***LICA PROJECT REPORT ON***

**Laser Security System Using 555 Timer**



SUBMITTED FOR PROJECT OF LICA LABORATORY AS PARTIAL FULFILLMENT OF B. TECH COURSE

By:

1. Sai Harshita Valluru -22ECB0F03
2. Kavya Kudumala - 22ECB0F04.

## Under the guidance of

Prof. Prithvi

Department of Electronics and Communication Engineering

NIT Warangal April 2024

**ABSTRACT**

Obstacle detection and avoidance during navigation of an autonomous vehicle is one of the challenging problems. Different sensors like RGB camera, Radar, and LiDAR are presently used to analyze the environment around the vehicle for obstacle detection. Analyzing the environment using supervised learning techniques has proven to be an expensive process due to the training of different obstacles for different scenarios. There has been increased interest in applying Reinforcement Learning (RL) techniques to understand the environment and make decisions. In this project we use reinforcement learning to make an **autonomous vehicle** navigate from the starting point to the destination and find the speed of vehicle after which the output is provided on fpga board.

***WHAT IS REINFORCEMENT LEARNING?***

Reinforcement learning is a type of machine learning technique that trains algorithms to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. It does so through trial-and-error using feedback from its own actions and experiences.

1. **Initialization**: The agent starts with little to no knowledge about the environment. It initializes its policy, typically randomly or based on some predefined rules.
2. **Interaction**: The agent interacts with the environment by taking actions based on its current policy. These actions influence the state of the environment, and the environment provides feedback in the form of rewards.
3. **Learning**: The agent learns from its interactions with the environment. It updates its policy based on the rewards received and its observations of the environment. This is often done using various learning algorithms such as Q-learning, SARSA, or deep reinforcement learning techniques like Deep Q-Networks (DQN) or Policy Gradient methods.
4. **Exploration vs. Exploitation**: A fundamental trade-off in reinforcement learning is between exploration (trying out new actions to discover better ones) and exploitation (taking actions that are known to yield high rewards). Balancing exploration and exploitation is crucial for efficient learning.
5. **Evaluation and Improvement**: The agent continually evaluates its policy and seeks to improve it over time. This might involve experimenting with different learning algorithms,etc.

**BRIEF IDEA:**

### Here is a brief description of the Reinforcement Learning that can be implemented on FPGA and it’s application in the mobility of autonomous vehicle-:

**The core elements of reinforcement learning are:**

1)**An Agent**: The learner and decision-maker.

2)**An Environment:** The world/problem the agent interacts with.

3)**States**: The current situation the agent finds itself in.

4)**Actions**: The choices made by the agent that cause a transition to a different state.

6)**Rewards:** The feedback signals that allow the agent to learn which actions lead to the goal.

The agent receives rewards by performing actions and learns to optimize its behavior to maximize the total reward over time. The rewards essentially guide the learning process towards the ideal behavior to achieve the specified goals.

**Some key characteristics**:

1)There is no supervisor, only a reward signal.

2)Feedback is delayed, not instantaneous.

3)Time really matters (current actions influence future situations).

Reinforcement learning is used in various applications like game playing, robotics, resource management, industrial control and many more. Some popular algorithms include Q-learning, SARSA, Policy Gradients and Actor-Critic methods.

**REINFORCEMENT LEARNING ON FPGA: -**

Implementing reinforcement learning algorithms on field-programmable gate arrays (FPGAs) is an area of active research and development. FPGAs offer advantages such as parallelism, low power consumption, and reconfigurability, which can be beneficial for certain reinforcement learning applications. Here are some key points about reinforcement learning on FPGAs:

1)**Hardware Acceleration**: FPGAs can be used to accelerate the computationally intensive parts of reinforcement learning algorithms, such as neural networks for function approximation, by implementing them in hardware. This can lead to significant performance improvements compared to software implementations on CPUs or GPUs.

2)**Customization**: FPGAs allow for customized hardware implementations of reinforcement learning algorithms, which can be tailored to the specific requirements of the application, potentially leading to better performance or resource utilization.

3)**Hardware-Software Co-design**: FPGAs can be used in conjunction with CPUs or GPUs in a hardware-software co-design approach, where the FPGA handles the computationally intensive parts of the algorithm, while the CPU or GPU manages the control flow and decision-making.

***TABLE OF CONTENTS***

* Introduction
* Theory and Usage of 555 Timer
* Major components of our Project
* Project Description
* Key features of our project
* Procedure
* Motivation
* Result
* Conclusion
* Future Work
* References

# INTRODUCTION

## THE SOFTWARE USED:

XILINX VIVADO (version 2014) as HDL- implementation tool

## Introduction to Verilog:

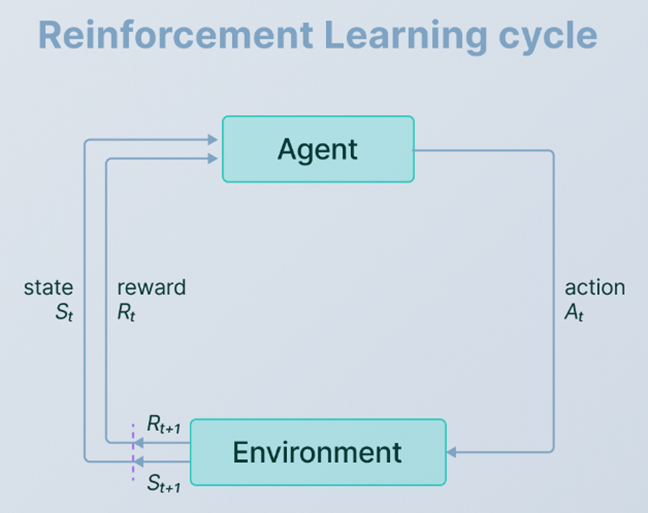
Verilog is a hardware description language (HDL) used to represent electrical systems. At the register-transfer level of abstraction, it is most widely employed in the design and verification of digital circuits. Today, Verilog is the most popular HDL used and practiced throughout the semiconductor industry. HDL designs are technology-independent, highly straightforward to create and debug, and are typically more helpful than schematics, especially for complex circuits. HDL was created to help engineers improve the design process by allowing them to define the intended hardware's functionality and then have automation tools turn that behavior into actual hardware pieces such as combinational gates and sequential logic.

# THEORY

As mentioned earlier in this report we are going to discuss about motion of autonomous vehicle using Reinforcement Learning. Here is a brief theory explaining about its functionality and process.

Obstacle detection and avoidance during navigation of an autonomous vehicle is one of the challenging problems. Different sensors like RGB camera, Radar, and LiDAR are presently used to analyze the environment around the vehicle for obstacle detection. Analyzing the environment using supervised learning techniques has proven to be an expensive process due to the training of different obstacles for different scenarios. There has been increased interest in applying Reinforcement Learning (RL) techniques to understand the environment and make decisions.

This reinforcement learning policy will find the next best action, given a current state. It chooses this action at random and aims to maximize the reward.



This project will implement a Reinforcement Learning algorithm on the Nexys 4 FPGA development board. The implementation is aimed to accomplish the following.

1)An Agent (an autonomous vehicle) will try to find the best path to take from point A to point B.

2)The Environment (terrain) will contain the path, obstacles along the path and areas which are not a path.

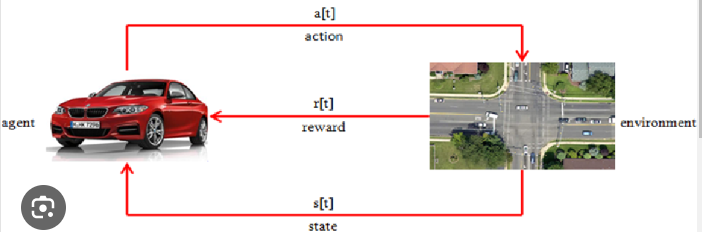
The Environment (valid and invalid paths, obstacles) can be altered by setting the slide switches.

3)The Agent will learn to go from point A to B by navigating the Environment.

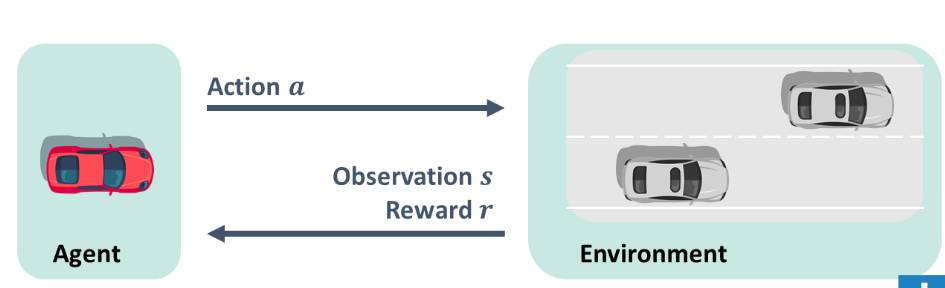
4)The reward system will help the Agent determine if the decisions made by the agent are good or bad.

Through reinforcement learning, Agent will try to find the optimal speed and distance travelled to identify the best path to reach the destination. The best path is the path with the shortest distance AND time. The Agent will be rewarded if the speed is high, but will receive a penalty if the Agent runs into an obstacle.

A visual indication of the navigation path and rewards will be provided via the 7-segment display and the LEDs.



Reinforcement learning concept diagram



#### PROBLEM STATEMENT: -

#### Implementation of Reinforcement learning on autonomous vehicle for motion planning. The vehicle navigates on a 4-lane road and is and is assigned a starting and ending point. A set of obstacles are introduced in its path. It is allowed to choose an action at random (left, right, straight, back). It is rewarded based on its action: -

#### +5 if it moves closer to its destination, +15 if it reaches its destination, -7 if it goes outside its boundary, -10 if it collides with an obstacle. Speed is calculated once it reaches its destination. The path with maximum reward is the optimal path and speed.

#### Key Features of our project:

The key features of our project are as follows:

1. **Linear Feedback Shift Register (LFSR):**
   * Our project includes an LFSR module that generates a pseudo-random sequence of values.
   * The LFSR module has configurable parameters for the width (WIDTH) and the polynomial (POLYNOMIAL).
   * The LFSR value is updated on each positive edge of the clock using the specified polynomial.
2. **Grid-based Environment:**
   * The environment is represented as a 4x10 grid, with the agent (car) moving within this grid.
   * The start position (start\_x, start\_y) and goal position (end\_x, end\_y) are defined.
   * Certain positions on the grid are marked as obstacles, which the agent should avoid.
3. **Agent Actions:**
   * The agent can take four different actions: move left, move right, move straight, or move back.
   * The actions are selected randomly using the pseudo-random values generated by the LFSR.
4. **Reward System:**
   * The agent receives rewards or penalties based on its actions and position in the environment.
   * Reaching the goal position (end\_x, end\_y) yields a reward of +15.
   * Moving without hitting an obstacle yields a reward of +5.
   * Trying to move out of bounds or into an obstacle yields penalties of -7 or -10, respectively.
5. **Environment Interaction:**
   * The project implements the logic for updating the agent's position (current\_x, current\_y) based on the selected action and the current position.
   * The code checks for valid moves and obstacle collisions before updating the agent's position.
6. **Reset Functionality:**
   * The project includes a reset signal (rst) that resets the agent's position to the start position and clears the reward signal.
7. **Clocked Design:**
   * The project is designed to operate on a clock (clk) signal, with state updates occurring on the positive edge of the clock.

**Functional description:**

Our Project implements a reinforcement learning environment for a car-like agent navigating through a grid-based world with obstacles. The environment is represented by a 4x10 grid, with predefined start and goal positions, as well as certain positions marked as obstacles. The agent can take four different actions: move left, right, straight, or back. A Linear Feedback Shift Register (LFSR) module is used to generate pseudo-random values, which determine the agent's actions randomly. The agent's position is updated based on the selected action, and rewards or penalties are assigned based on the consequences of the action. Reaching the goal position yields a positive reward, while moving without hitting an obstacle yields a smaller positive reward. Attempting to move out of bounds or into an obstacle result in negative penalties. Our Verilog code simulates the interaction between the agent and the environment, providing a reward signal based on the agent's actions. It also calculates the final speed of vehicle along with the suitable path to reach the destination.

## FUNCTIONS DONE IN OUR PROJECT:

Given below is a brief functional description of the project: -

**LFSR Module**: This module is responsible for generating a pseudo-random sequence of values using a Linear Feedback Shift Register (LFSR). The LFSR is initialized with all ones when the rst signal is asserted. On each positive edge of the clk signal, the LFSR value is updated based on the specified polynomial (POLYNOMIAL). The current value of the LFSR is assigned to the output register q.

**Car Module**: This module implements the core functionality of the car-like agent and its environment.

1. **Initialization**:
   * When the rst signal is asserted, the module initializes the current\_x and current\_y registers to the start position (start\_x, start\_y), and the reward register is set to 0.
2. **Goal Condition**:
   * The module checks if the car's current position (current\_x, current\_y) matches the goal position (end\_x, end\_y). If it does, the module sets the reward register to 15 and stops further execution.
3. **Action Selection**:
   * The module selects the next action for the car based on the least significant two bits of the random value generated by the LFSR instance.
   * The action register is assigned the selected action (left, right, straight, or back).
4. **Position Update**:
   * Based on the selected action and the car's current position, the module updates the current\_x and current\_y registers accordingly.
   * The module checks for valid moves and obstacle collisions before updating the position.
5. **Reward Calculation**:
   * The module calculates the reward or penalty based on the car's current position and the selected action.
   * If the car moves without hitting an obstacle, it receives a reward of +5.
   * If the car tries to move out of bounds or into an obstacle, it receives a penalty of -7 or -10, respectively.
   * The reward register is updated with the calculated reward or penalty.
6. **Clock and Reset**:
   * The module operates on the positive edge of the clk signal.
   * When the rst signal is asserted, the module resets its state and initializes the car's position and reward.

7.**FPGA Implementation:**

* So, on the fpga board the car obstacles are given, based on the coordinates the module will tell us the best path from starting point to ending point.

****

## MAJOR COMPONENTS OF OUR PROJECT:

## Linear Feedback Shift Register (LFSR) Module

## Module instantiation

## Width parameter (WIDTH)

## Polynomial parameter (POLYNOMIAL)

## Input ports: clk, rst

## Output port: q

## Car Module

## Module instantiation

## Input ports: clk, rst

## Output ports: current\_x, current\_y, action, reward

## Internal signals and registers

## Instantiation of the LFSR module

## Parameters for start and end positions (start\_x, start\_y, end\_x, end\_y)

## Parameters for actions (left, right, straight, back)

## Case statement for handling actions

## Environment Representation

## Predefined start position (start\_x, start\_y)

## Predefined goal position (end\_x, end\_y)

## Positions marked as obstacles

## Action Selection

## Usage of the LFSR's pseudo-random value (random\_value)

## Position Update

## Registers for current\_x and current\_y

## Logic for updating current\_x and current\_y based on the selected action

## Reward Calculation

## Register for reward

## Logic for updating reward based on the car's position and action

## Reset Functionality

## Reset signal (rst)

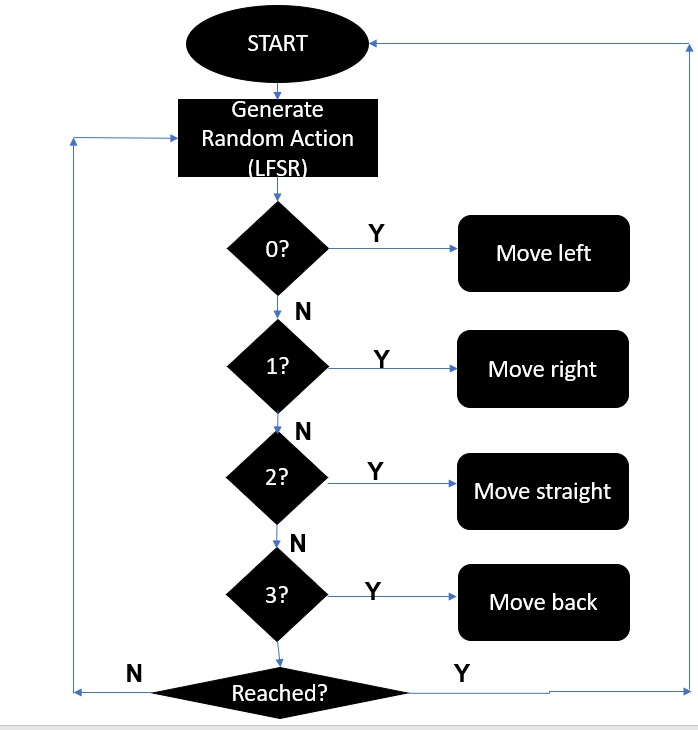
## Reset logic for current\_x, current\_y, and reward

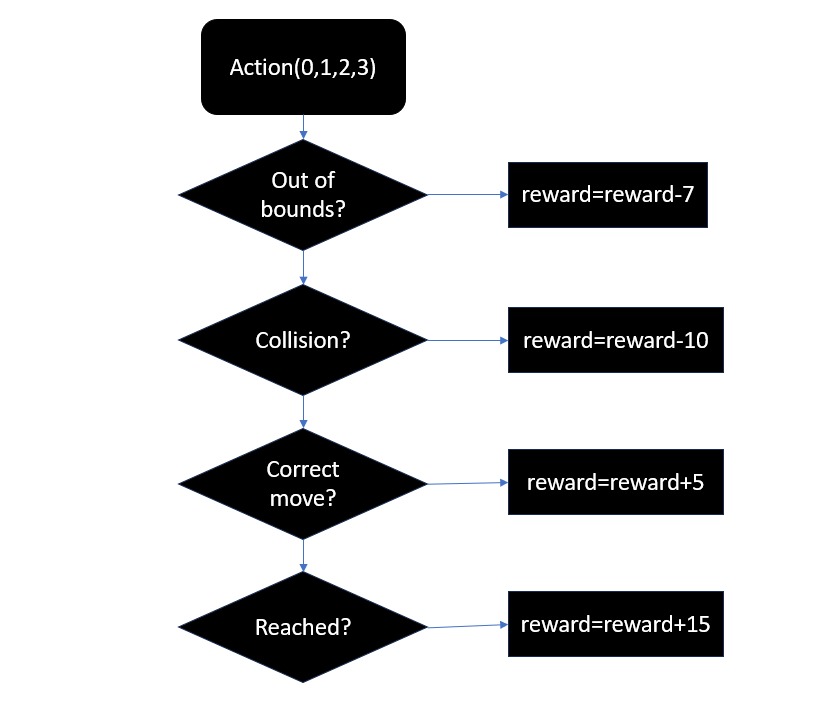
## Clocked Design

## Usage of the clock signal (clk)

## Sensitivity list for the always blocks

# WORKFLOW





**MOTIVATION**

### What made us choose this project?

We as a group felt there should be something helpful that we learn after performing this project. Choosing Navigation of Autonomous Vehicle Using Reinforcement Learning project controller as a FPGA mini project can be motivated by several factors, depending on the goals and learning objectives of the project.

1. **Exploration of Reinforcement Learning Concepts**: Our project represents a basic reinforcement learning setup, where an agent (the car) interacts with an environment (the grid) by taking actions and receiving rewards or penalties based on the consequences of those actions. This allows for the exploration and demonstration of fundamental reinforcement learning concepts, such as the agent-environment interaction, reward signals, and the trial-and-error learning process.
2. **Hardware Implementation of a Learning Environment**: Our project is written in Verilog, a hardware description language, suggesting that the motivation might be to implement the reinforcement learning environment in hardware, potentially using Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). This could be useful for applications that require real-time learning, low power consumption, or parallel processing capabilities.
3. **Simplicity and Educational Purposes**: The environment and the agent's behaviour are relatively simple, with a small grid size, predefined obstacles, and random action selection using an LFSR. This simplicity could serve as a starting point for understanding and teaching reinforcement learning concepts, allowing students or researchers to grasp the basics before moving on to more complex algorithms and environments.
4. **Foundation for Further Development**: It establishes a framework for the agent-environment interaction and reward calculation. This foundation could be extended by integrating actual reinforcement learning algorithms, such as Q-learning, SARSA, or Deep Q-Networks (DQN), to enable the agent to learn an optimal policy through trial-and-error interactions with the environment.
5. **Hardware-Software Co-design**: The hardware implementation of the reinforcement learning environment could potentially be integrated with software components running on a CPU or GPU, enabling a hardware-software co-design approach. This could leverage the strengths of both hardware and software components for efficient and powerful reinforcement learning systems.

# VERILOG HDL CODE

DESIGN SOURCE CODE: -

//with clk code

`timescale 1ns / 1ps

module lfsr #(

parameter WIDTH = 16,

parameter POLYNOMIAL = 16'h8005

)(

input clk,

input rst,

output reg [WIDTH-1:0] q

);

reg [WIDTH-1:0] lfsr;

always @(posedge clk or posedge rst) begin

if (rst) begin

lfsr <= {WIDTH{1'b1}};

end else begin

lfsr <= {lfsr[WIDTH-2:0], lfsr[WIDTH-1] ^ lfsr[WIDTH-3] ^ lfsr[WIDTH-5] ^ lfsr[WIDTH-16]};

end

end

always @(\*) begin

q = lfsr;

end

endmodule

//

module car(

input clk,

input rst,

output reg [0:6] seg,

output reg [7:0] DIGIT

);

integer k=0;

reg [3:0] current\_x;

reg [3:0] current\_y;

reg [1:0] action;

reg signed[15:0] reward;

reg [2:0] digit\_select; // 2 bit counter for selecting each of 4 digits

reg [16:0] digit\_timer; // counter for digit refresh

// Parameters for segment patterns

parameter ZERO = 7'b000\_0001; // 0

parameter ONE = 7'b100\_1111; // 1

parameter TWO = 7'b001\_0010; // 2

parameter THREE = 7'b000\_0110; // 3

parameter FOUR = 7'b100\_1100; // 4

parameter FIVE = 7'b010\_0100; // 5

parameter SIX = 7'b010\_0000; // 6

parameter SEVEN = 7'b000\_1111; // 7

parameter EIGHT = 7'b000\_0000; // 8

parameter NINE = 7'b000\_0100; // 9

reg [3:0] start\_x = 4'b0000;

reg [3:0] start\_y = 4'b0001;

reg [3:0] end\_x = 4'b0011;

reg [3:0] end\_y = 4'b1001;

parameter left = 2'b00;

parameter right = 2'b01;

parameter straight = 2'b10;

parameter back = 2'b11;

wire [15:0] random\_value;

reg [26:0] count;

reg seconds;

//

//

always @(posedge clk or posedge rst)

begin

if(rst) begin

count <= 0;

seconds <= 0;

end else if (count == 27'd50\_000\_000) begin

count <= 0;

seconds <= ~seconds;

end else begin

count <= count + 1'b1;

end

end

//

//

lfsr #(

.WIDTH(16),

.POLYNOMIAL(16'h8005)

) lfsr\_inst (

.clk(seconds),

.rst(rst),

.q(random\_value)

);

//

always @(posedge seconds or posedge rst)

begin

if (rst) begin

reward=+16'sb000000000000000;

current\_x <= start\_x;

current\_y <= start\_y;

k = 0;

end

else if (current\_x == end\_x && current\_y == end\_y) begin

k = 1;

reward=reward+15;

$finish;

end

else begin

action = random\_value[1:0];

case (action)

left: if (current\_x > 0 && !(current\_x - 1 == 0 && current\_y == 4) && !(current\_x - 1 == 1 && current\_y == 2) && !(current\_x - 1 == 2 && current\_y == 6) && !(current\_x - 1 == 3 && current\_y == 5))

begin

current\_x = current\_x - 1'b1;

k = 0;

reward=reward+5; //reward= +5

end

else if (current\_x<=0)

reward=reward-7; //reward= -7

else if ((current\_x - 1 == 0 && current\_y == 4) ||(current\_x - 1 == 1 && current\_y == 2) || (current\_x - 1 == 2 && current\_y == 6) || (current\_x - 1 == 3 && current\_y == 5))

reward=reward-10; //reward=-10

right: if (current\_x < 3 && !(current\_x + 1 == 0 && current\_y == 4) && !(current\_x + 1 == 1 && current\_y == 2) && !(current\_x + 1 == 2 && current\_y == 6) && !(current\_x + 1 == 3 && current\_y == 5)) begin

current\_x = current\_x + 1'b1;

k = 0;

reward=reward+5; //reward= +5

end

else if (current\_x>=3)

reward=reward-7; //reward= -7

else if((current\_x + 1 == 0 && current\_y == 4) || (current\_x + 1 == 1 && current\_y == 2) || (current\_x + 1 == 2 && current\_y == 6) || !(current\_x + 1 == 3 && current\_y == 5))

reward=reward-10; //reward=-10

straight: if (current\_y < 9 && !(current\_x == 0 && current\_y + 1 == 4) && !(current\_x == 1 && current\_y + 1 == 2) && !(current\_x == 2 && current\_y + 1 == 6) && !(current\_x == 3 && current\_y + 1 == 5)) begin

current\_y = current\_y + 1'b1;

k = 0;

reward=reward+5; //reward= +5

end

else if (current\_y>=9)

reward=reward-7; //reward= -7

else if((current\_x == 0 || current\_y + 1 == 4) || (current\_x == 1 && current\_y + 1 == 2) || (current\_x == 2 && current\_y + 1 == 6) || (current\_x == 3 && current\_y + 1 == 5))

reward=reward-10; //reward=-10

back: if (current\_y>0 && !(current\_x == 0 && current\_y-1 == 4) && !(current\_x== 1 && current\_y-1 == 2) && !(current\_x == 2 && current\_y-1 == 6) && !(current\_x == 3 && current\_y-1 == 5))

begin

current\_y = current\_y - 1'b1;

k = 0;

reward=reward+5; //reward= +5

end

else if (current\_x<=3)

reward=reward-7; //reward= -7

else if((current\_x == 0 && current\_y-1 == 4) || (current\_x== 1 && current\_y-1 == 2) || (current\_x == 2 && current\_y-1 == 6) ||(current\_x == 3 && current\_y-1 == 5))

reward=reward-10; //reward=-10

endcase

end

end

//

//

always @(posedge clk or posedge rst) begin

if(rst) begin

digit\_select <= 0;

digit\_timer <= 0;

end

else // 1ms x 4 displays = 4ms refresh period

if(digit\_timer == 99\_999) begin // The period of 100MHz clock is 10ns (1/100,000,000 seconds)

digit\_timer <= 0; // 10ns x 100,000 = 1ms

digit\_select <= digit\_select + 1;

end

else

digit\_timer <= digit\_timer + 1;

end

always @(digit\_select) begin

case(digit\_select)

3'b000 : begin

DIGIT = 8'b11111110;

end

3'b001 : begin

DIGIT = 8'b11111101;

end

3'b010 : begin

DIGIT = 8'b11111011;

end

3'b011 : begin

//DIGIT = 8'b11110111;

DIGIT = 8'b11111111;

end

3'b100 : begin

//DIGIT = 8'b11101111;

DIGIT = 8'b11111111;

end

3'b101 : begin

//DIGIT = 8'b11011111;

DIGIT = 8'b11111111;

end

3'b110 : begin

//DIGIT = 8'b10111111;

DIGIT = 8'b11111111;

end

3'b111 : begin

//DIGIT = 8'b01111111;

DIGIT = 8'b11111111;

end

endcase

end

always @\*

case(digit\_select)

3'b000 : begin

case(current\_y)

4'b0000: seg = ZERO;

4'b0001: seg = ONE;

4'b0010: seg = TWO;

4'b0011: seg = THREE;

4'b0100: seg = FOUR;

4'b0101: seg = FIVE;

4'b0110: seg = SIX;

4'b0111: seg = SEVEN;

4'b1000: seg = EIGHT;

4'b1001: seg = NINE;

endcase

end

3'b001 : begin

case(current\_x)

4'b0000: seg = ZERO;

4'b0001: seg = ONE;

4'b0010: seg = TWO;

4'b0011: seg = THREE;

4'b0100: seg = FOUR;

4'b0101: seg = FIVE;

4'b0110: seg = SIX;

4'b0111: seg = SEVEN;

4'b1000: seg = EIGHT;

4'b1001: seg = NINE;

endcase

end

3'b010 : begin

case(action)

4'b0000: seg = ZERO;

4'b0001: seg = ONE;

4'b0010: seg = TWO;

4'b0011: seg = THREE;

4'b0100: seg = FOUR;

4'b0101: seg = FIVE;

4'b0110: seg = SIX;

4'b0111: seg = SEVEN;

4'b1000: seg = EIGHT;

4'b1001: seg = NINE;

endcase

end

3'b011 : seg = 7'b000\_0001;

3'b100 : seg = 7'b000\_0001;

3'b101 : seg = 7'b000\_0001;

3'b110 : seg = 7'b000\_0001;

3'b111 : seg = 7'b000\_0001;

endcase

//

//

endmodule

# RESULT

# 

# 

**CONCLUSION**

In conclusion, the project successfully implemented a simple reinforcement learning environment for a car-like agent navigating through a grid-based world with obstacles. The Verilog code effectively modeled the environment, agent actions, and reward system, providing a foundation for exploring and understanding the fundamental concepts of reinforcement learning.

The reward calculation mechanism played a crucial role, providing positive rewards for reaching the goal position and moving without hitting obstacles, as well as negative penalties for attempting to move out of bounds or into obstacles. This reward signal served as the feedback mechanism for the agent to learn from its actions and experiences.

Overall, this project successfully demonstrated the potential of hardware implementations for reinforcement learning environments and provided a solid starting point for further research and development in this field. With the integration of advanced learning algorithms and potential hardware-software co-design, this project could contribute to the advancement of reinforcement learning techniques and their applications in various domains, such as robotics.

# FUTURE WORK

Our future work related to this project can potentially be:-

1. **Integration of Reinforcement Learning Algorithms**: Future work could involve integrating popular reinforcement learning algorithms like Q-learning, SARSA, or Deep Q-Networks (DQN) to enable the agent (car) to learn an optimal policy through trial-and-error interactions with the environment.
2. **Exploration vs. Exploitation Strategy**: Incorporating an exploration-exploitation strategy would allow the agent to balance between exploring new actions and exploiting the learned knowledge. This could involve techniques like epsilon-greedy or SoftMax action selection strategies, which would enhance the agent's learning capabilities.
3. **Function Approximation and Deep Reinforcement Learning:** Future work could explore the use of function approximation techniques, such as neural networks or other machine learning models, to generalize and represent the value function or policy in more complex environments. This could pave the way for implementing deep reinforcement learning algorithms like Deep Q-Networks (DQN) or Deep Deterministic Policy Gradients (DDPG).

# REFERENCES

1) https://github.com

2) <https://www.geeksforgeeks.org>

3) https://www.tutorialspoint.com

# 