<

Pipe Maintenance Robot Design & Simulation

Chenxin Zhang

1

***Abstract*—**This project aims to develop a pipeline inspection robot to address the challenges associated with pipeline inspection. By leveraging the concept of non-destructive evaluation, the robot can perform its assigned tasks without causing any damage to the pipelines. The robot's circuitry is designed to enable precise control of its movements within the pipes, allowing it to move forward and backward, identify potential pipeline issues, and provide feedback on detected problems. Furthermore, a Finite State Machine (FSM) is designed to model the robot's operations, and the corresponding Verilog code is implemented. Simulation tests are carried out using two predefined datasets, with waveforms generated to compare and validate the detection results.

***Index Terms*— NDE, Verilog, Xilinx Vivado, Finite State Machine**

I. INTRODUCTION

Pipe inspection in urban areas presents ongoing challenges. Pipes, with their complex structures, are often buried underground, making them difficult to inspect. Traditional inspection methods, like surface excavation, can damage existing infrastructure and lead to soil erosion. To overcome these challenges, non-destructive evaluation (NDE) technology offers a solution. NDE techniques, which include radiation, ultrasonics, infrared, electromagnetic fields, and eddy currents, allow for the assessment of materials, parts, and equipment without impacting their functionality or operational condition. In pipeline inspection, NDE is frequently employed, with techniques such as eddy current testing for high-speed inspections and current deflection for pipeline monitoring.

This report introduces a solution that leverages a pipeline inspection robot to support human efforts and eliminate the need for invasive underground inspections. The compact, agile robot moves through pipelines, gathering critical data, assessing the pipeline's condition, and alerting of any required repairs. Additionally, it performs maintenance tasks autonomously, effectively reducing the need for human intervention. This innovative robot embodies the principles of non-destructive testing.

The main goal of this article is to explain the technical foundation of non-destructive pipeline inspection robots using Verilog. Specific input parameters necessary for the robot's operation are carefully developed along with their outputs. With the help of Vivado Verilog code, the robot's functions are implemented. Furthermore, two extensive testbenches are designed, generating waveforms to replicate real-world scenarios. This research aims to provide essential technical support for the safe and sustainable operation of urban pipelines.

II. DESIGN

The Pipe Maintenance Robot should be equipped with essential features that allowed the robot in order to navigate and carry out the maintenance tasks within the pipes. These parameters enable accurate control over the robot, helping to achieve the desired outcomes effectively.

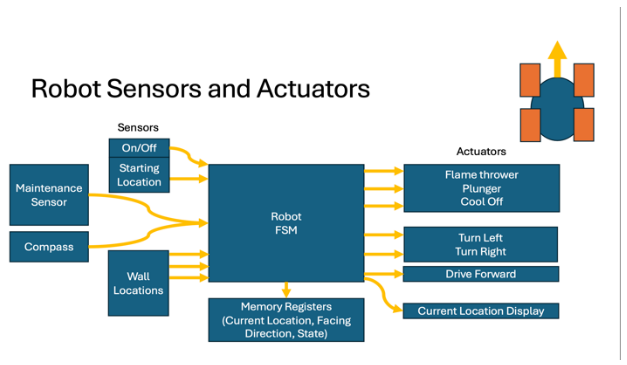
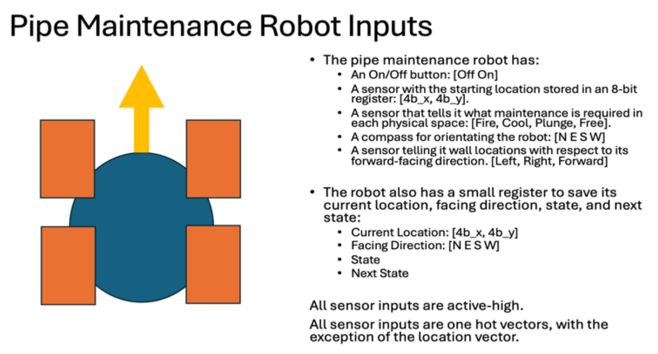


Fig. 1. Parameter of Robot.

The Finite State Machine (FSM) of the Pipe Maintenance Robot includes several key inputs. It features an [On, Off] button for activation control and a sensor linked to an 8-bit register that stores the robot's initial location, denoted as [4b_x, 4b_y]. A built-in compass helps orient the robot by providing directional indicators for North, East, South, and West (N, E, S, W).

The robot also uses sensors to identify walls in relation to its front-facing direction, distinguishing between Left, Right, and Forward (L, R, F) walls. It stores its current location, orientation, and status in a register, which aids in efficient navigation and task performance within the pipe environment.

All sensor inputs are active-high and represented as one-hot vectors, except for the location vector, ensuring precise control and smooth operation of the robot.


Pipe Maintenance Robot Inputs

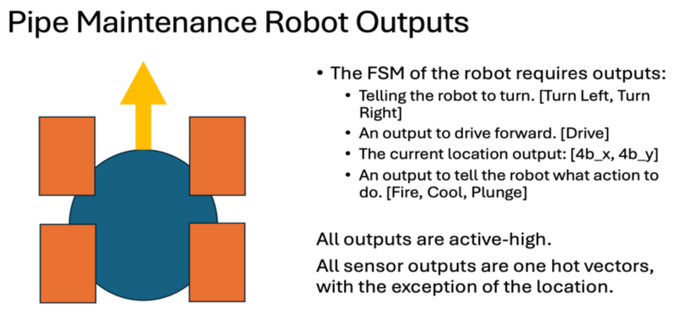
- The pipe maintenance robot has:
  - An On/Off button: [Off On]
  - A sensor with the starting location stored in an 8-bit register: [4b_x, 4b_y].
  - A sensor that tells it what maintenance is required in each physical space: [Fire, Cool, Plunge, Free].
  - A compass for orientating the robot: [N E S W]
  - A sensor telling it wall locations with respect to its forward-facing direction. [Left, Right, Forward]
- The robot also has a small register to save its current location, facing direction, state, and next state:
  - Current Location: [4b_x, 4b_y]
  - Facing Direction: [N E S W]
  - State
  - Next State

All sensor inputs are active-high.

All sensor inputs are one hot vectors, with the exception of the location vector.

Fig. 2. Input of Robot.

The outputs of the pipe maintenance robot's FSM provide essential commands and information for its operation and navigation within the pipeline. These include turn instructions—Turn Left and Turn Right—to adjust the robot's orientation as needed. Additionally, a forward-drive output enables the robot to advance through the pipe efficiently. The current location output, represented by [4b_x, 4b_y], offers critical positional data to track the robot's movement along the pipe. An output specifying the required action, such as "Fire," "Cool," or "Plunge," directs the robot in performing maintenance tasks based on the pipeline's condition. All outputs are active-high and represented as one-hot vectors, ensuring accurate control and effective communication between the robot and its operational environment.


Pipe Maintenance Robot Outputs

- The FSM of the robot requires outputs:
  - Telling the robot to turn. [Turn Left, Turn Right]
  - An output to drive forward. [Drive]
  - The current location output: [4b_x, 4b_y]
  - An output to tell the robot what action to do. [Fire, Cool, Plunge]

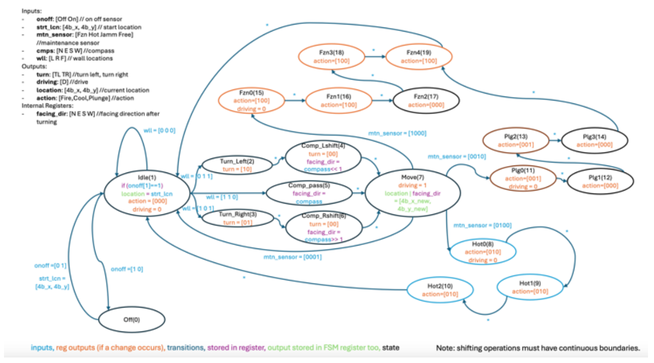
All outputs are active-high.

All sensor outputs are one hot vectors, with the exception of the location.

Fig. 3. Output of Robot.

Based on the configuration of relevant parameters and the initial design of the Pipe Robot's functions, a corresponding FSM diagram can be developed. This FSM consists of 20 states, including an "Off" state, an "Idle" state, two turning states, three "Compass Shift" states, one movement recognition state, three "Hot" states, four "Plunge" states, and five "Frozen" states.

The detailed FSM Verilog code design for the pipeline inspection robot is available in the repository. This Verilog code translates the FSM diagram and can be executed to simulate and test various properties of the pipeline inspection robot. The different states are interconnected through input and output parameters defined in Figure1. Based on the input and output information above, the corresponding FSM diagram can be created (excluding related inputs and outputs here).


Inputs:
- onoff: [Off On] // on off sensor
- strt_lcn: [4b_x, 4b_y] // start location
- mtn_sensor: [Fzn Hot Jamm Free] //maintenance sensor
- cmps: [N E S W] //compass
- wll: [L R F] // wall locations

Outputs:
- turn: [TL TR] //turn left, turn right
- driving: [D] //drive
- location: [4b_x, 4b_y] //current location
- action: [Fire,Cool,Plunge] //action

Internal Registers:
- facing_dir: [N E S W] //facing direction after turning

Fig. 4. Finite State Machine (FSM) of the Robot.

III. SMIULATION in VERILOG by XILINX VIVADO

Based on the corresponding inputs and outputs a Verilog code can be written; the corresponding functions can be implemented as well in this case. This code handles the state transition logic and action execution of the pipeline maintenance robot, ensuring accurate control of the robot's behavior in response to external inputs.

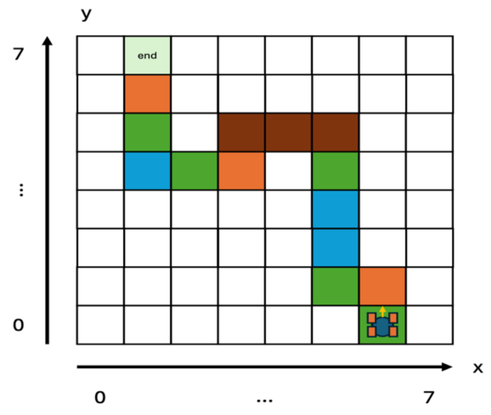


Fig. 5. Simulation Graph 1

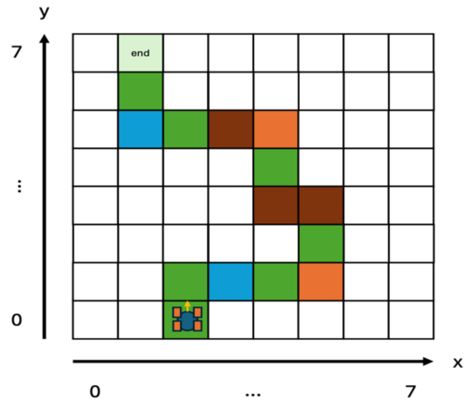


Fig. 6. Simulation Graph 2

<

1. *Testbench Code*

Simulation 1:

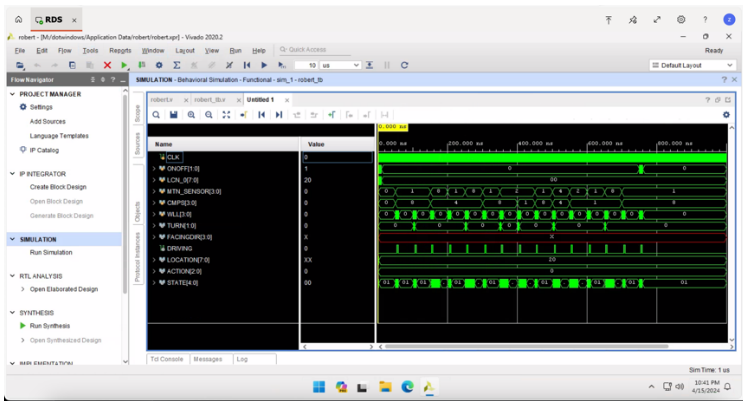
```
module robert_tb;
    reg CLK; //clock
    reg [1:0] ONOFF; // on/off
    reg [7:0] LCN_0; //starting location
    reg [3:0] MTN_SENSOR; //maintenance sensor
    reg [3:0] CMPS; //compass
    reg [3:0] WLL; //wall locations

    wire [1:0] TURN;
    wire [3:0] FACINGDIR;
    wire DRIVING;
    wire [7:0] LOCATION;
    wire [2:0] ACTION;
    wire [4:0] STATE;

    // Clock generation
    initial begin
        CLK = 0;
        forever #2 CLK = ~CLK;
    end

    robert
inst0(CLK,ONOFF,LCN_0,MTN_SENSOR,CMPS,WLL,TURN,FACINGDIR,DRIVING,LOCATION,AC
TION,STATE);

  initial begin
    ONOFF = 2'b01; LCN_0 = 8'b00100000; MTN_SENSOR = 4'b0000; CMPS =
4'b0000; WLL = 3'b000;
    #10 ONOFF = 2'b00;
    #40 ONOFF = 2'b01; LCN_0 = 8'b00001000;
    #40 MTN_SENSOR = 4'b1000; CMPS = 4'b0001; WLL = 3'b000;
    #40 MTN_SENSOR = 4'b0000; CMPS = 4'b0010; WLL = 3'b001;
    #40 MTN_SENSOR = 4'b0100; CMPS = 4'b0100; WLL = 3'b010;
    #40 MTN_SENSOR = 4'b0000; CMPS = 4'b1000; WLL = 3'b100;
    #40 MTN_SENSOR = 4'b0010; CMPS = 4'b0001; WLL = 3'b101;
    #40 MTN_SENSOR = 4'b0001; CMPS = 4'b0100; WLL = 3'b001;
    #40 ONOFF = 2'b10; MTN_SENSOR = 4'b0000; CMPS = 4'b0000; WLL = 3'b000;
    #10 ONOFF = 2'b00;
end

endmodule
```

Simulation 2:

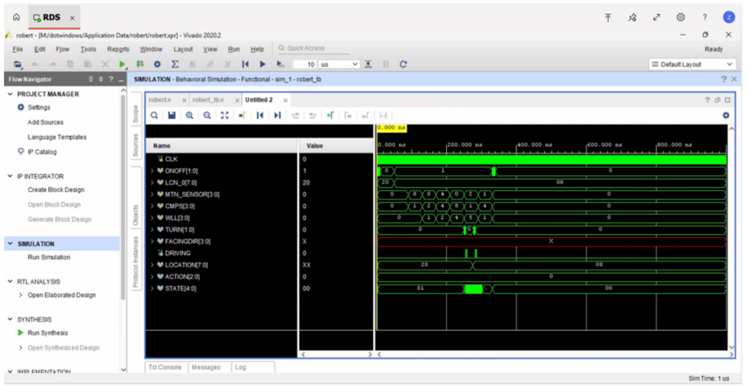
```
module xx_debug;
    reg CLK; //clock
    reg [1:0] ONOFF; // on/off
    reg [7:0] LCN_0; //starting location
    reg [3:0] MTN_SENSOR; //maintenance sensor
    reg [3:0] CMPS; //compass
    reg [3:0] WLL; //wall locations
    wire [1:0] TURN;
    wire [3:0] FACINGDIR;
    wire DRIVING;
    wire [7:0] LOCATION;
    wire [2:0] ACTION;
    wire [4:0] STATE;
    initial begin
        CLK = 0;
        forever #2 CLK = ~CLK;
    end
    pipeFSM
inst0(CLK,ONOFF,LCN_0,MTN_SENSOR,CMPS,WLL,TURN,FACINGDIR,DRIVING,LOCATION,ACTIO
N,STATE);
    initial begin
    ONOFF=2'b01; LCN_0=8'b00100000; MTN_SENSOR=4'b0000; CMPS=4'b0000;
WLL=3'b000;
    #10 ONOFF=2'b00; LCN_0=8'b00000000; MTN_SENSOR=4'b0000; CMPS=4'b0000;
WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b101;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b1000; CMPS=4'b0100; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b0100; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b1000; CMPS=4'b0100; WLL=3'b011;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0010; CMPS=4'b1000; WLL=3'b011;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0010; CMPS=4'b0001; WLL=3'b101;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0100; CMPS=4'b1000; WLL=3'b011;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0010; CMPS=4'b0001; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b0001; WLL=3'b110;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b1000; CMPS=4'b0001; WLL=3'b101;
    #10 WLL=3'b000;
    #40 MTN_SENSOR=4'b0001; CMPS=4'b1000; WLL=3'b110;
    #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;
    end
endmodule
```

IV. WAVEFORM and ANALYSIS

A. Waveform 1



B. Waveform 2



C. Analysis based on the waveform

Waveform plots from two different simulation results provide representative models for this report. These waveforms, generated by executing Verilog code in the Xilinx Vivado environment and running the corresponding testbench, visually display the simulation results. Additionally, they illustrate the dynamic data changes, reflecting the system's response and performance under various test conditions. These detailed waveform diagrams allow readers to better understand and evaluate the design's effectiveness and potential applications.

For the simulation, two groups of tests were conducted. The first group simulated the path shown in Fig. 5, with the testbench code configured to follow a predefined journey. Similarly, the second group simulated the path in Fig. 6, with the testbench code adjusted accordingly. Additional details about the testbench code for the pipe FSM, as well as the design and implementation, can be found in the repository.

<

V. Conclusion

As people demand more ways to make life easier, new engineering challenges keep increasing. Non-destructive evaluation (NDE) technology is becoming more important, and the need to improve it is growing. The pipe maintenance robot discussed in this article is a great example of using NDE technology in real-life situations. The robot uses advanced testing methods, along with Verilog programming, to perform its tasks. This design helps improve the inspection and maintenance of urban pipelines using NDE methods. In general, building a pipeline robot starts with designing a clear plan, called a finite state machine (FSM), which guides its actions. Code is then written and tested to make the robot work as planned. While working, the robot detects its direction, checks pipeline conditions like unusual temperatures (too hot or too cold), and gives feedback based on what it finds. It also decides its next move by looking at the positions of objects to its left, right, and front. Right now, when it checks for temperature problems or directions, the robot only uses simple indicators like 0 or 1 to show its status. This basic system is not enough for more complicated real-world tasks. In the future, as technology improves and the robot's settings are optimized, these issues are expected to be solved, making the robot more useful for practical applications.

References

? https://www.nde-ed.org/NDEEngineering/index.xhtml

? https://digitalsystemdesign.in/fsm-design/

? https://blog.csdn.net/weixin_42470069/article/details/107602611

? https://blog.csdn.net/chuoji2469384644/article/details/119675723

? https://www.researchgate.net/publication/317393901_A_New_Sensitive_Excitation_Technique_in_Nondestructive_Inspection_for_Underground_Pipelines_by_Using_Differential_Coils

Chenxin, Zhang A sophomore level student in College of Engineering at Michigan State University who is pursuing a B.S Degree in Electrical Engineering and Computer Science. He is currently an Undergraduate Research assistant under Navid Yazdi, Undergraduate Teaching assistant under Manni Liu.