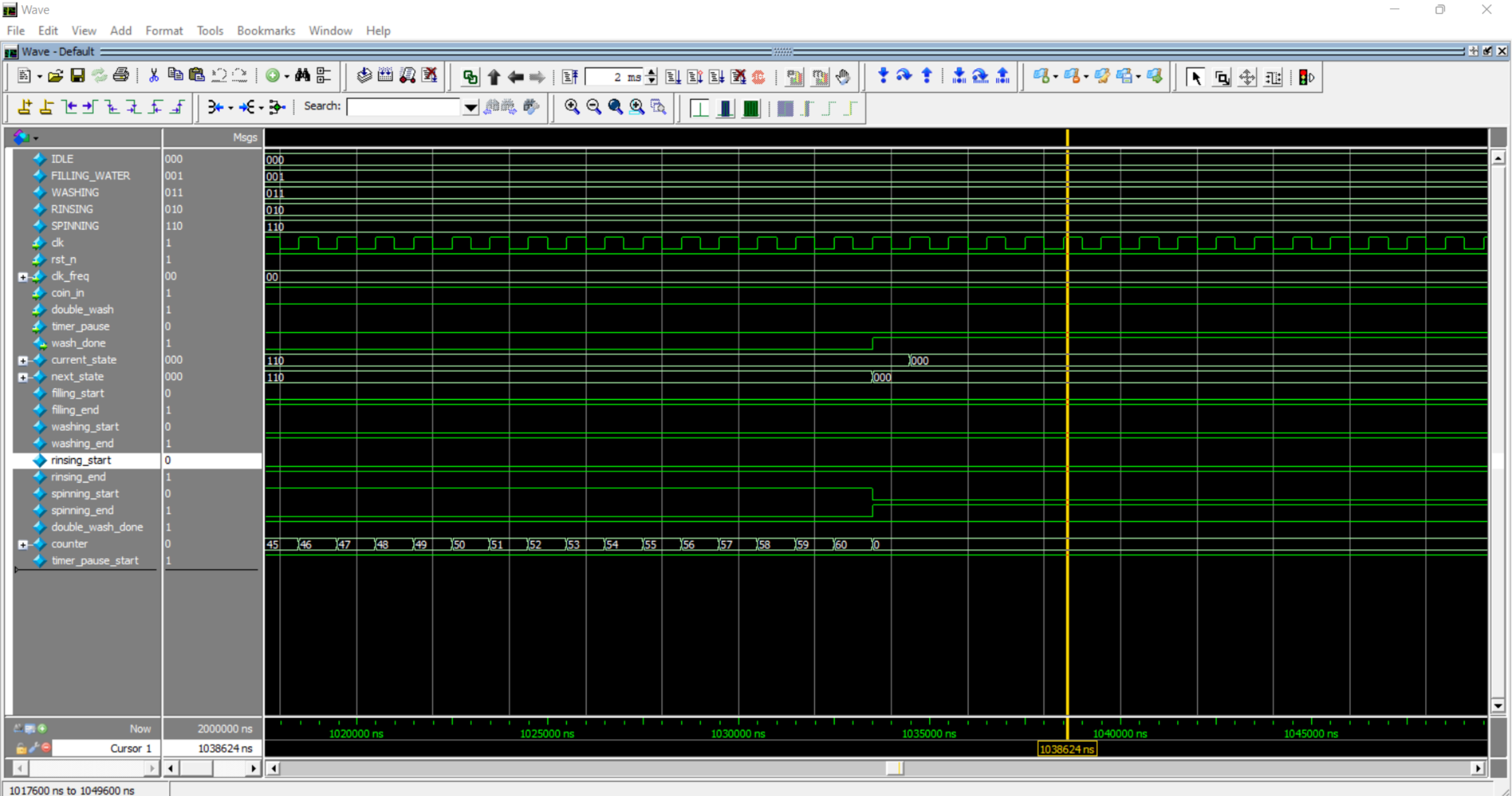
After the second washing state is done. (5 Minutes = 300 clock cycles after scaling down)



After the second rinsing state is done. (2 Minutes = 120 clock cycles after scaling down), and the timer pause is pressed during spinning.



After the spinning state is done. (1 Minute = 60 clock cycles after scaling down)



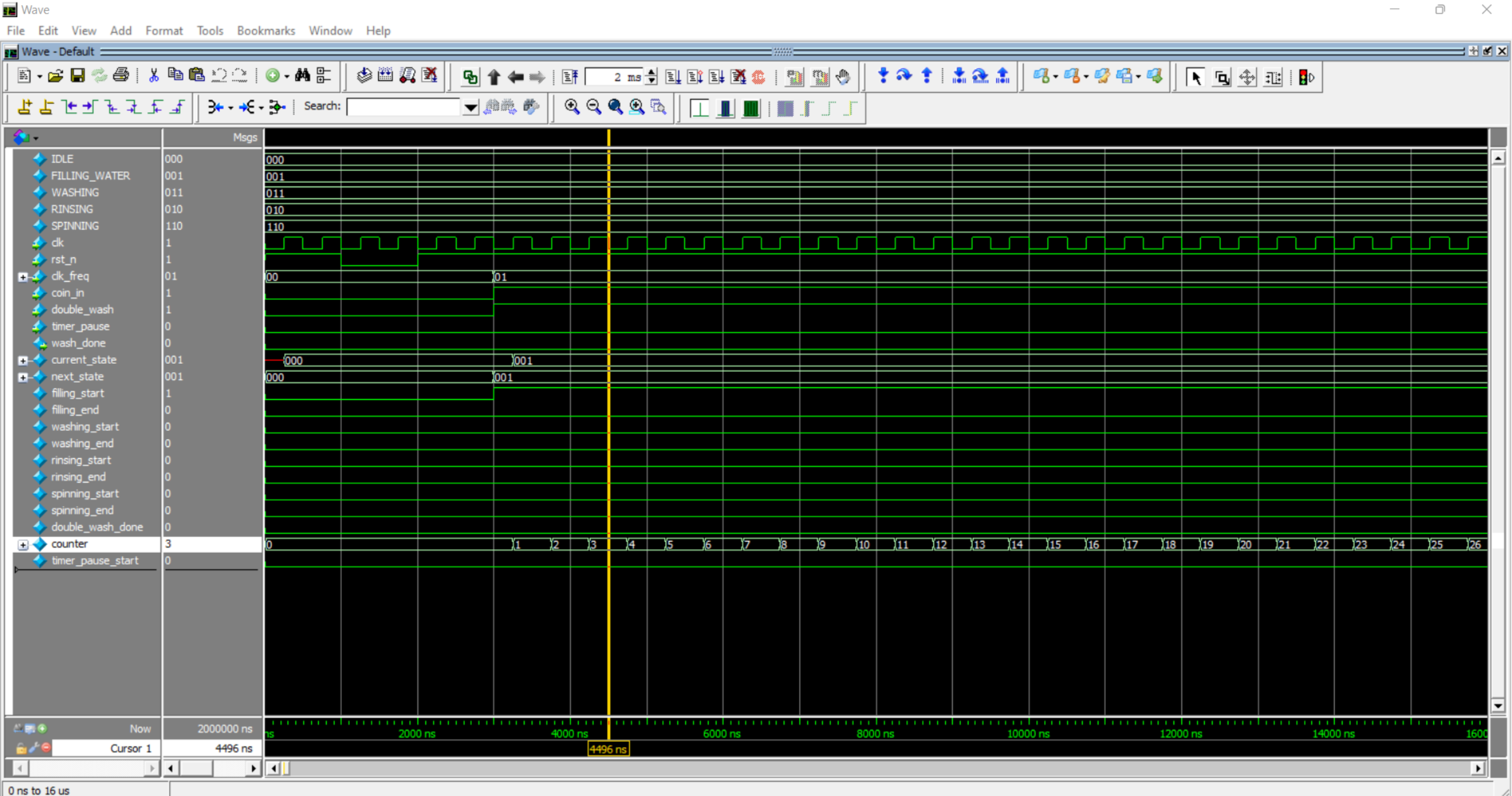
Initial Block (Test Case 4)

Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed. Next are the simulation results from ModelSim software. (2 MHz)

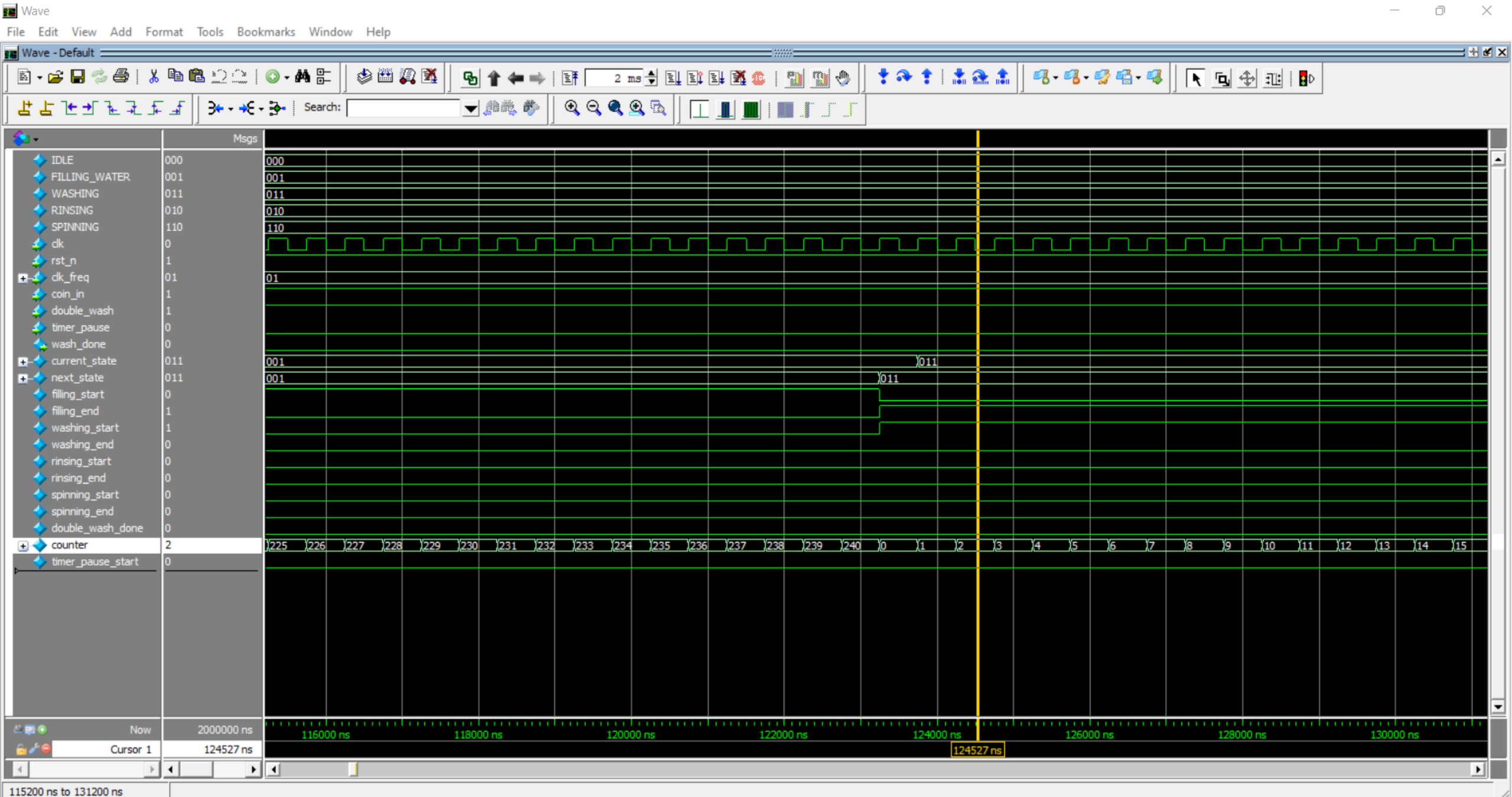
```
29  initial begin
30
31   // Save Waveform
32     $dumpfile("washing_machine.vcd") ;
33     $dumpvars;
34
35   // initialization
36     initialize() ;
37
38   // Reset
39     reset() ;
40
41   // double_wash
42     double_wash() ;
43     clk_freq_tb     = 2'b01 ;      // frequency is 2 MHz
44     coin_in_tb      = 1'b1  ;      // coin is deposited
45
46  #975
47   // Timer_Pause
48     Timer_Pause() ;
49
50  #3000
51  $finish ;
52  end
```

```
 98  ////////////////////////////////////////////////////
 99  ////////////////// Clock Generator  ////////////////
100  ////////////////////////////////////////////////////
101
102  //always #0.5    clk_tb = ~clk_tb ;   // period = 1 us       (1 MHz)
103    always #0.25   clk_tb = ~clk_tb ;   // period = 0.5 us     (2 MHz)
104  //always #0.125  clk_tb = ~clk_tb ;   // period = 0.25 us    (4 MHz)
105  //always #0.0625 clk_tb = ~clk_tb ;   // period = 0.125 us   (8 MHz)
106
```

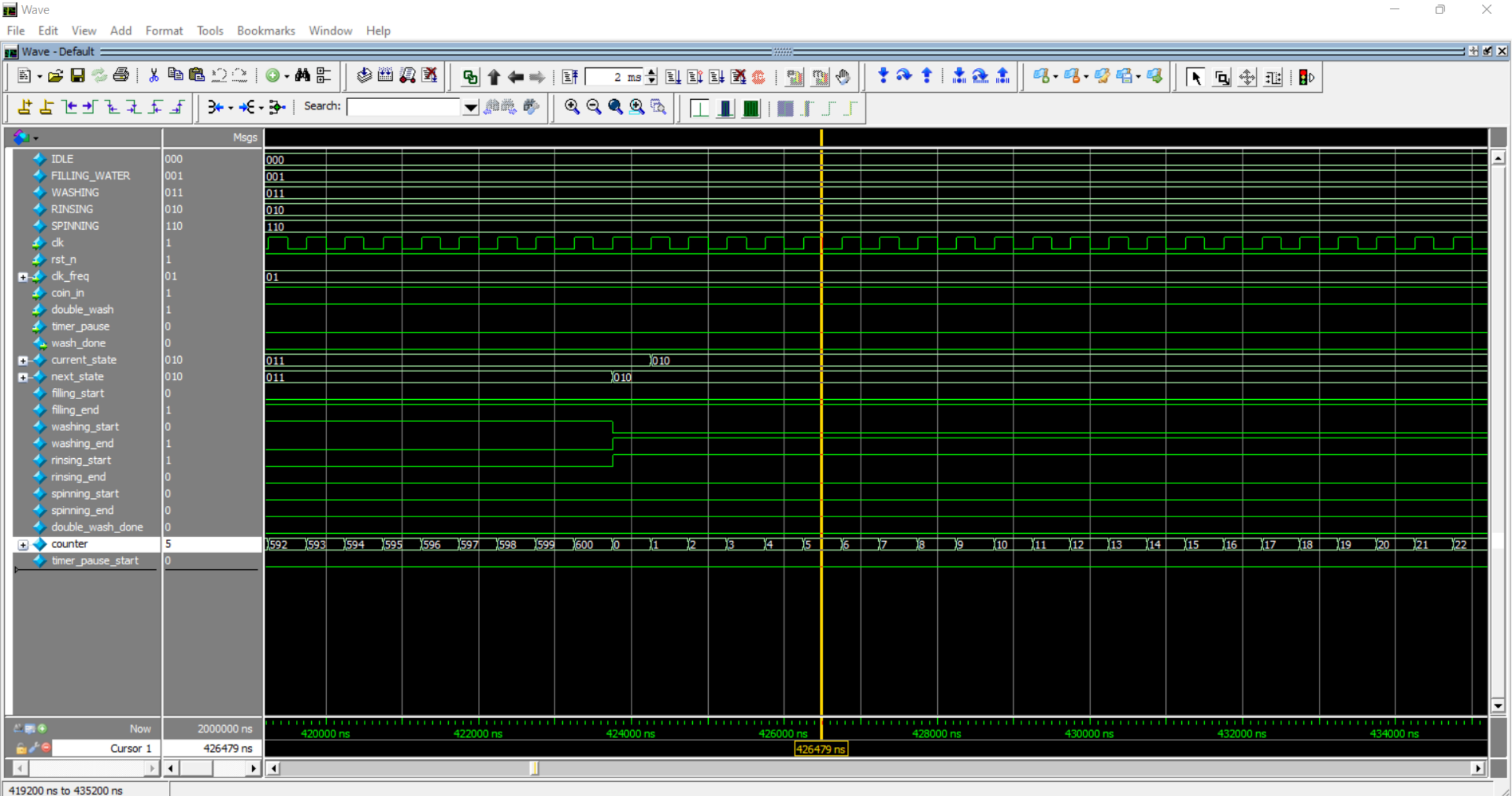
At the beginning.



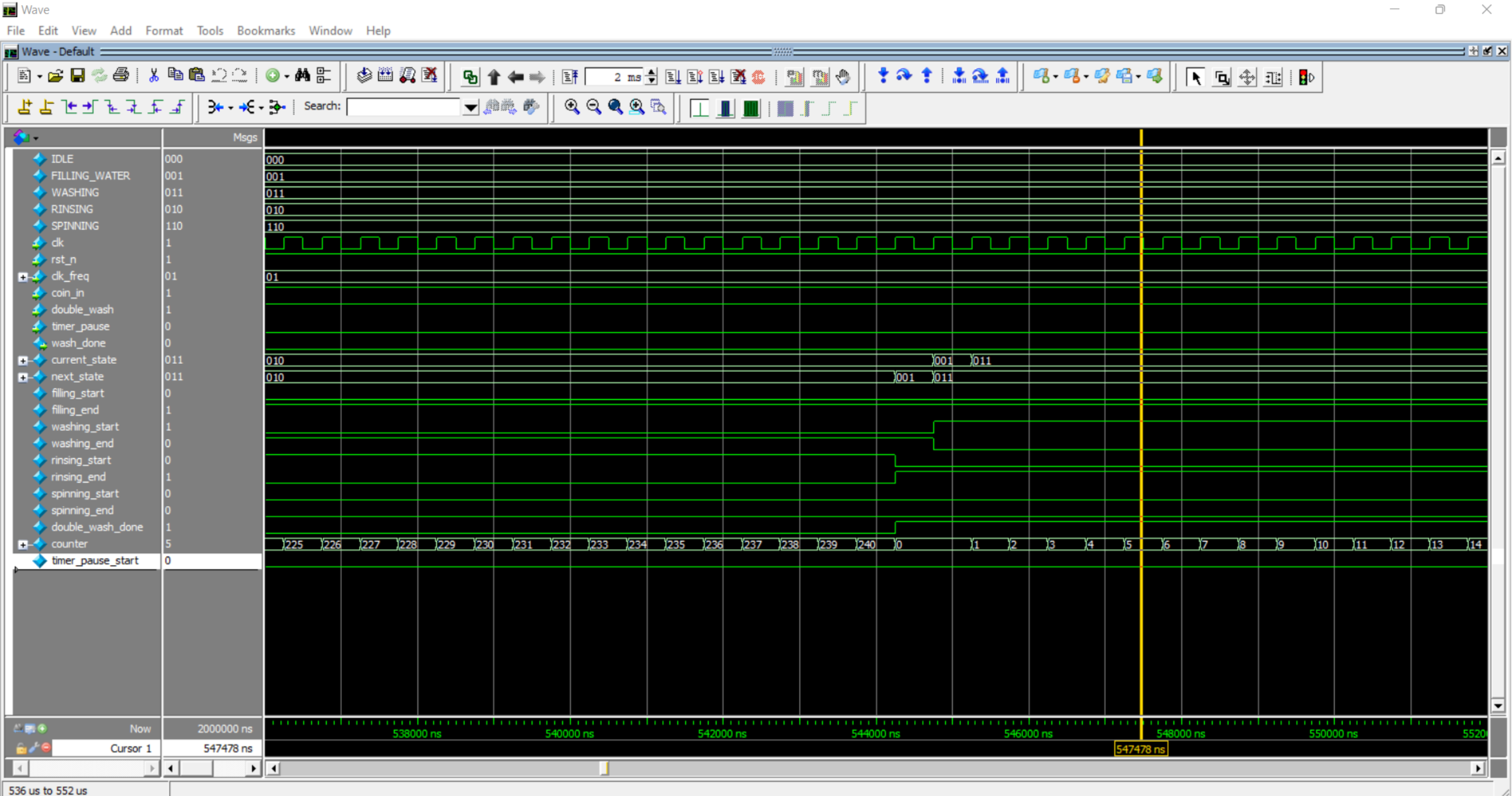
After the filling state is done. (2 Minutes = 240 clock cycles after scaling down)



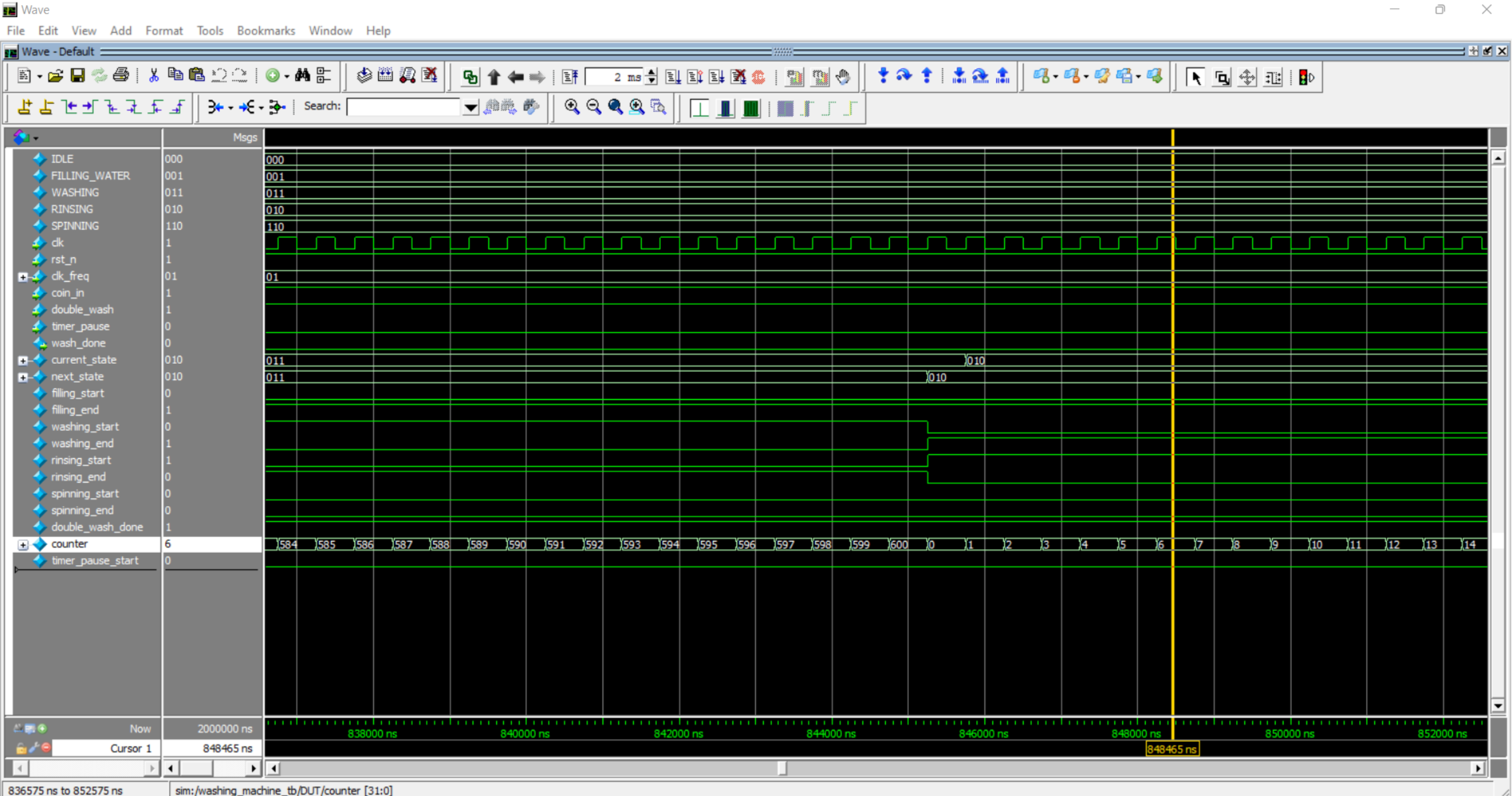
After the first washing state is done. (5 Minutes = 600 clock cycles after scaling down)



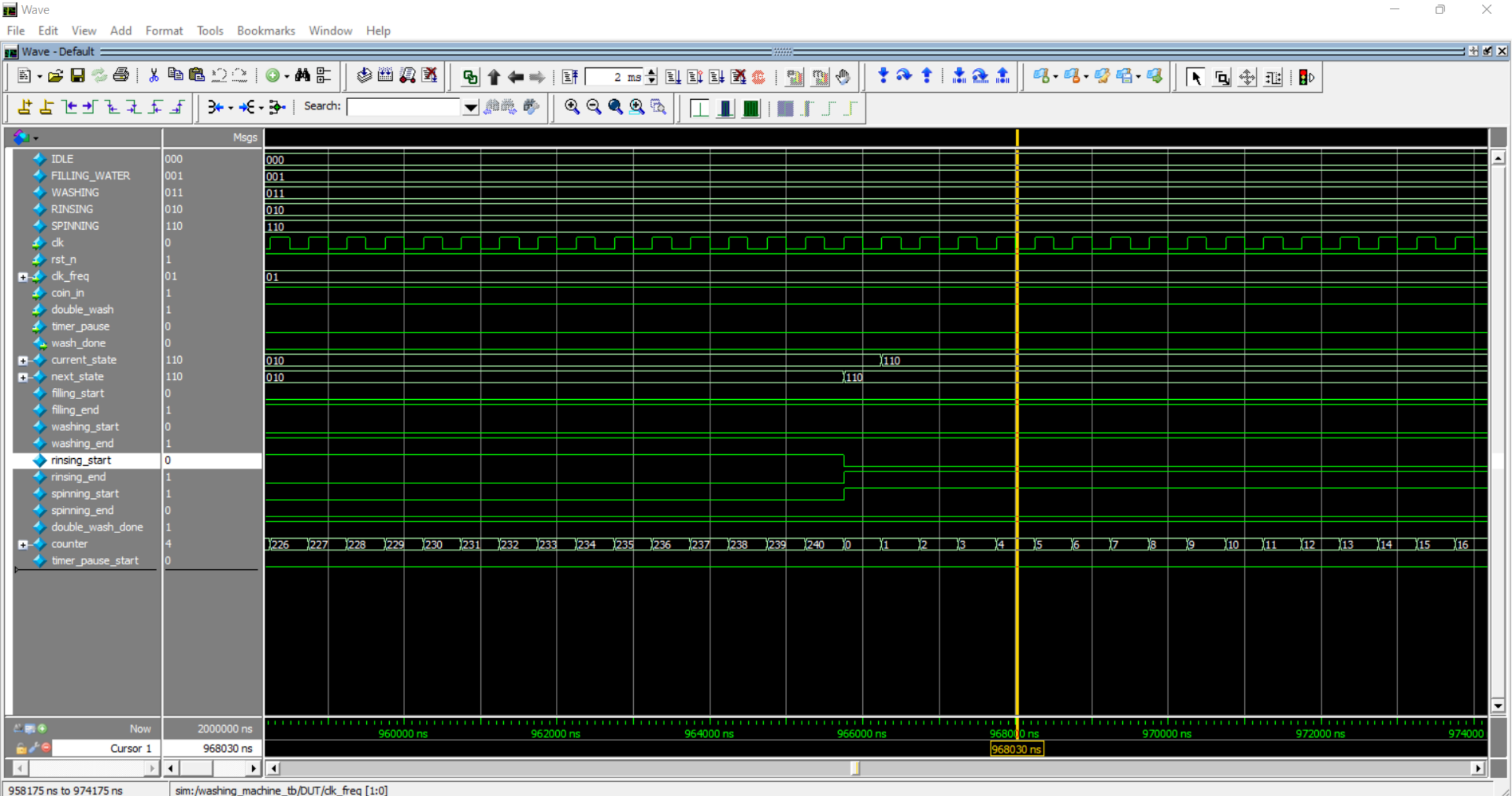
After the first rinsing state is done. (2 Minutes = 240 clock cycles after scaling down)



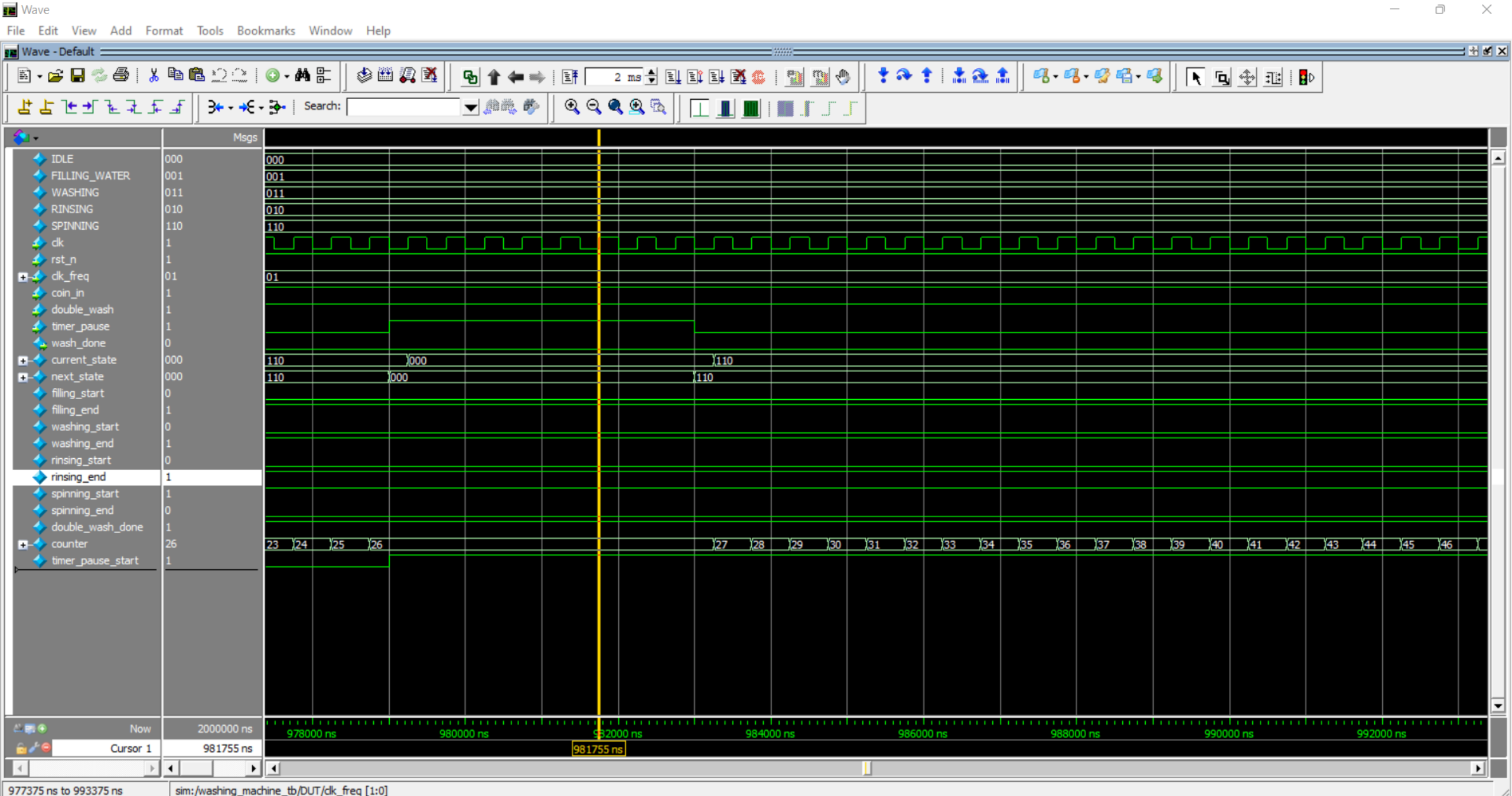
After the second washing state is done. (5 Minutes = 600 clock cycles after scaling down)



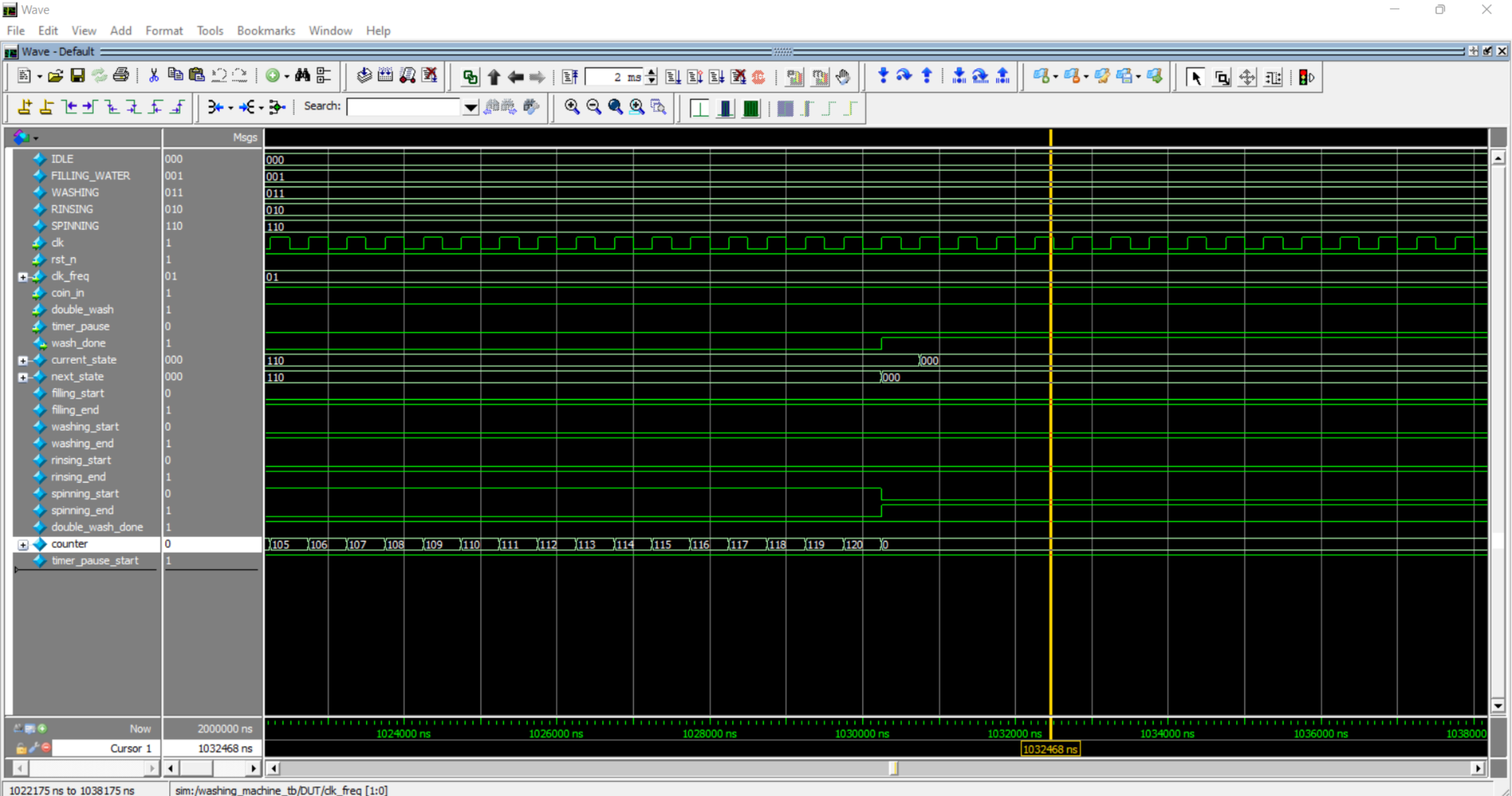
After the second rinsing state is done. (2 Minutes = 240 clock cycles after scaling down)



Timer pause button is pressed during spinning.



After the spinning state is done (1 minute = 120 clock cycles after scaling down).



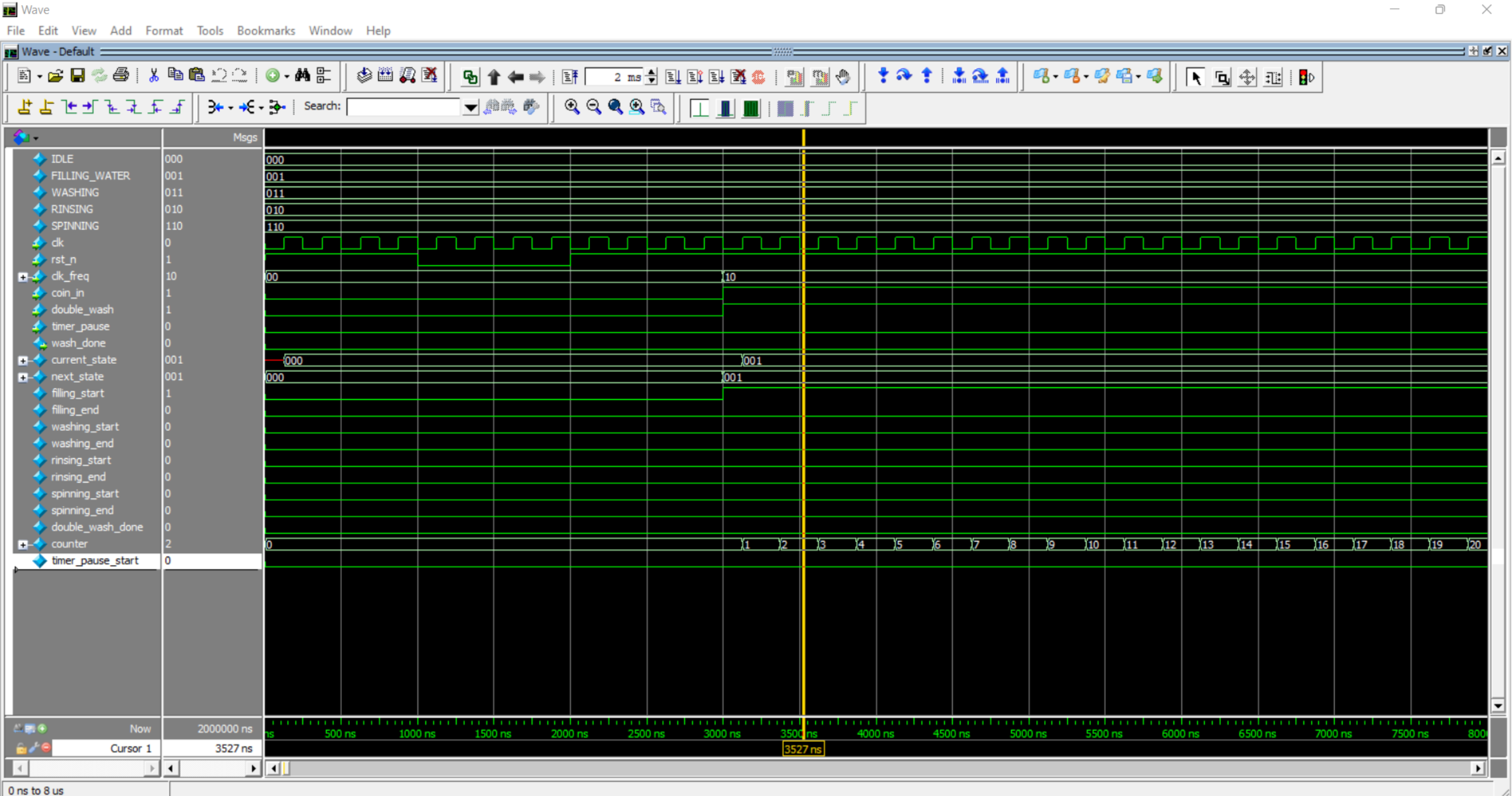
Initial Block (Test Case 5)

Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed. Next are the simulation results from ModelSim software. (4 MHz)

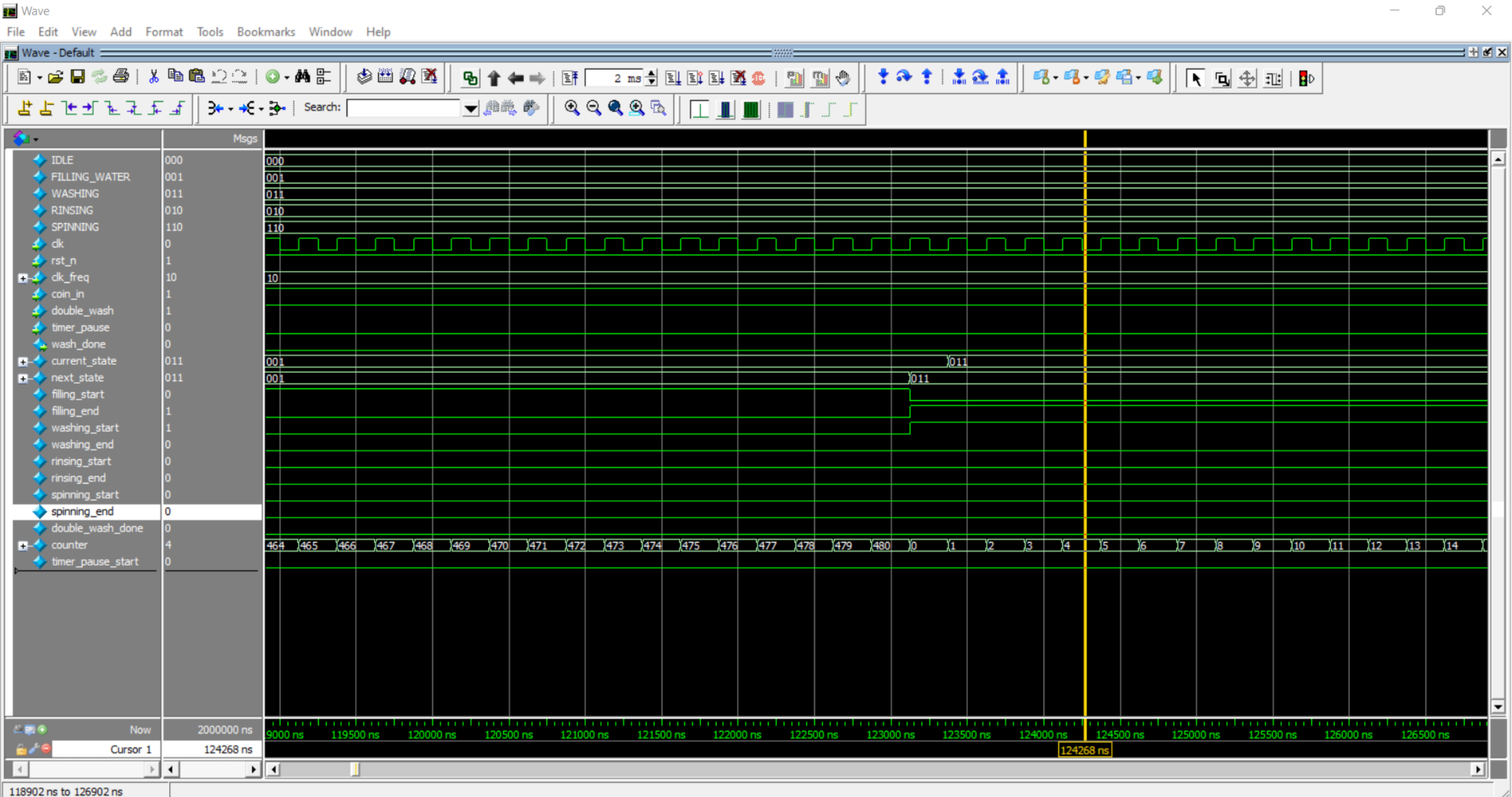
```
 98 ////////////////////////////////////////////////////////
 99 ////////////////// Clock Generator  ////////////////////
100 ////////////////////////////////////////////////////////
101
102 //always #0.5    clk_tb = ~clk_tb ;   // period = 1 us       (1 MHz)
103 //always #0.25   clk_tb = ~clk_tb ;   // period = 0.5 us     (2 MHz)
104   always #0.125  clk_tb = ~clk_tb ;   // period = 0.25 us    (4 MHz)
105 //always #0.0625 clk_tb = ~clk_tb ;   // period = 0.125 us   (8 MHz)
106
```

```
29 initial begin
30
31  // Save Waveform
32    $dumpfile("washing_machine.vcd") ;
33    $dumpvars;
34
35  // initialization
36    initialize() ;
37
38  // Reset
39    reset() ;
40
41  // double_wash
42    double_wash() ;
43    clk_freq_tb    = 2'b10 ;      // frequency is 4 MHz
44    coin_in_tb     = 1'b1  ;      // coin is deposited
45
46 #975
47  // Timer_Pause
48    Timer_Pause() ;
49
50 #3000
51 $finish ;
52 end
```

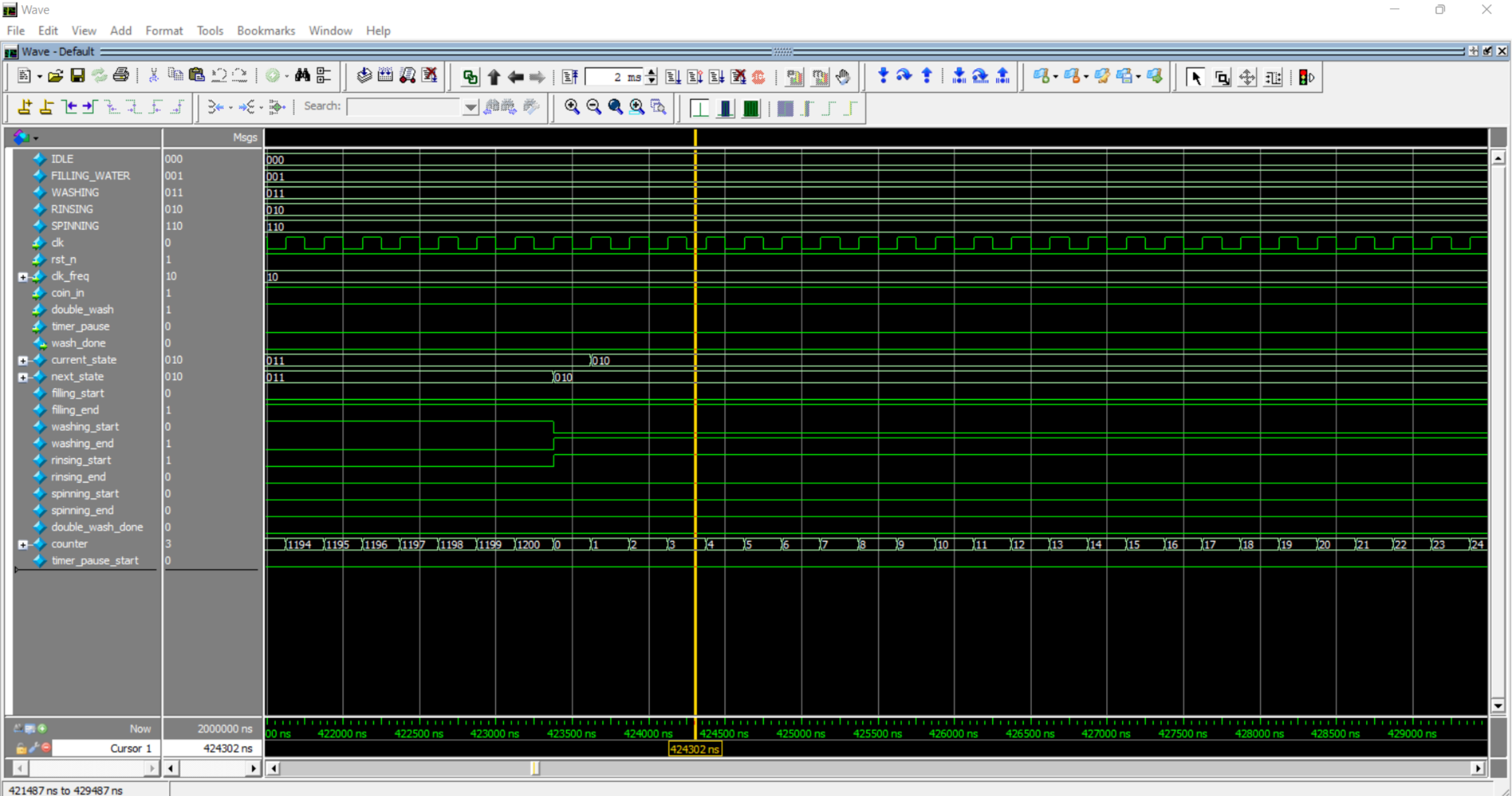
At the beginning.



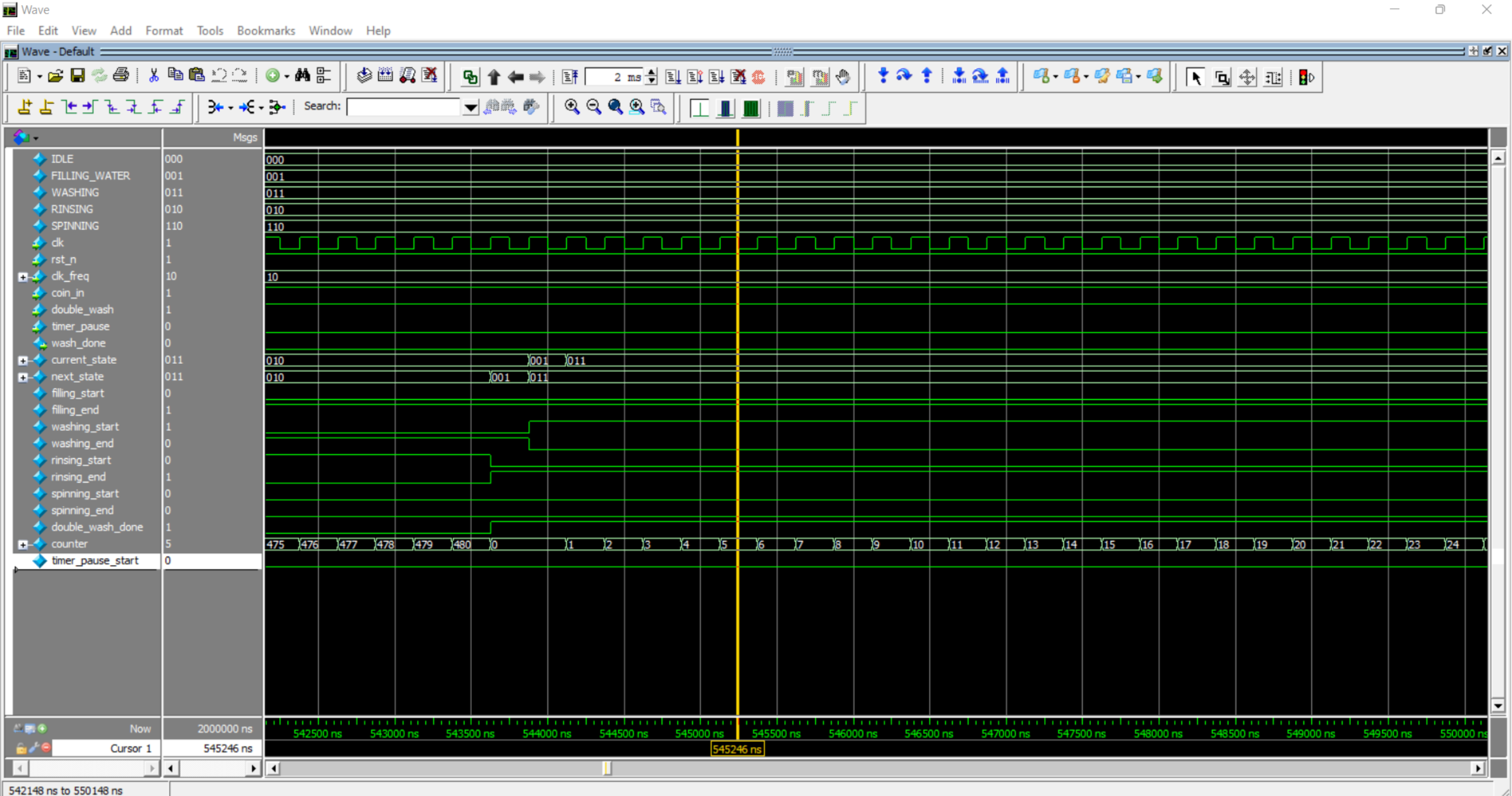
After the filling state is done. (2 Minutes = 480 clock cycles after scaling down)



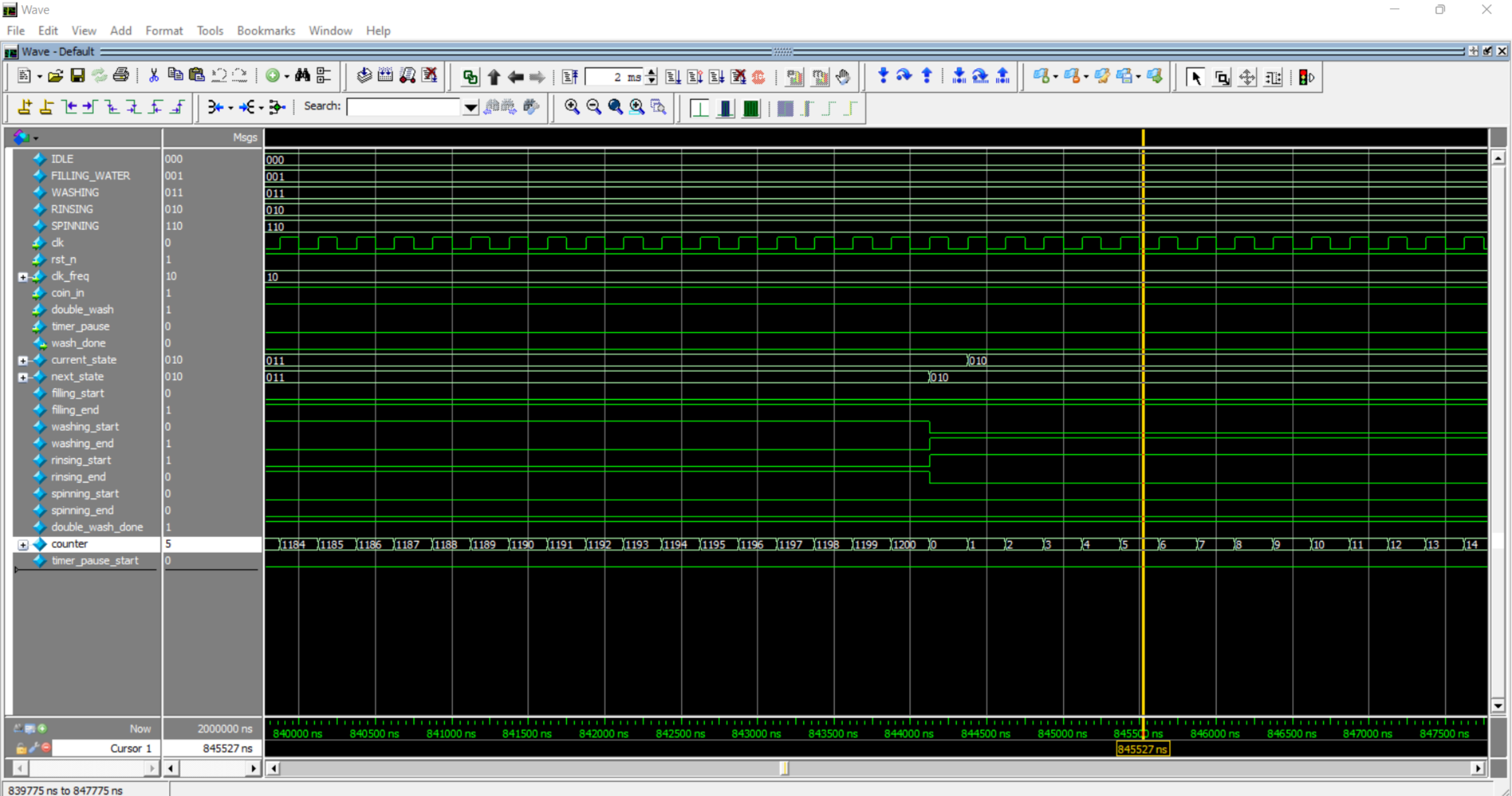
After the first washing state is done. (5 Minutes = 1200 clock cycles after scaling down)



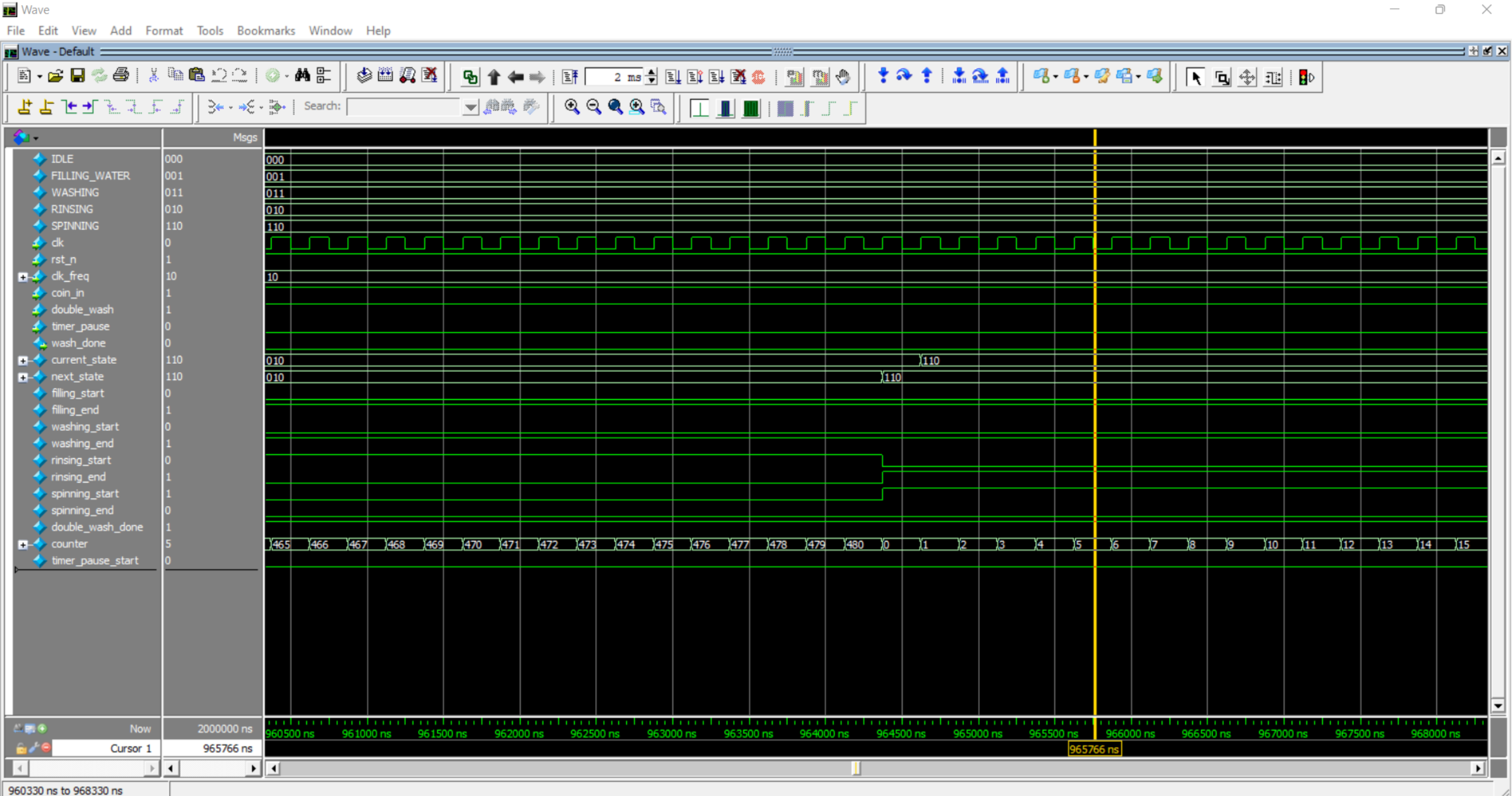
After the first rinsing state is done. (2 Minutes = 480 clock cycles after scaling down)



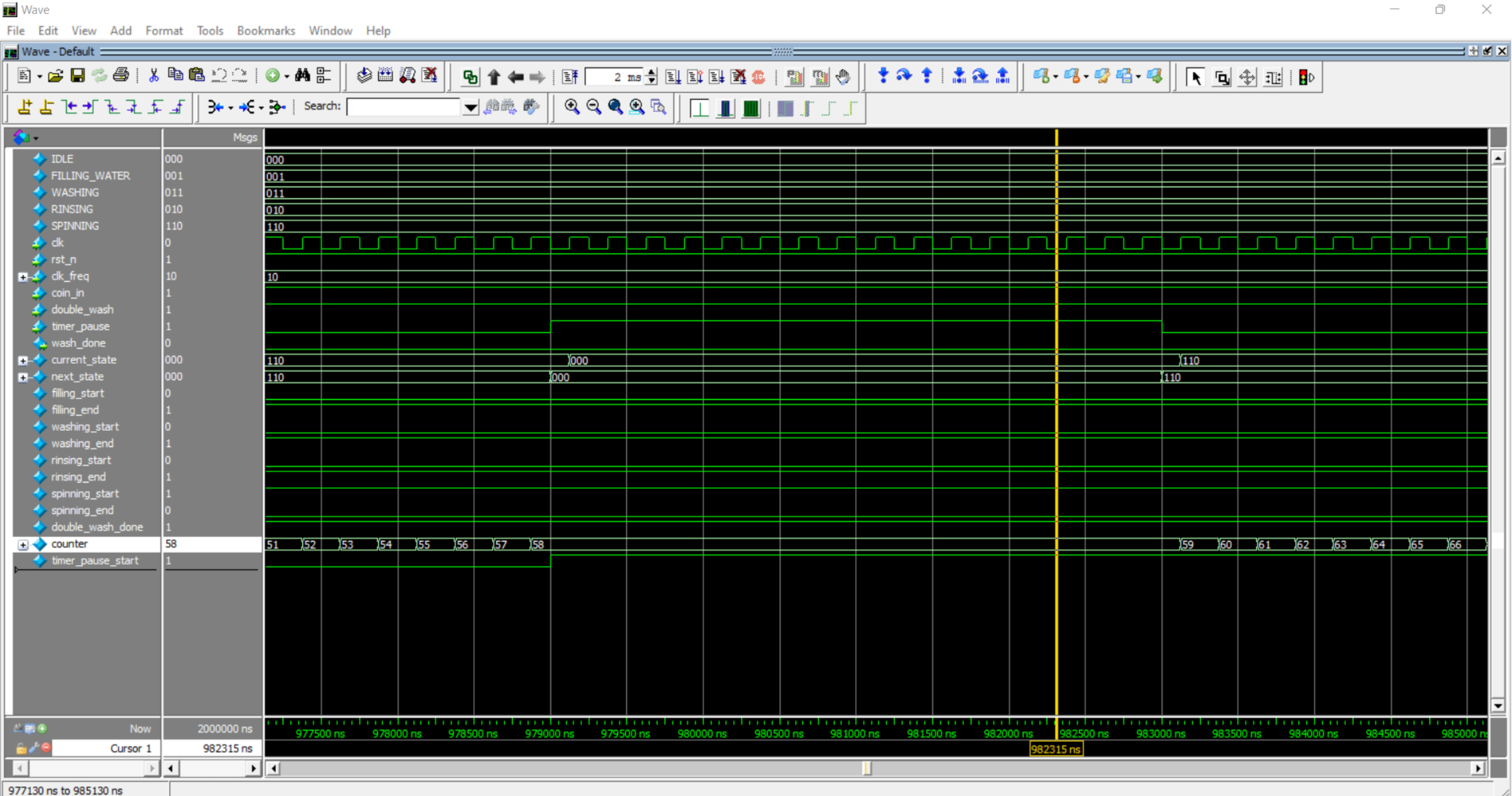
After the second washing state is done. (5 Minutes = 1200 clock cycles after scaling down)



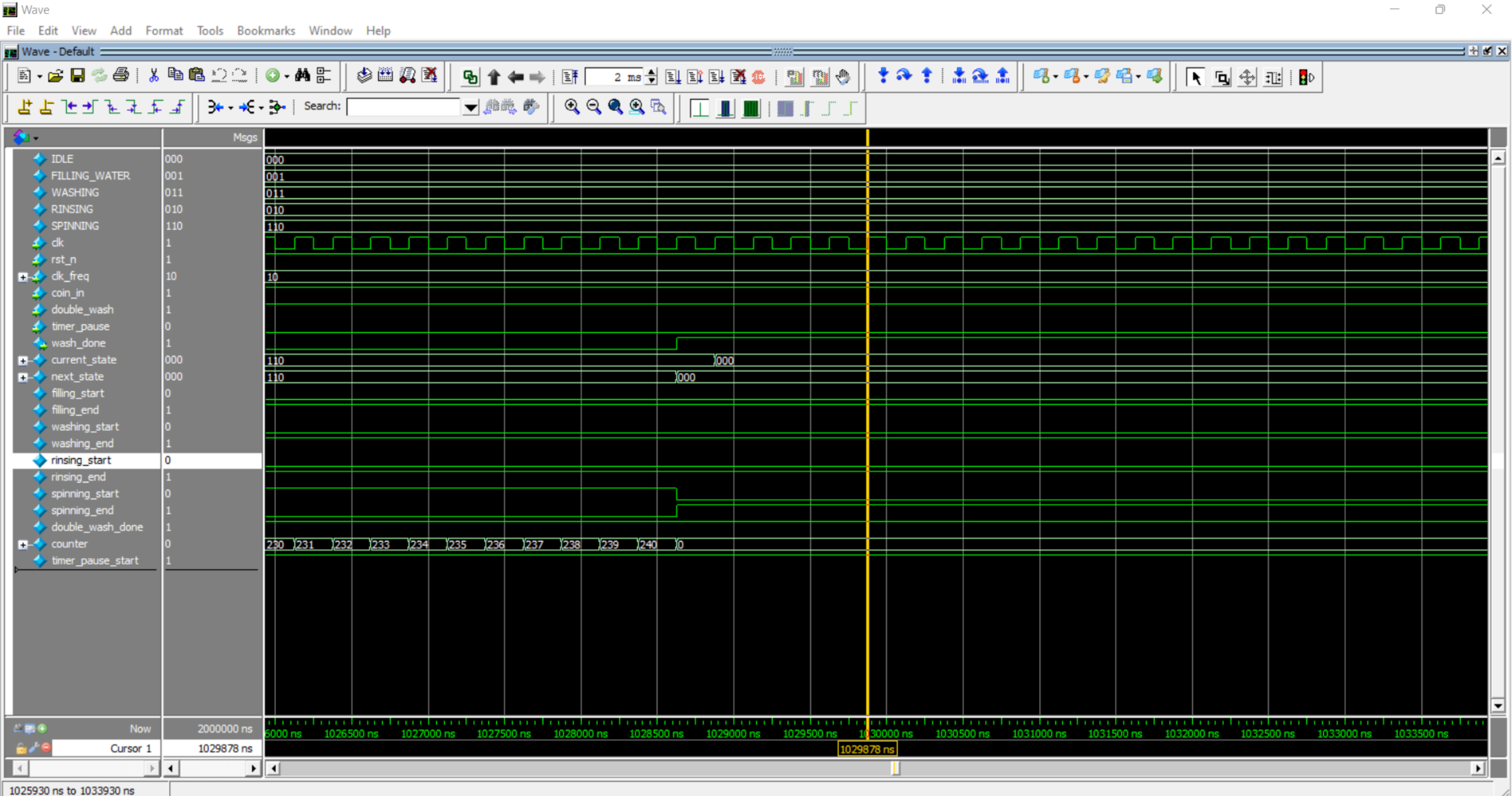
After the second rinsing state is done. (2 Minutes = 480 clock cycles after scaling down)



Timer pause button is pressed during spinning.



After the spinning state is done (1 minute = 240 clock cycles after scaling down).



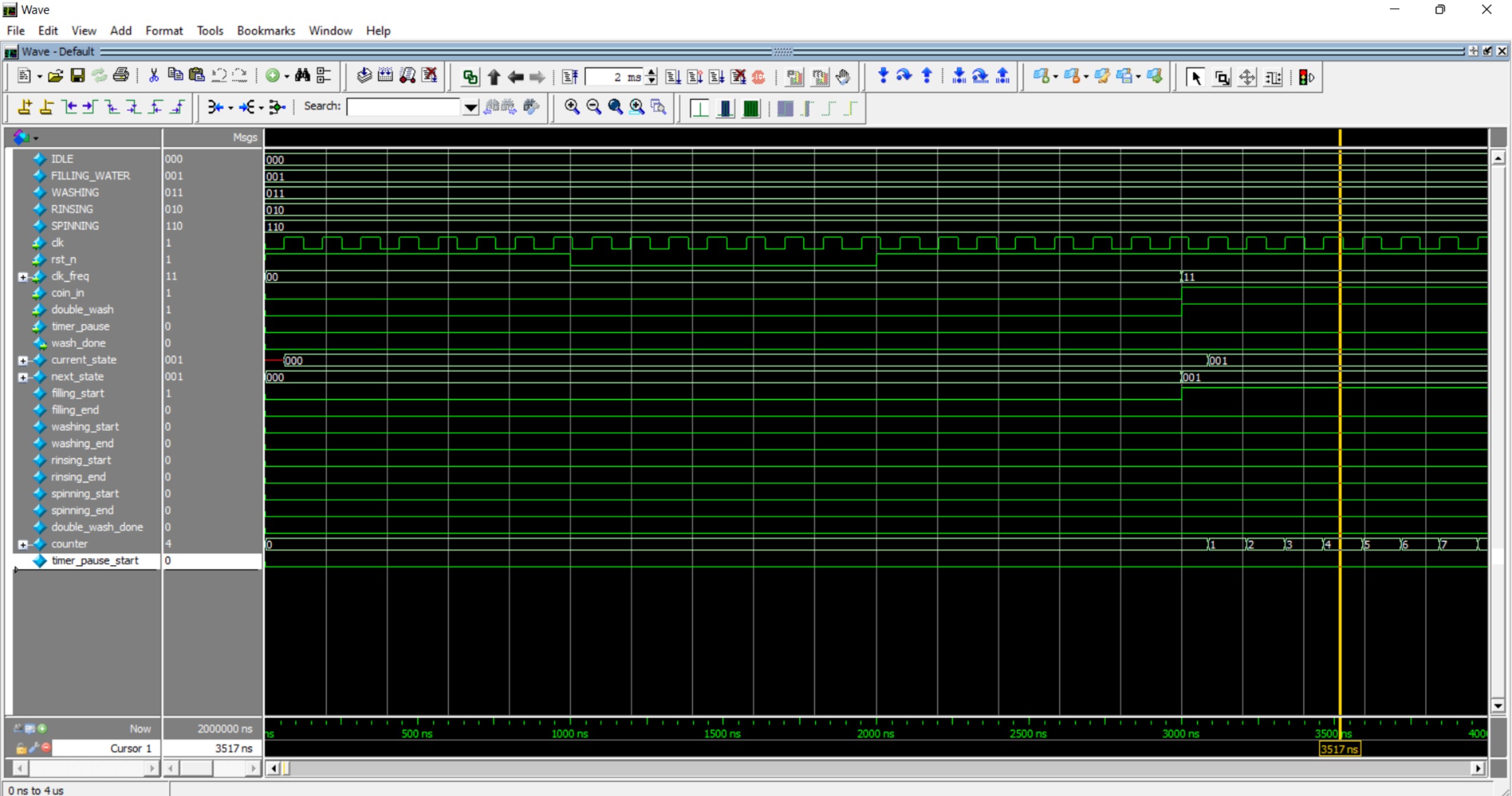
Initial Block (Test Case 6)

Here, we are testing the design knowing that the double wash and timer pause buttons are both pressed. Next are the simulation results from ModelSim software. (8 MHz)

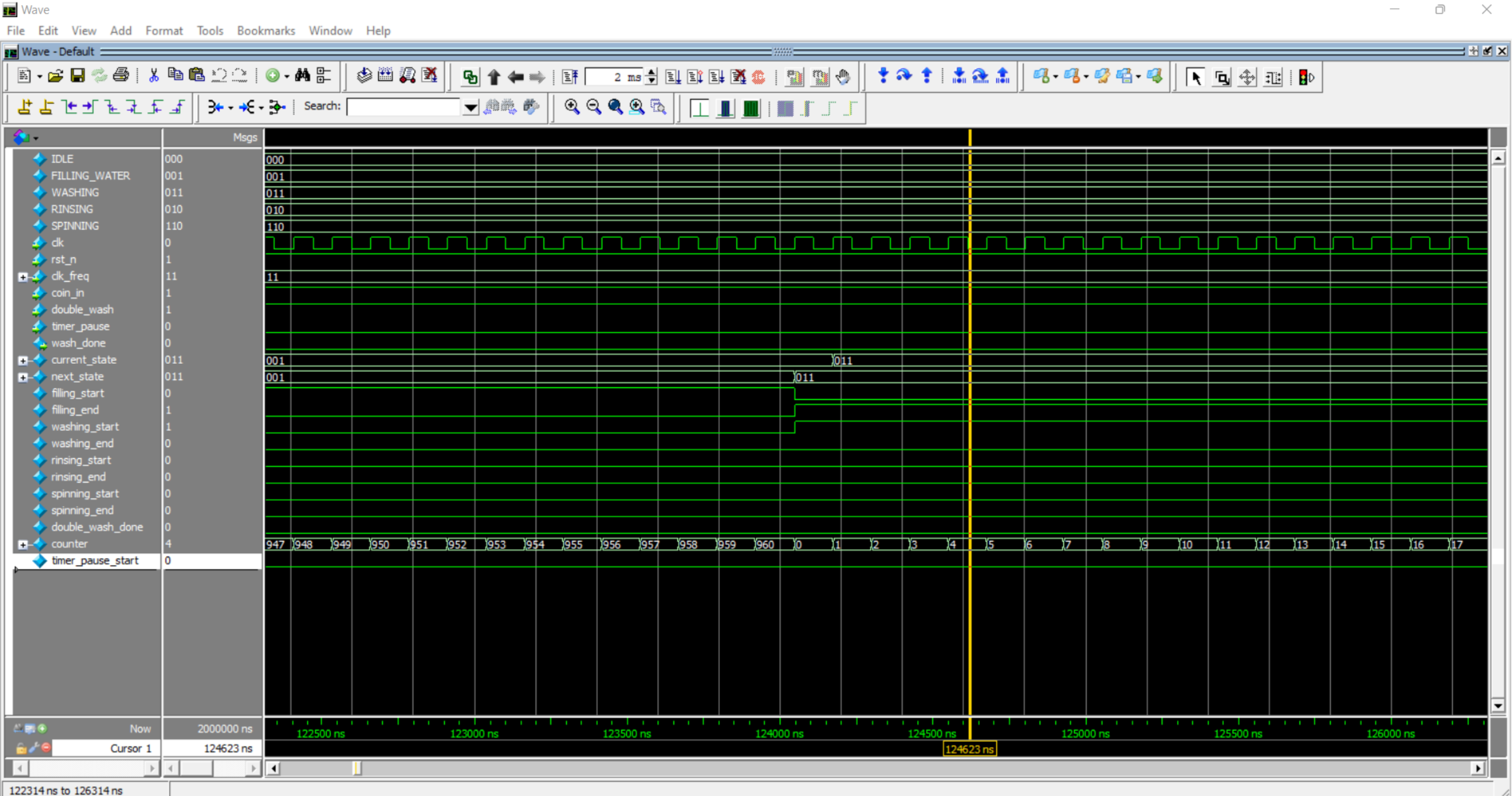
```
98  ////////////////////////////////////////////////////
99  ////////////////// Clock Generator  ////////////////////
100 ////////////////////////////////////////////////////
101
102 //always #0.5    clk_tb = ~clk_tb ;   // period = 1 us       (1 MHz)
103 //always #0.25   clk_tb = ~clk_tb ;   // period = 0.5 us     (2 MHz)
104 //always #0.125  clk_tb = ~clk_tb ;   // period = 0.25 us    (4 MHz)
105   always #0.0625 clk_tb = ~clk_tb ;   // period = 0.125 us   (8 MHz)
106
```

```
29  initial begin
30
31   // Save Waveform
32     $dumpfile("washing_machine.vcd") ;
33     $dumpvars;
34
35   // initialization
36     initialize() ;
37
38   // Reset
39     reset() ;
40
41   // double_wash
42     double_wash() ;
43     clk_freq_tb    = 2'b11 ;      // frequency is 8 MHz
44     coin_in_tb     = 1'b1  ;      // coin is deposited
45
46  #975
47   // Timer_Pause
48     Timer_Pause() ;
49
50  #3000
51  $finish ;
52  end
```

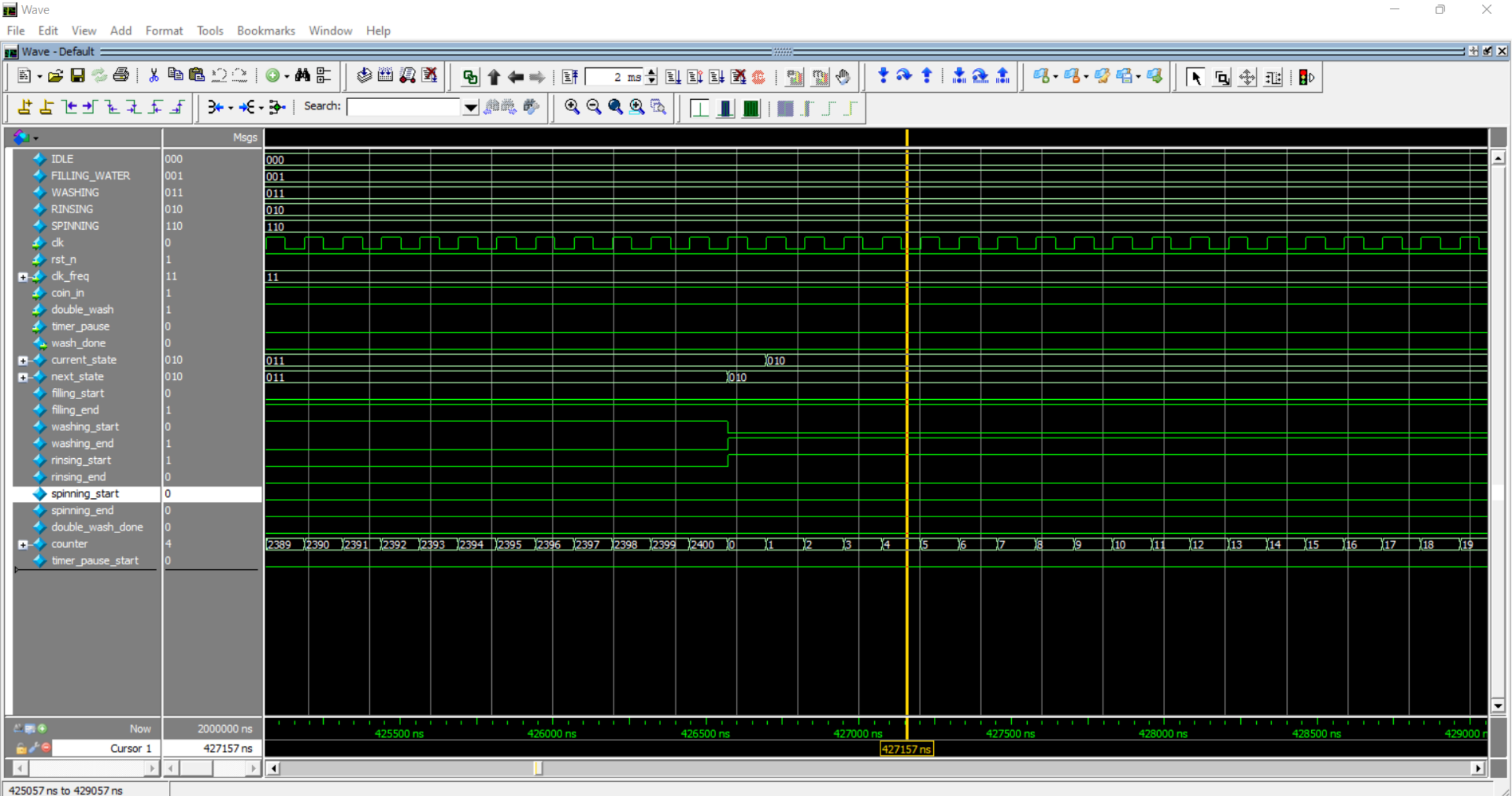
At the beginning.



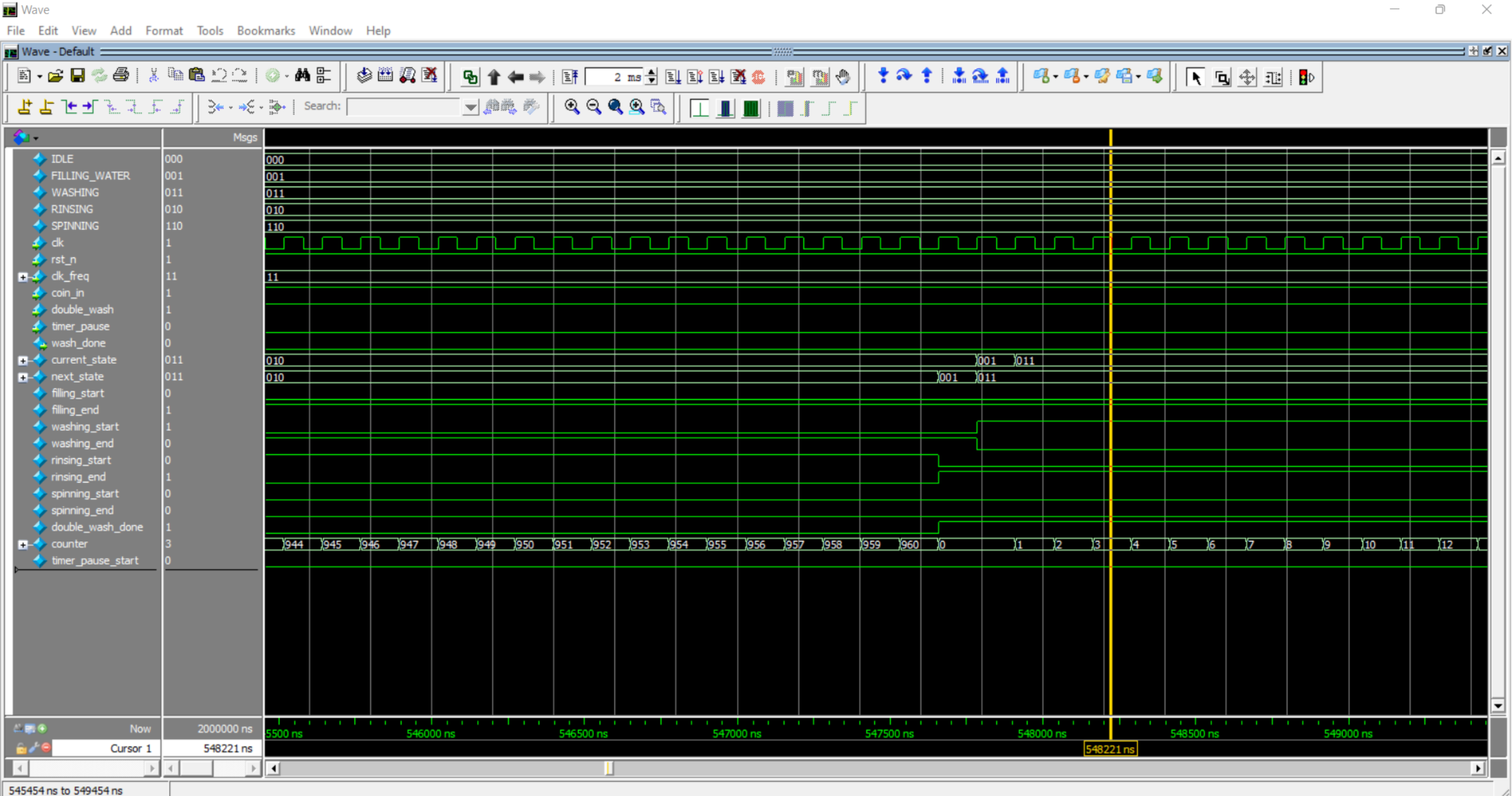
After the filling state is done. (2 Minutes = 960 clock cycles after scaling down)



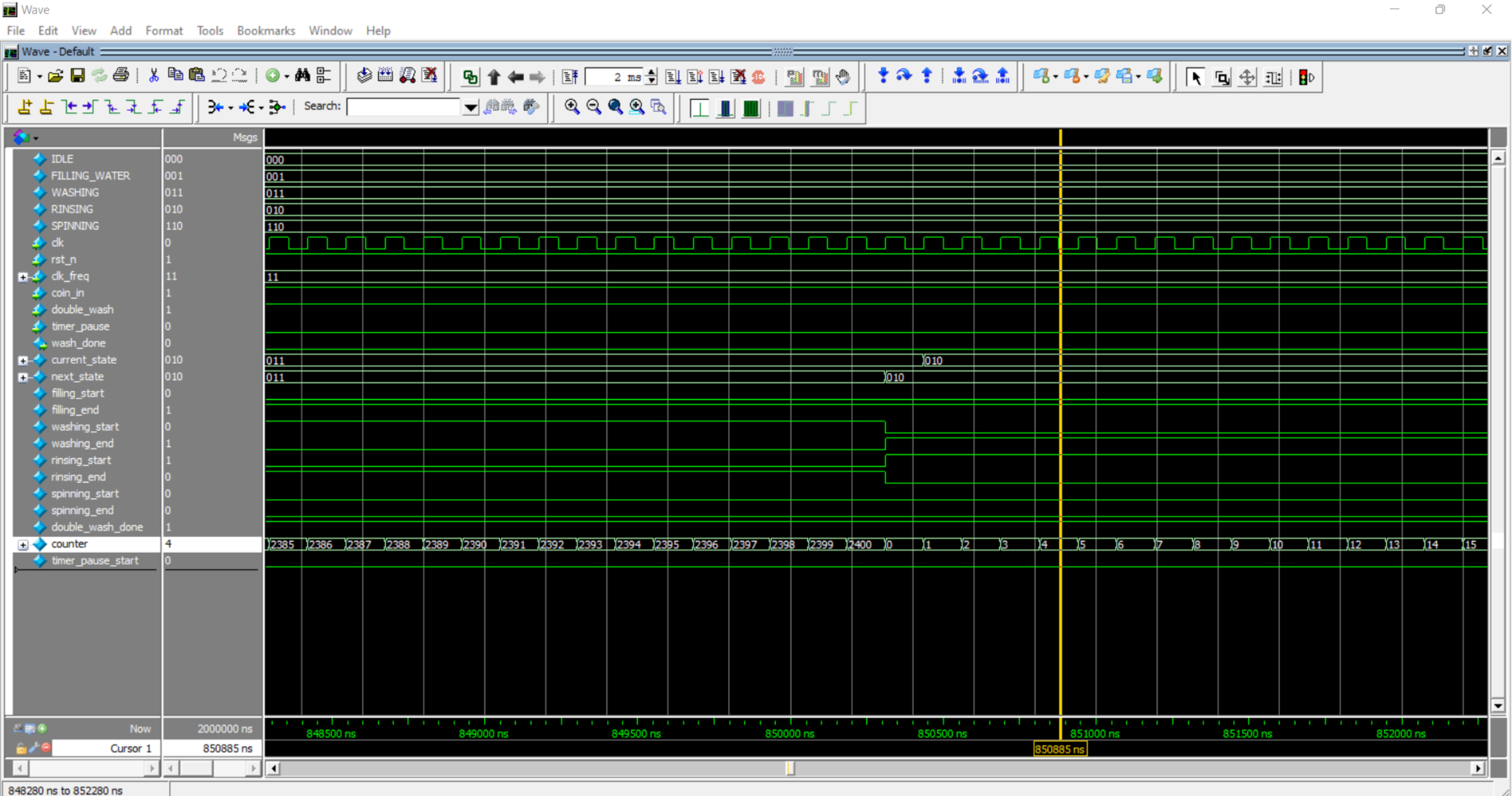
After the first washing state is done. (5 Minutes = 2400 clock cycles after scaling down)



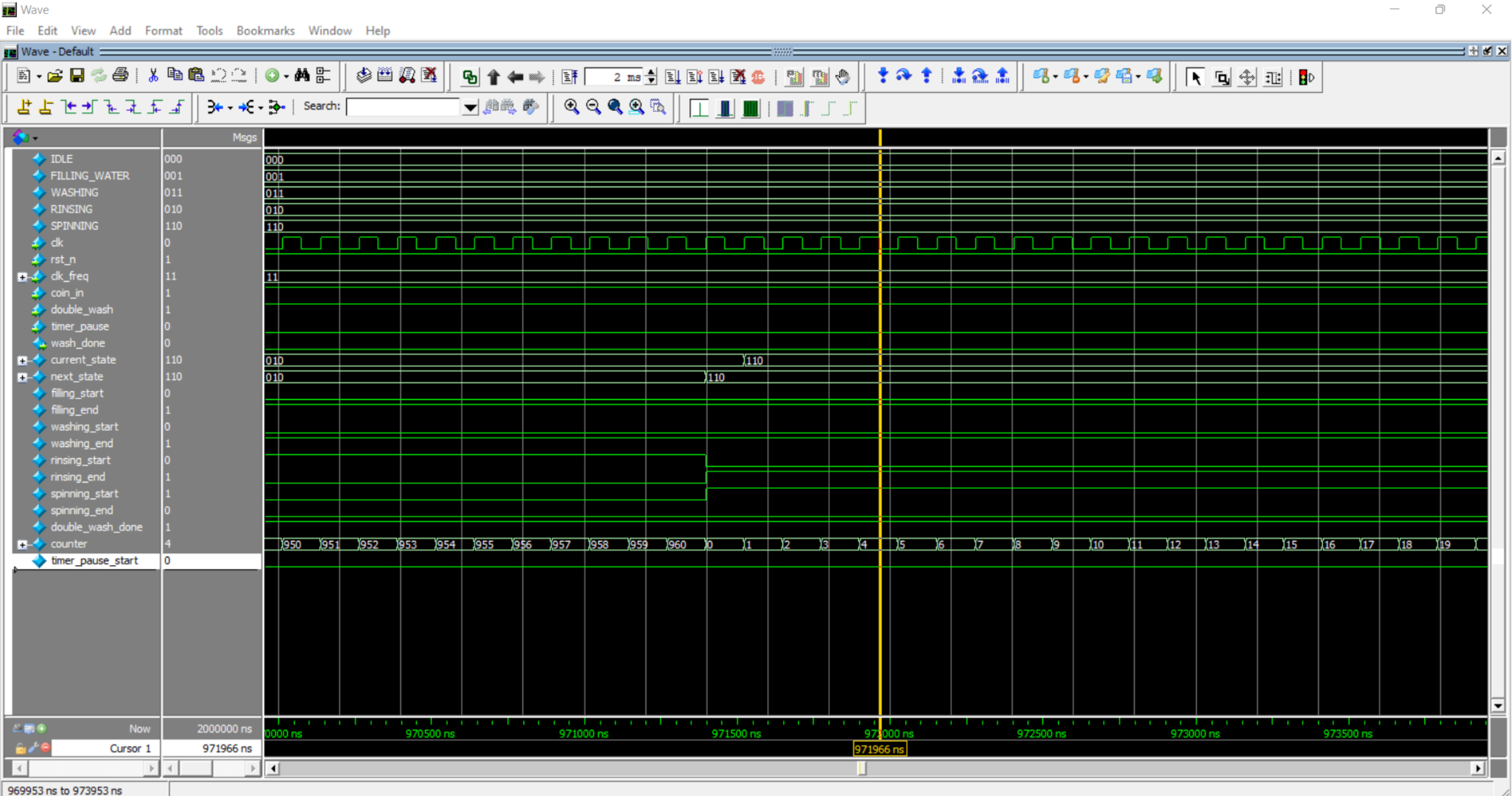
After the first rinsing state is done. (2 Minutes = 960 clock cycles after scaling down)



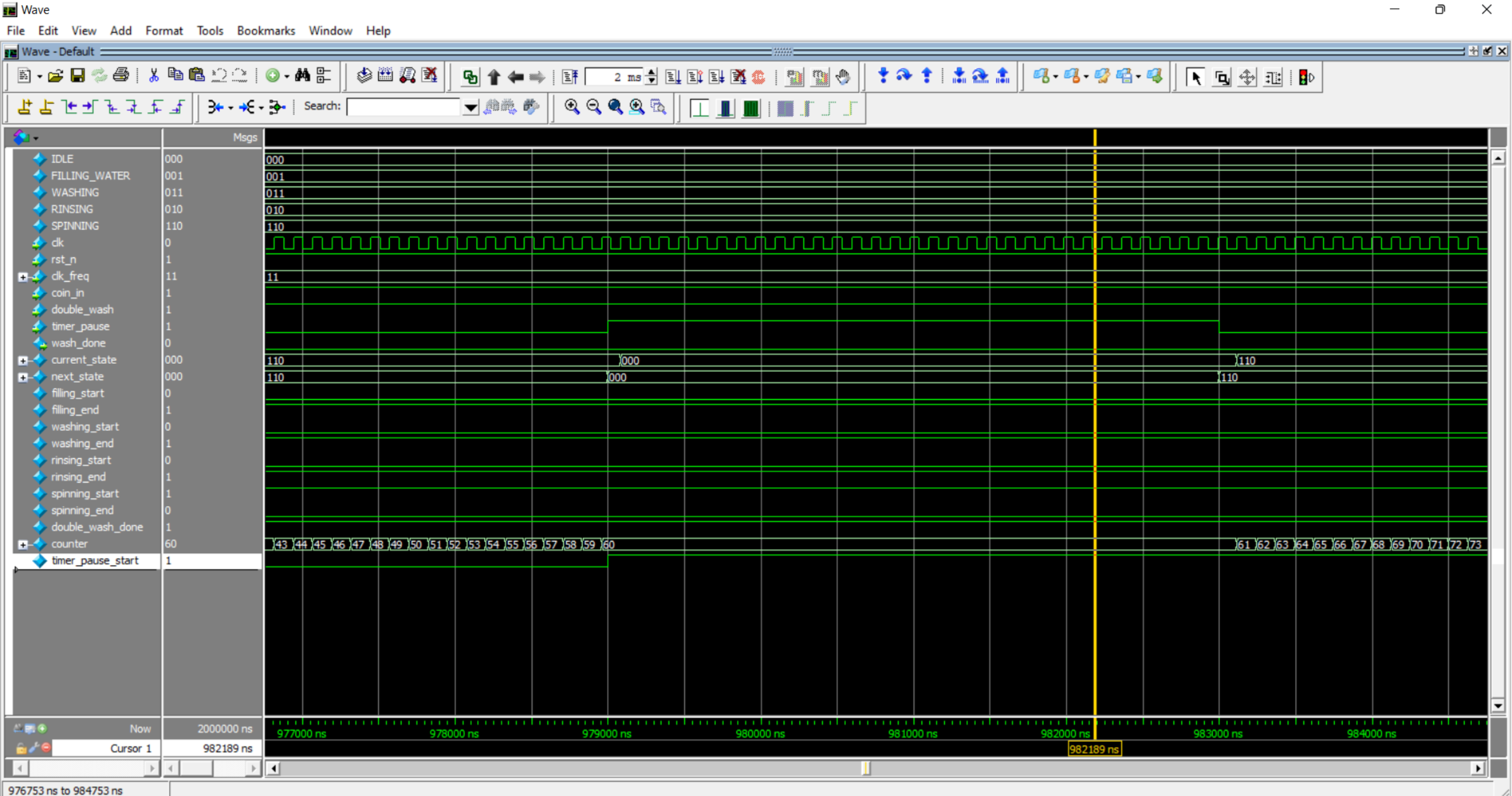
After the second washing state is done. (5 Minutes = 2400 clock cycles after scaling down)



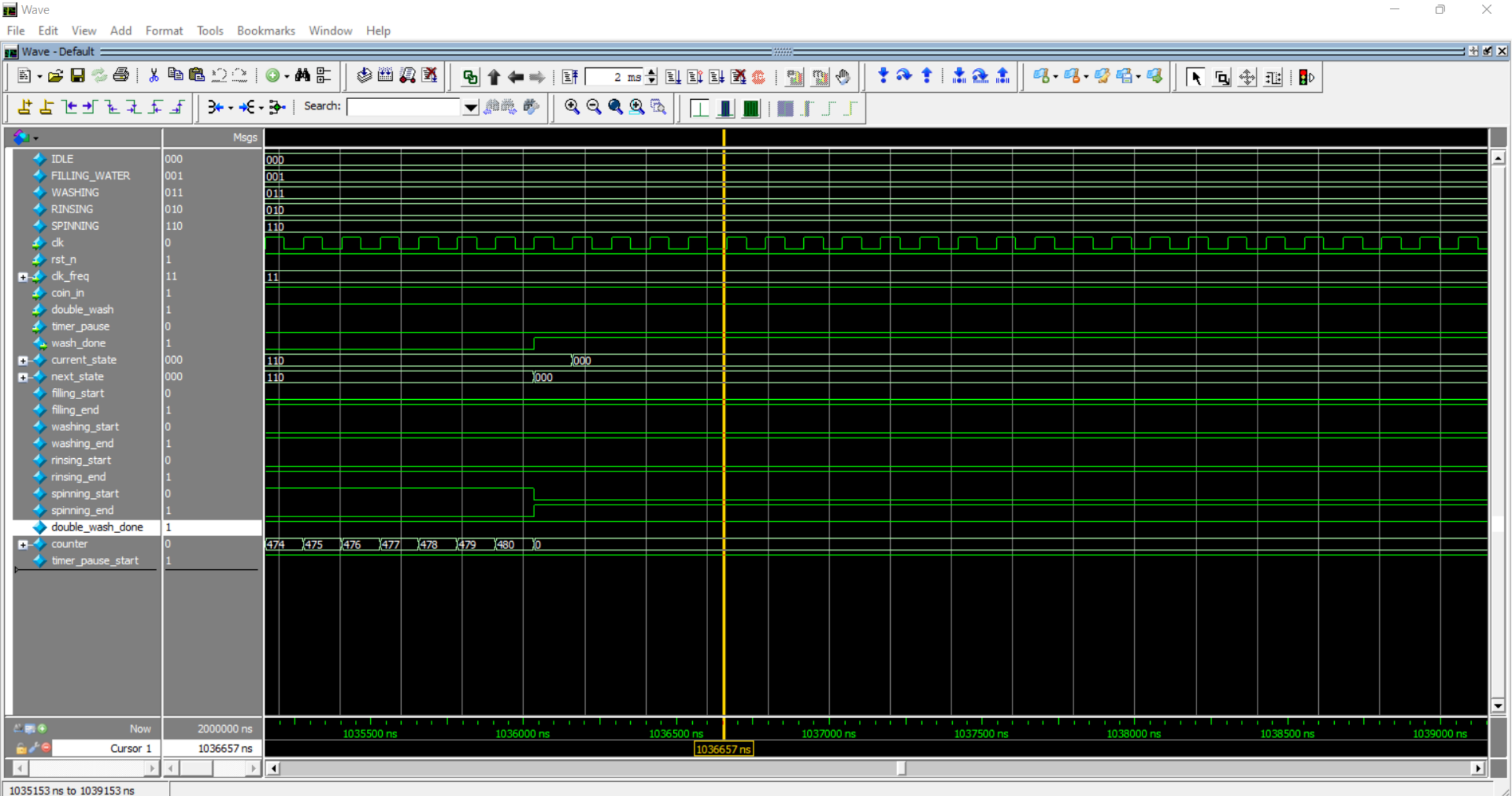
After the second rinsing state is done. (2 Minutes = 960 clock cycles after scaling down)



Timer pause button is pressed during spinning.



After the spinning state is done (1 minute = 480 clock cycles after scaling down).



Contact Info.

Yousef Sherif
LinkedIn: https://www.linkedin.com/in/yousef-sherif-6343b219b/
Email: shirefy49@gmail.com
Phone Number: 01098307950

Design and Testbench Link
https://drive.google.com/drive/folders/1rP7rD-fCSyzJzpP3ytStmZZrxpRLwOrx?usp=sharing