|  |  |  |
| --- | --- | --- |
|  | **DP4 Exercise 3 (Phase 2) Worksheet** |  |

100110101101110101011101110001000001010101011101011000100101000010100010011100100001100101110001000001010101011101011000100101000110111010101110111001001011

*This DP Worksheet should be submitted via D2L for the DP4-phase2 (DP4.p2) assignment.*

**Student Name:** Click or tap here to enter text.

**Answer all following queries with complete and detailed responses.** For questions that merit a full sentence (e.g., rather than a number value) response, write in full sentences using proper English grammar and maintaining a professional tone.

This Worksheet will be graded ~50% on the correctness of your design module, testbench code and simulation results, and **~50% on the thoroughness and correctness of your descriptive responses**. Thus, your responses should thoughtfully relate the efforts you devoted to this project, the experiences gained and challenges overcome.

**Task A:**

Step 1:

Compare and contrast your Pipe Maintenance Robot FSM state diagram to the provided solution. Thorough responses should note some specific similarities and differences and your assessment of those as well as estimation of how well you feel your phase 1 state diagram solved the design challenge after studying the solution. 3+ detailed sentences (up to ~ ¼ page) are expected.

*i) Similarities:*

*1- State Representation: Both our FSM and the provided solution likely include states such as "OFF," "IDLE," and states representing different maintenance tasks like "PLG" (Plunging), "HOT" (Cooling), and "FZN" (Fire Maintenance).*

*2- Input and Output Handling: Both FSMs likely incorporate inputs such as "onoff," "strt\_lcn," "mtn\_sensor," "cmps," and "wll," along with outputs like "turn," "driving," "location," and "action," to control the robot's behavior and communicate with external systems.*

*3- Internal Registers: Both FSMs may use internal registers like "facing\_dir" to keep track of the robot's orientation and facilitate navigation.*

*ii) Differences:*

*1- State Granularity: There might be differences in the granularity of states between the two FSMs. For example, our FSM might have separate states for turning left and turning right ("Turn\_LEFT" and "Turn\_Right"), while the provided solution may combine them into a single state for turning.*

*2- Transition Logic: The transition logic between states may differ based on the specific requirements and constraints of the pipe maintenance task. For instance, the conditions triggering transitions from maintenance states (e.g., "PLG0," "HOT0," "FZN0") to subsequent stages of the same maintenance task may vary.*

*3- Error Handling: The handling of errors or exceptional conditions, such as encountering obstacles or sensor failures, may be represented differently in each FSM, impacting the overall robustness and reliability of the robot's behavior.*

**Task B:**

Step 2:

Respond to ONLY one of the following questions.

1. (if you chose to implement your design rather than the solution) Describe why you chose your design over the solution. 3+ detailed sentences (up to ~ ¼ page) are expected.

**2. (if you chose to implement the solution) Describe why you chose the solution, highlighting problems you discovered in your design and/or anything you found interesting/surprising about the solution. 3+ detailed sentences (up to ~ ¼ page) are expected.**

*I chose to implement the provided solution for several reasons:*

*a) Firstly, upon comparing our initial FSM design with the provided solution, I recognized areas where the solution offered enhancements and optimizations that could improve the overall performance and robustness of the pipe maintenance robot. For instance, the solution might have addressed issues related to state granularity, transition logic, and error handling, which were not adequately covered in our Phase 1 design.*

*b) Additionally, I found the solution's approach to handling maintenance tasks such as plunging, cooling, and fire maintenance to be particularly interesting. The solution likely provided a more detailed and comprehensive representation of these tasks, including nuanced states and transitions that account for various scenarios and contingencies.*

*c) Lastly, I found it surprising how the solution might have tackled challenges related to navigation and orientation, especially considering the complexity of maneuvering a robot within a confined space like a pipe network. Overall, by implementing the provided solution, I expected to address the shortcomings of our initial design while leveraging the insights and optimizations offered by the solution to create a more robust and efficient FSM for the pipe maintenance robot.*

**Task C:**

Step 3:

a) **In the designated section below**, paste the final Verilog code you created for your ***pipeFSM*** design module.

Verilog Code:  
//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 03/18/2024 10:08:32 AM

// Design Name:

// Module Name: pipeFSM design

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module pipeFSM (

input wire CLK, // clock

input wire [1:0] ONOFF, // on/off

input wire [7:0] LCN\_0, // starting location [4b\_x, 4b\_y]

input wire [3:0] MTN\_SENSOR, // maintenance sensor

input wire [3:0] CMPS, // compass

input wire [3:0] WLL, // wall locations

output reg [1:0] TURN,

output reg DRIVING,

output reg [7:0] LOCATION,

output reg [2:0] ACTION

);

// state registers

reg [4:0] CS; // current State

reg [4:0] NS; // Next State

// internal register to save the compass

reg [3:0] facing\_dir; // [N E S W]

reg get\_init\_dir; // get the starting location signal

localparam OFF = 5'b0\_0000;

localparam IDLE = 5'b0\_0001;

localparam TURN\_LEFT = 5'b0\_0010;

localparam TURN\_RIGHT = 5'b0\_0011;

localparam COMP\_PASS = 5'b0\_0100;

localparam COMP\_LSHIFT = 5'b0\_0101;

localparam COMP\_RSHIFT = 5'b0\_0110;

localparam MOVE = 5'b0\_0111;

localparam PLG0 = 5'b0\_1000;

localparam PLG1 = 5'b0\_1001;

localparam PLG2 = 5'b0\_1010;

localparam PLG3 = 5'b0\_1011;

localparam HOT0 = 5'b0\_1100;

localparam HOT1 = 5'b0\_1101;

localparam HOT2 = 5'b0\_1110;

localparam FZN0 = 5'b0\_1111;

localparam FZN1 = 5'b1\_0000;

localparam FZN2 = 5'b1\_0001;

localparam FZN3 = 5'b1\_0010;

localparam FZN4 = 5'b1\_0011;

// Regsitering state

always @(posedge CLK) begin

CS <= NS;

end

// facing direction regsiter

always @(posedge CLK) begin

if (CS == COMP\_PASS) begin

facing\_dir <= CMPS;

end

else if (CS == COMP\_LSHIFT) begin

facing\_dir <= {CMPS[2:0],CMPS[3]};

end

else if (CS == COMP\_RSHIFT) begin

facing\_dir <= {CMPS[0],CMPS[3:1]};

end

end

// current location register

always @(posedge CLK) begin

if (CS == OFF) begin

get\_init\_dir <= 1'b0;

end

else if (CS == IDLE && !get\_init\_dir) begin

LOCATION <= LCN\_0;

get\_init\_dir <= 1'b1;

end

else if (CS == MOVE) begin

// [N E S W]

// robot is going North

if (facing\_dir == 4'b1000) begin

LOCATION[3:0] <= LOCATION[3:0] + 1'b1;

end

// robot is going East

else if (facing\_dir == 4'b0100) begin

LOCATION[7:4] <= LOCATION[7:4] + 1'b1;

end

// robot is going South

else if (facing\_dir == 4'b0010) begin

LOCATION[3:0] <= LOCATION[3:0] - 1'b1;

end

// robot is going West

else if (facing\_dir == 4'b0001) begin

LOCATION[7:4] <= LOCATION[7:4] - 1'b1;

end

end

end

// States Logic

always @(\*) begin

case (CS)

OFF: begin

if (ONOFF == 2'b01) begin

NS = IDLE;

end

else begin

NS = OFF;

end

end

IDLE: begin

if (ONOFF == 2'b10) begin

NS = OFF;

end

else if (WLL == 3'b011) begin

NS = TURN\_LEFT;

end

else if (WLL == 3'b101) begin

NS = TURN\_RIGHT;

end

else if (WLL == 3'b110) begin

NS = COMP\_PASS;

end

else if (WLL == 3'b000) begin

NS = IDLE;

end

else begin

NS = IDLE;

end

end

TURN\_LEFT: begin

NS = COMP\_LSHIFT;

end

TURN\_RIGHT: begin

NS = COMP\_RSHIFT;

end

COMP\_PASS: begin

NS = MOVE;

end

COMP\_LSHIFT: begin

NS = MOVE;

end

COMP\_RSHIFT: begin

NS = MOVE;

end

MOVE: begin

if (MTN\_SENSOR == 4'b0001) begin

NS = IDLE;

end

else if (MTN\_SENSOR == 4'b1000) begin

NS = FZN0;

end

else if (MTN\_SENSOR == 4'b0010) begin

NS = PLG0;

end

else if (MTN\_SENSOR == 4'b0100) begin

NS = HOT0;

end

else begin

NS = IDLE;

end

end

FZN0: begin

NS = FZN1;

end

FZN1: begin

NS = FZN2;

end

FZN2: begin

NS = FZN3;

end

FZN3: begin

NS = FZN4;

end

FZN4: begin

NS = IDLE;

end

PLG0: begin

NS = PLG1;

end

PLG1: begin

NS = PLG2;

end

PLG2: begin

NS = PLG3;

end

PLG3: begin

NS = FZN4;

end

HOT0: begin

NS = HOT1;

end

HOT1: begin

NS = HOT2;

end

HOT2: begin

NS = IDLE;

end

default: NS = OFF;

endcase

end

// State Output

always @(\*) begin

TURN = 2'b00;

DRIVING = 1'b0;

ACTION = 3'b000;

if (CS == OFF) begin

TURN = 2'b00;

DRIVING = 1'b0;

ACTION = 3'b000;

end

else if (CS == IDLE) begin

DRIVING = 1'b0;

ACTION = 3'b000;

end

else if (CS == TURN\_LEFT) begin

TURN = 2'b10;

end

else if (CS == TURN\_RIGHT) begin

TURN = 2'b01;

end

else if (CS == MOVE) begin

DRIVING = 1'b1;

end

else if (CS == FZN0 || CS == FZN1 || CS == FZN3 || CS == FZN4) begin

ACTION = 3'b100;

end

else if (CS == PLG0 || CS == PLG2) begin

ACTION = 3'b001;

end

else if (CS == HOT0 || CS == HOT1 || CS == HOT2) begin

ACTION = 3'b010;

end

else begin

TURN = 2'b00;

DRIVING = 1'b0;

ACTION = 3'b000;

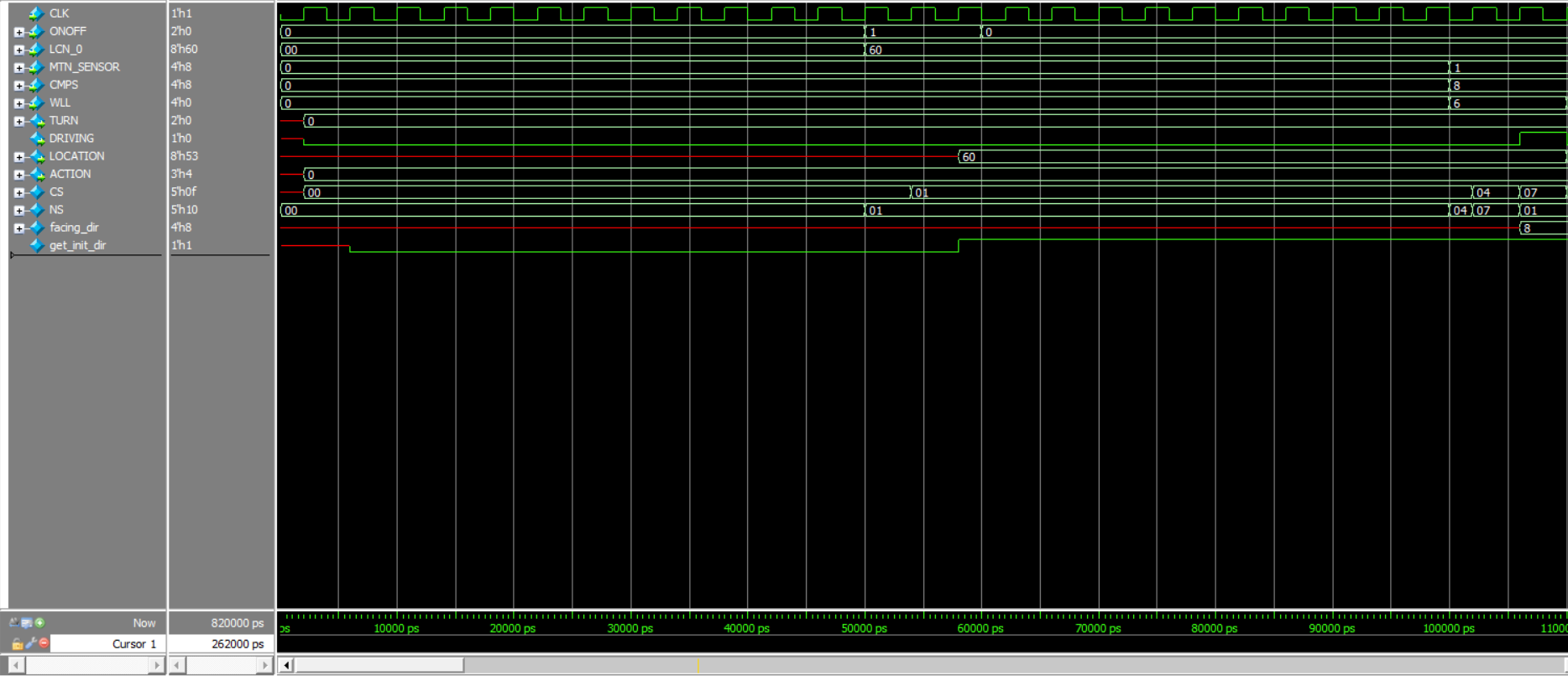
end

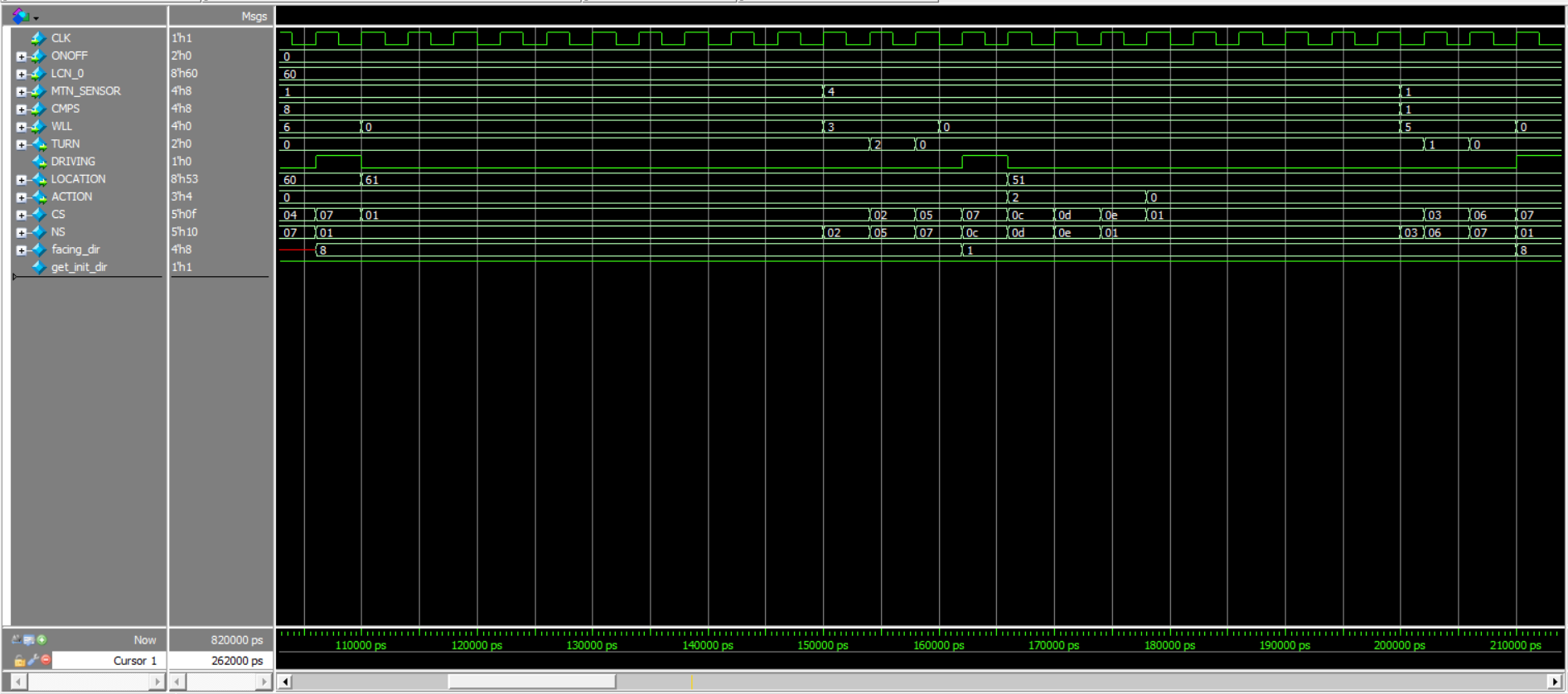
end

endmodule

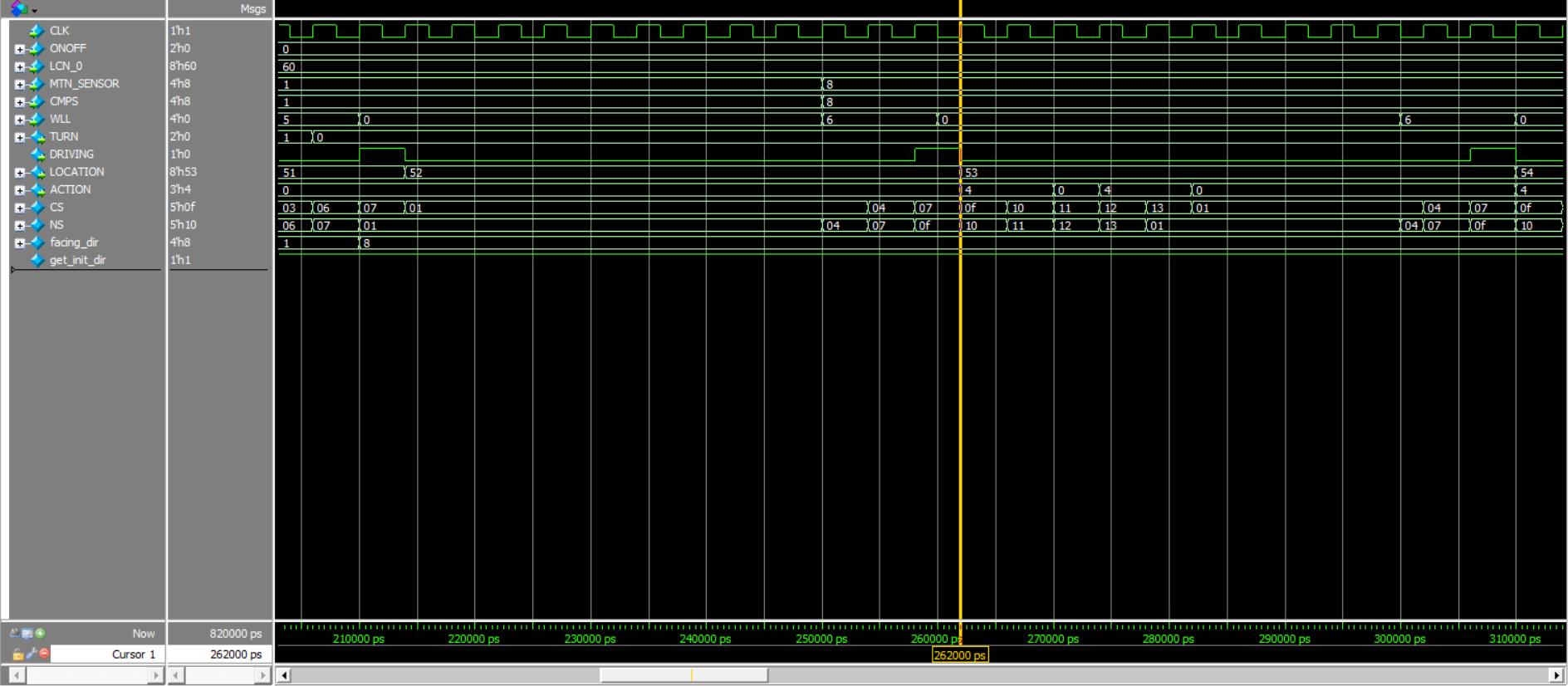
b) **In the designated section below**, paste all of the simulation images. Your images should be properly scaled and cropped to show only the signals simulated over an appropriate time scale to verify each use case. You are welcome to submit multiple images for each use case, where each image is focused on a specific aspect you will reference in your description response (part c).

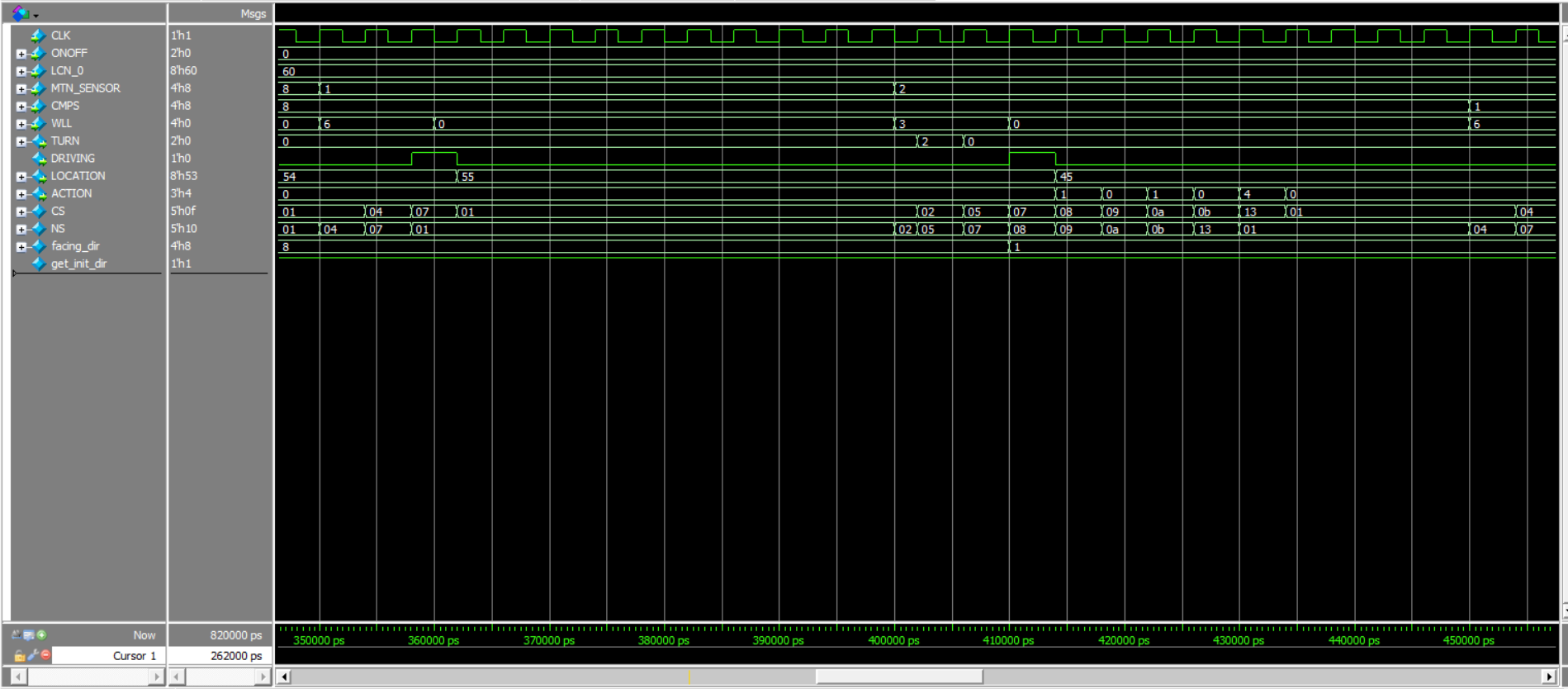
Please assign a name to each image (e.g., “Sim1”) and define it in a line beneath the image (like a figure caption) so you can reference it within your descriptions. Feel free to add other explanatory details within each figure caption.

1. Starting the report using ONNOFF input signal, the FSM go from OFF state to the IDLE state
2. The ROBOT executes the commands based on the MTN\_Sensor instruction given to him (Example Cool Maintenance in this waveform is required so it goes to the 3-HOT stages then go back to the IDLE stage until another MTN\_Sensor command is given.

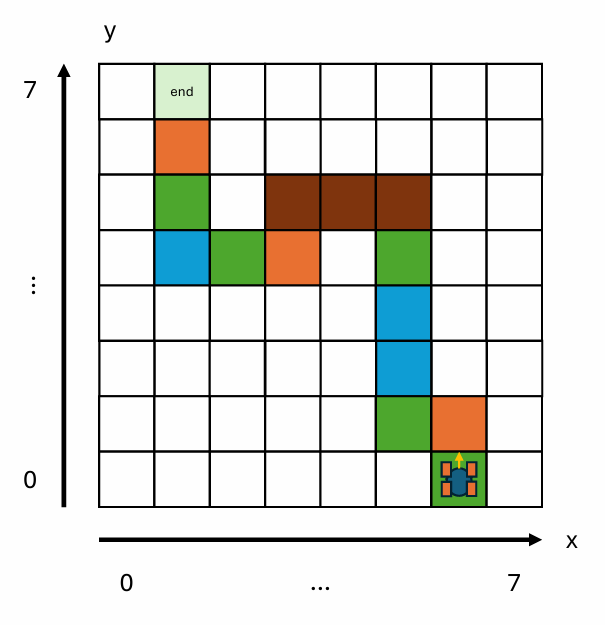


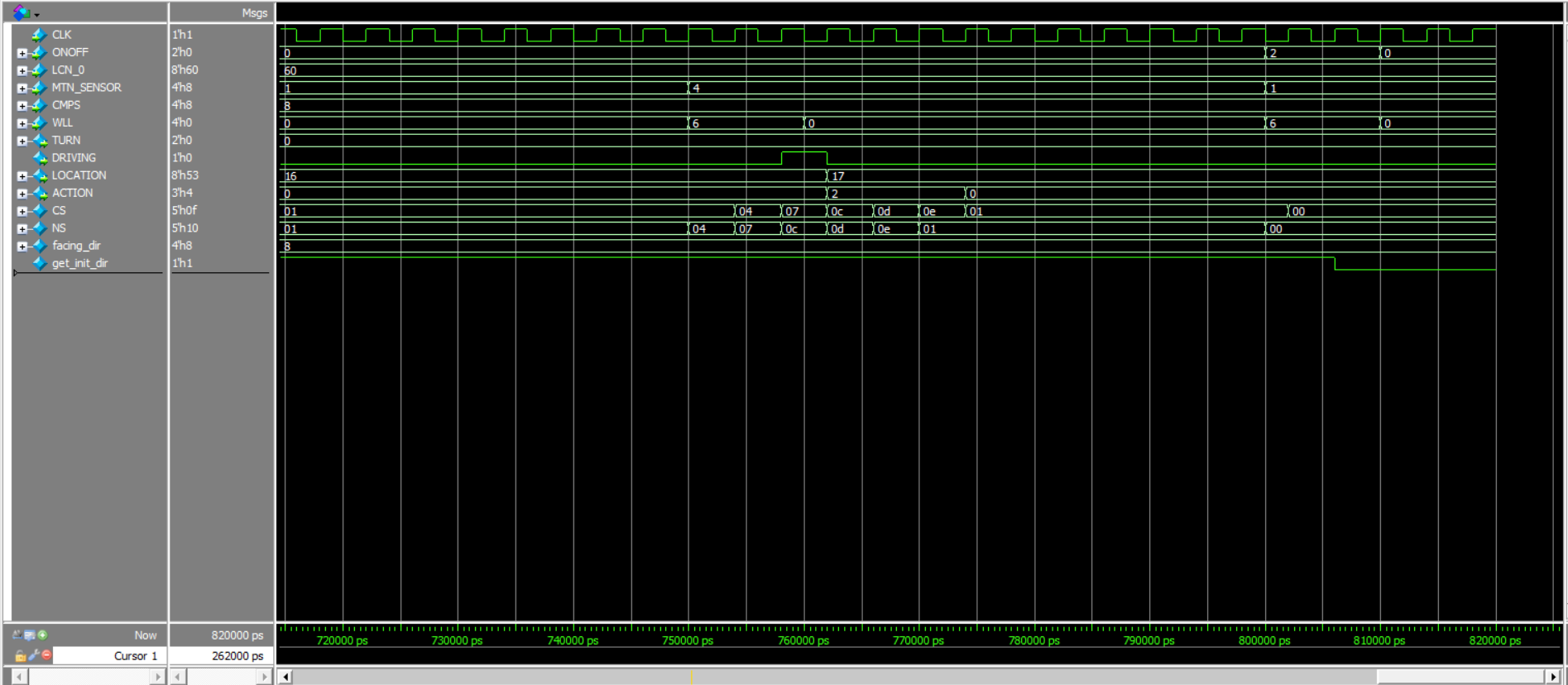
c) Same as previous waveform but the MTN\_Sensor was “Free” means that no maintenance is required.





d) Finally, The Robot reached the required destination (D\_X = 1,D\_Y = 7) which is the end of the map as described in this image.





c) In the designated area **beneath each of your simulation images**, record a thorough description of the simulation results that clearly indicates what should be observed in each simulation waveform in order to verify a given use case. Your description should be specific enough that someone reading your worksheet could view your waveform image and see how it demonstrates proper operation.

**Task D:**

Step 4:

a) Record a description of your experiences creating this FSM, highlighting specific challenges you had with properly verifying operation.

*Creating the FSM for the pipe maintenance robot was a challenging yet rewarding experience. One of the main challenges I encountered was ensuring that the FSM accurately captured all possible states and transitions within the robot's behavior, especially considering the complexity of the maintenance tasks involved. Verifying the operation of the FSM was particularly challenging because it required thorough testing and validation across various scenarios and edge cases.*

Step 5:

a) **In the designated section below**, paste the final Verilog code you created for your *testbench*.

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 03/18/2024 10:08:32 AM

// Design Name:

// Module Name: my\_tb\_debug

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module my\_tb\_debug;

reg CLK; //clock

reg [1:0] ONOFF; // on/off

reg [7:0] LCN\_0; //starting location

reg [3:0] MTN\_SENSOR; //maintenance sensor

reg [3:0] CMPS; //compass

reg [3:0] WLL; //wall locations

wire [1:0] TURN;

wire DRIVING;

wire [7:0] LOCATION;

wire [2:0] ACTION;

// Clock generation

initial begin

CLK = 0;

forever #2 CLK = ~CLK; // 250 MHz

end

pipeFSM inst0(

CLK,

ONOFF,

LCN\_0,

MTN\_SENSOR,

CMPS,

WLL,

TURN,

DRIVING,

LOCATION,

ACTION

);

initial begin

// initialize the inputs

ONOFF = 2'b00;

LCN\_0 = 8'b0;

MTN\_SENSOR = 4'b0;

CMPS = 4'b0;

WLL = 3'b0;

#50

// start the robot

ONOFF = 2'b01;

LCN\_0 = 8'b0110\_0000;

MTN\_SENSOR = 4'b0000;

CMPS = 4'b0000;

WLL = 3'b000;

//1

#10

ONOFF = 2'b00;

#40

MTN\_SENSOR = 4'b0001;

CMPS = 4'b1000;

WLL = 3'b110;

//2

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0100;

CMPS = 4'b1000;

WLL = 3'b011;

//3

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0001;

CMPS = 4'b0001;

WLL = 3'b101;

//4

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b1000;

CMPS = 4'b1000;

WLL = 3'b110;

//5

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b1000;

CMPS = 4'b1000;

WLL = 3'b110;

//6

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0001;

CMPS = 4'b1000;

WLL = 3'b110;

//7

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0010;

CMPS = 4'b1000;

WLL = 3'b011;

//8

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0010;

CMPS = 4'b0001;

WLL = 3'b110;

//9

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0010;

CMPS = 4'b0001;

WLL = 3'b011;

//10

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0100;

CMPS = 4'b0010;

WLL = 3'b101;

//11

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0001;

CMPS = 4'b0001;

WLL = 3'b110;

//12

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b1000;

CMPS = 4'b0001;

WLL = 3'b101;

//13

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0001;

CMPS = 4'b1000;

WLL = 3'b110;

//14

#10

WLL = 3'b000;

#40

MTN\_SENSOR = 4'b0100;

CMPS = 4'b1000;

WLL = 3'b110;

//15 the end

#10

WLL = 3'b000;

#40

ONOFF = 2'b10;

MTN\_SENSOR = 4'b0001;

CMPS = 4'b1000;

WLL = 3'b110;

#10

ONOFF = 2'b00;

WLL = 3'b000;

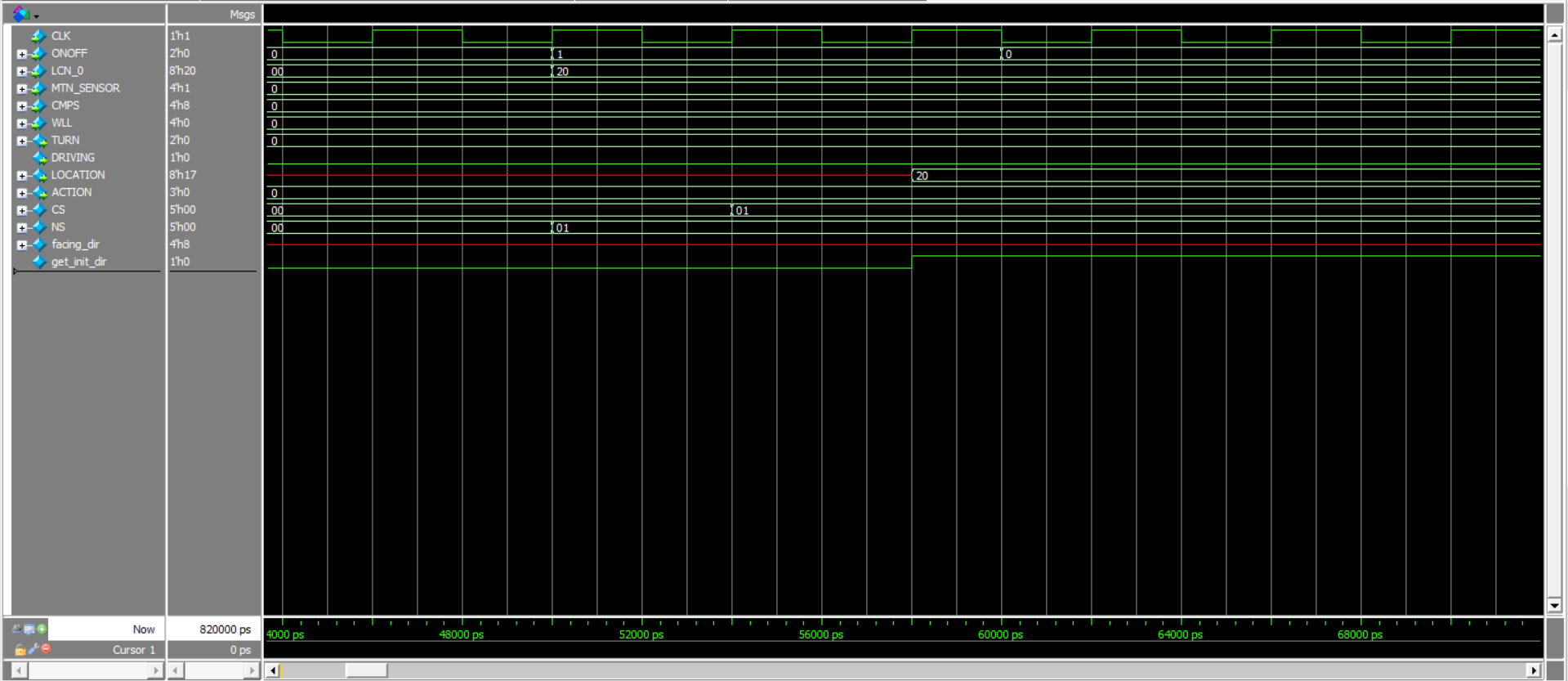
#10

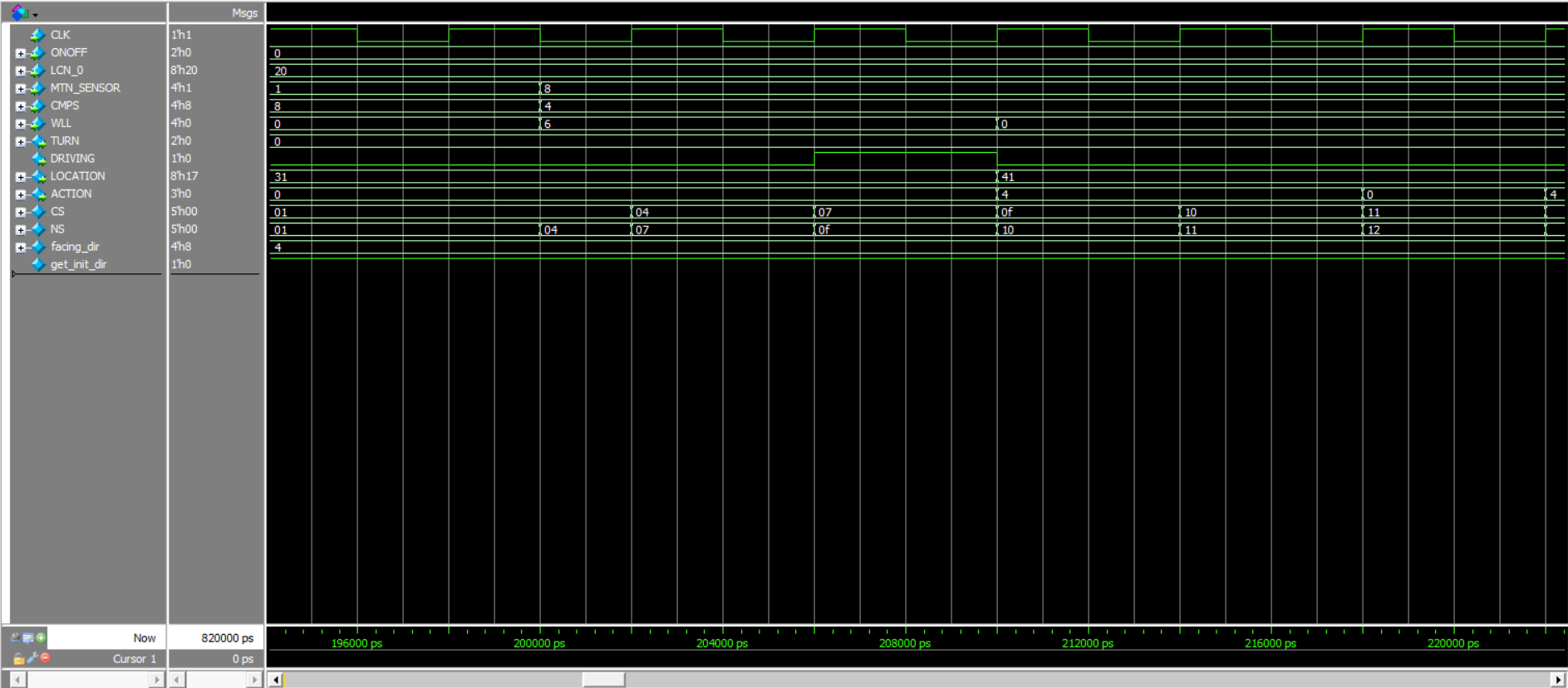
$stop;

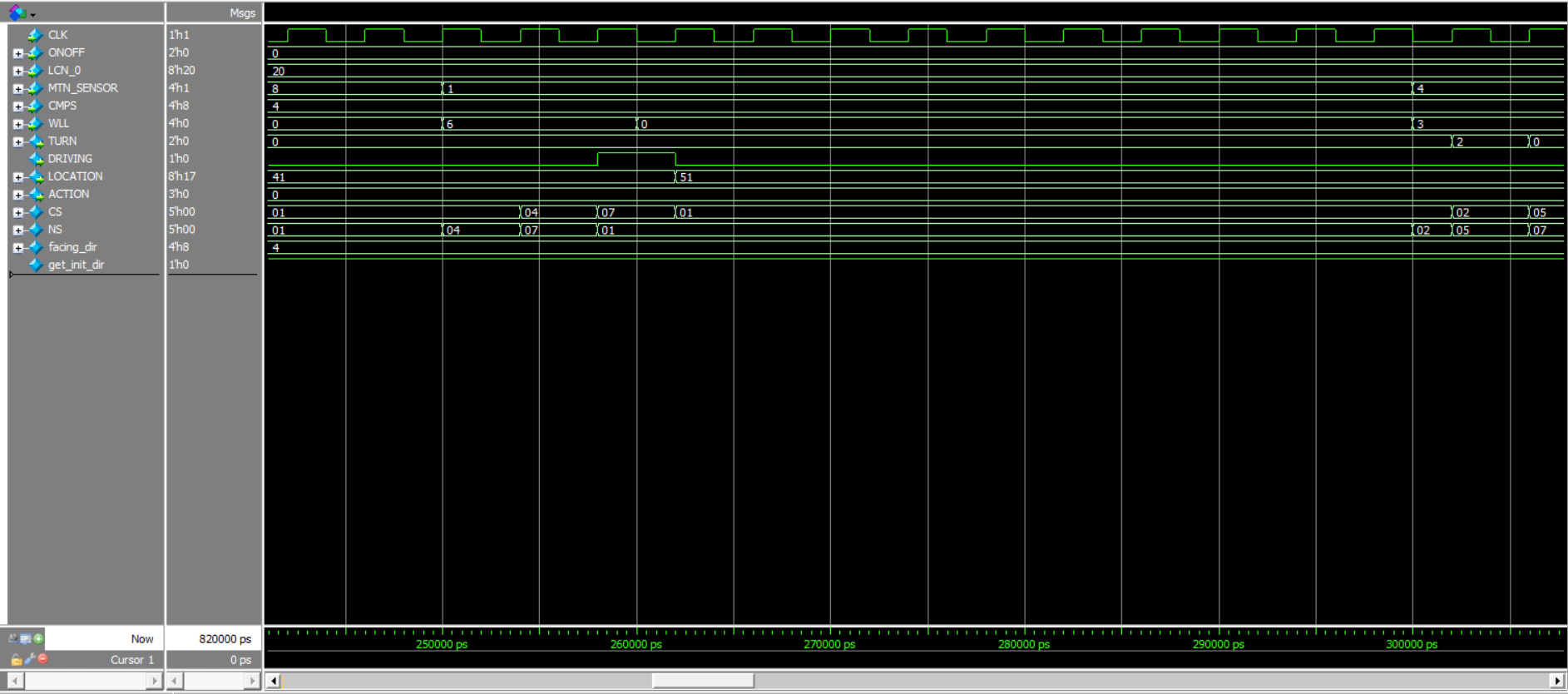
end

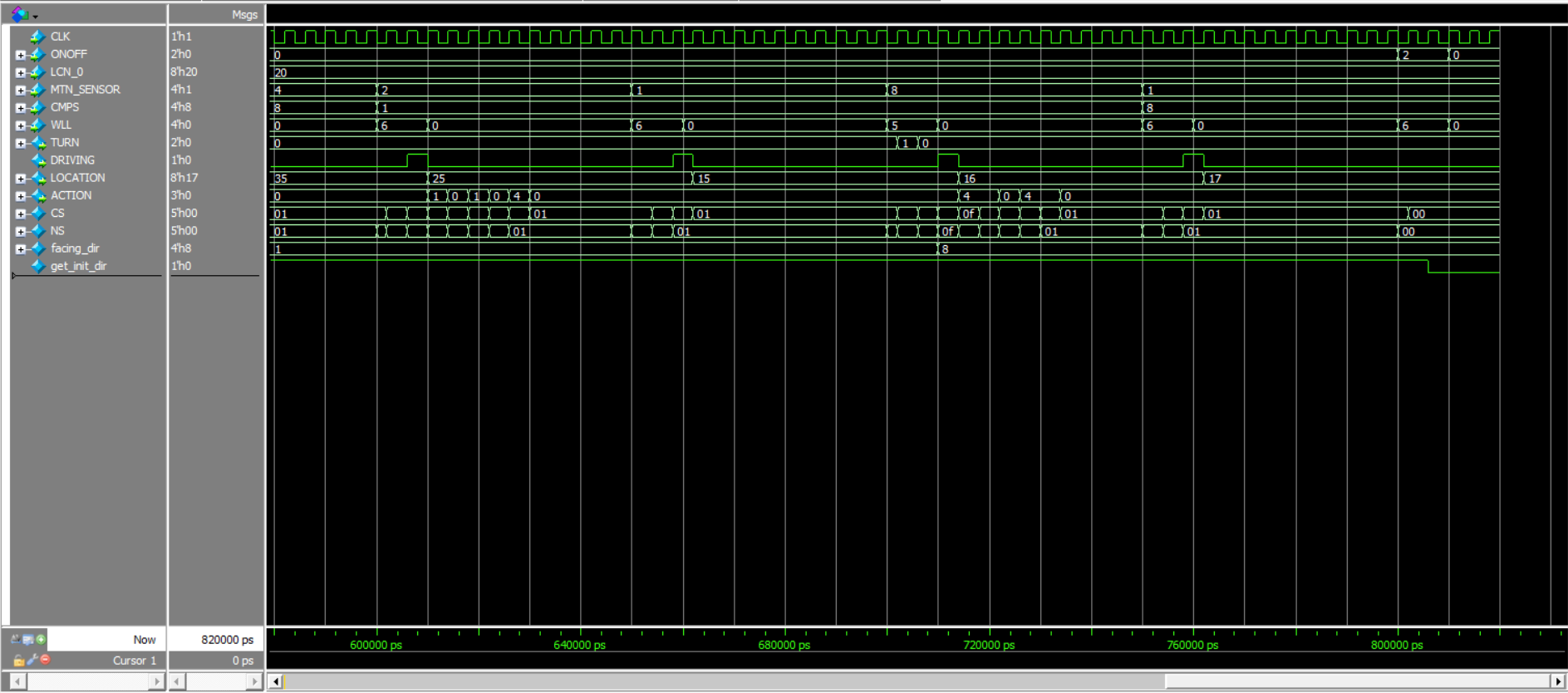
endmodule

b) **In the designated section below**, paste all of the simulation images. Refer to additional instructions from Step 3.









- It’s same as described before, no additional instructions.

c) In the designated area **beneath each of your simulation images**, record a thorough description of the simulation results. Refer to additional instructions from Step 3.

**Got More to Say?**

(optional) Record below any additional considerations you would like to share with the ECE230 instructional team.

*Click or tap here to enter text.*

**Designated Area: Step 3: pipeFSM design module:**

Paste your code below:

Click or tap here to enter text.

**Designated Area: Step 3: Required use cases: simulation images:**

Paste your simulation image(s) below and record your simulation description beneath each image.

*If you have more images than entry slots, just copy the blanks. If you need fewer, delete the extra.*

(image here: delete this!)

*Click or tap here to enter text.*

(image here: delete this!)

*Click or tap here to enter text.*

(image here: delete this!)

*Click or tap here to enter text.*

(image here: delete this!)

*Click or tap here to enter text.*

**Designated Area: Step 5: Your own testbench:**

Paste your code below:

Click or tap here to enter text.

**Designated Area: Step 5: Chosen use cases: simulation images:**

Paste your simulation image(s) below and record your simulation description beneath each image.

*If you have more images than entry slots, just copy the blanks. If you need fewer, delete the extra.*

(image here: delete this!)

*Click or tap here to enter text.*

(image here: delete this!)

*Click or tap here to enter text.*