VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

LSI DESIGN CONTEST IN OKINAWA 2021

DESIGN HARWARE ACCELERATOR FOR DYNA-Q ALGORITHM TO SOLVE 5x5 MAZE PROBLEM

INSTRUCTOR: DUONG Tran (duongtd@uit.edu.vn)

TEAM: UIT#EAD2021

MEMBER:

1. THIEN Le
2. HIEU Nguyen
3. PHUC Nguyen

VIETNAM, JAN 2021

INFORMATION PAGE

1. **Team Name:** UIT#EAD2021
2. **School Name:** University of Information Technology – HCM_VNU
3. **Member Information:**

No.	Name	Phone	Email	T-Shirt	Address	Grade
1	THIEN Le	0978011439	1751075@gm.uit.edu.vn	M	University of Information Technology – VNUHCM	4
2	HIEU Nguyen	0971137579	18520750@gm.uit.edu.vn	S		3
3	PHUC Nguyen	0964630920	17520909@gm.uit.edu.vn	M		4

4. **The task level of the design:** Intermediate

PREFACE

Over the last few year, reinforcement learning has achieved remarkable successes in game playing tasks like Atari or chess, control tasks in robotics or logistic problems. More and more research has been carried out to study the potential and applicability of end-to-end Reinforcement Learning. One of the most exciting problem is how to efficiently simulate real-world environment and develop reinforcement learning algorithms in fast computing hardwares. Through LSI design contest, we hope to make one small step closer to that goal by introducing accelerator design of Dyna Q Learning algorithm and uses it to solve a Grid World problem. Our RL agent is exposed to the Maze environment and only receives information of the cell it stands (no look ahead). With little environment information, our agent is able to find the optimal path to treasure in relatively small amount of training steps.

LIST OF CONTENT

Chapter 1. Overview 1
1.1. Introduction 1
1.2. Maze Problem 2
1.3. Dyna Q Learning 3
1.4. Why we chose Dyna-Q 4
Chapter 2. System Design 6
2.1. Environment block 7
2.2. Q-Table block 9
2.3. Dyna-Q Training block 10
Chapter 3. Dyna-Q Design 13
3.1. Algorithm State Machine (ASM) 13
3.2. Datapath 17
3.3. Controller 20
Chapter 4. Result Evolution 23
4.1. Verification Flow 23
4.2. Simulation 25
4.3. Synthesis & Critical path speed 26
4.4. Performance of Dyna-Q vs Q-Learning 27
4.5. Implementation and Testing 29
4.5.1. Demo maze from LSI Contest 2021 32
4.5.2. Random maze 43
4.5.2.1. Maze 1 43
4.5.2.2. Maze 2 46

4.5.2.3. Maze 3 50
4.5.3. A lot of tests 52
Chapter 5. Conclusion 54
5.1. Appeal points and originality 54
5.2. Limitation 55
5.3. Recommendation 55

LIST OF FIGURES

Figure 1.1: Reinforcement Learning mechanism.....1
Figure 2.1: Dyna-Q's algorithm system deign.....6
Figure 2.2: The inside of Environment block 7
Figure 2.3: The inside of Q-table block 10
Figure 2.4: An ASIC's structure 11
Figure 3.1: The ASM of Dyna-Q Algorithm 16
Figure 3.2: The data-path of Dyna-Q's accelerator 17
Figure 4.1: The verification flow 23
Figure 4.2: The training report.....25
Figure 4.3: Pre-Synthesis Waveform 25
Figure 4.4: The critical path of Dyna-Q.....26
Figure 4.5: Implement and Testing flow.....30
Figure 4.6: KIT test 31
Figure 4.7: Maze from LSI Contest 2021- Intermediate.....32
Figure 4.8: The configuration on the DE2 KIT and result.....33
Figure 4.9: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 0) – Hardware version 33
Figure 4.10: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 0) – Software version 34
Figure 4.11: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 1) – Hardware version 34
Figure 4.12: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 1) – Software version 34
Figure 4.13: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 2) – Hardware version 35
Figure 4.14: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 2) – Software version 35

Figure 4.15: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 3) – Hardware version 35
Figure 4.16: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 3) – Software version 36
Figure 4.17: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 8) – Hardware version 36
Figure 4.18: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 8) – Software version 36
Figure 4.19: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 9) – Hardware version 37
Figure 4.20: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 9) – Software version 37
Figure 4.21: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 14) – Hardware version 37
Figure 4.22: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 14) – Software version 38
Figure 4.23: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 5) – Hardware version 38
Figure 4.24: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 5) – Software version 38
Figure 4.25: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 10) – Hardware version 39
Figure 4.26: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 10) – Software version 39
Figure 4.27: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 15) – Hardware version 39
Figure 4.28: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 15) – Software version 40

Figure 4.29: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 20) – Hardware version 40
Figure 4.30: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 20) – Software version 40
Figure 4.31: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 11) – Hardware version 41
Figure 4.32: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 11) – Software version 41
Figure 4.33: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 12) – Hardware version 41
Figure 4.34: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 12) – Software version 42
Figure 4.35: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 17) – Hardware version 42
Figure 4.36: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 17) – Software version 42
Figure 4.37: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 22) – Hardware version 43
Figure 4.38: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 22) – Software version 43
Figure 4.39: Random maze (1).....43
Figure 4.40: Optimized road of the random maze 1 (Start Location is 4) – Hardware version 44
Figure 4.41: Optimized road of the random maze 1 (Start Location is 4) – Software version 44
Figure 4.42: Optimized road of the random maze 1 (Start Location is 1) – Hardware version 44
Figure 4.43: Optimized road of the random maze 1 (Start Location is 1) – Software version 45

Figure 4.44: Optimized road of the random maze 1 (Start Location is 24) – Hardware version 45
Figure 4.45: Optimized road of the random maze 1 (Start Location is 24) – Software version 45
Figure 4.46: Random maze (2).....46
Figure 4.47: Optimized road of the random maze 2 (Start Location is 0) – Hardware version 46
Figure 4.48: Optimized road of the random maze 2 (Start Location is 0) –Software version 46
Figure 4.49: Optimized road of the random maze 2 (Start Location is 6) – Hardware version 47
Figure 4.50: Optimized road of the random maze 2 (Start Location is 6) –Software version 47
Figure 4.51: Optimized road of the random maze 2 (Start Location is 10) – Hardware version 47
Figure 4.52: Optimized road of the random maze 2 (Start Location is 10) –Software version 48
Figure 4.53: Optimized road of the random maze 2 (Start Location is 20) – Hardware version 48
Figure 4.54: Optimized road of the random maze 2 (Start Location is 20) –Software version 48
Figure 4.55: Optimized road of the random maze 2 (Start Location is 22) – Hardware version 49
Figure 4.56: Optimized road of the random maze 2 (Start Location is 22) –Software version 49
Figure 4.57: Random maze (3).....50
Figure 4.58: Optimized road of the random maze 3 (Start Location is 10) – Hardware version 50

Figure 4.59: Optimized road of the random maze 3 (Start Location is 10) – Software version 50
Figure 4.60: Optimized road of the random maze 3 (Start Location is 11) – Hardware version 51
Figure 4.61: Optimized road of the random maze 3 (Start Location is 11) – Software version 51
Figure 4.62: Optimized road of the random maze 3 (Start Location is 17) – Hardware version 51
Figure 4.63: Optimized road of the random maze 3 (Start Location is 17) – Software version 52
Figure 4.64: The comparation result between Hardware and Software version – Same cases 52
Figure 4.65: The comparation result between Hardware and Software version – Different cases.....53
Figure 5.1: Q-Learning vs. Deep Q Learning. 55

LIST OF TABLES

Table 1.1: Parameter comparation 5
Table 2.1: The purpose of each block in system design 6
Table 2.2: Describing signals of Environment block.....7
Table 2.3: Decode value in Environment block.....8
Table 2.4: Reward decode value 8
Table 2.5: Describing signals of Q-Table block 9
Table 2.6: Describing signal of Dyna-Q Training block 11
Table 3.1: Describing each state in ASM 13
Table 3.2: Describing the Data Flow for the data-path.....18
Table 3.3: Describing for controller.....21
Table 4.1: The description of each step in Verification flow 24
Table 4.2: Hardware utilization and Max Frequency of Dyna-Q and Q-learning accelerator 26
Table 4.3: The comparation of Dyna-Q and Q-Learning's result (1) 27
Table 4.4: The comparation of Dyna-Q and Q-Learning's result (2) 28
Table 4.5: The comparation between two epsilon condition. 29
Table 4.6: Describing each step in Implement and Testing Flow.....30

Chapter 1. Overview

1.1. Introduction

It seems reasonable that the first thing comes to mind when someone ask about method of learning is learning by interacting with our surroundings. For example, babies, with no pre-programmed rules embedded in them, can gradually learn complicated tasks such as talking and walking just from interacting with their parents. These interactions offer human sense of cause and effect, teach us consequences of our own action, and through those, we adjust our behaviours to favour the actions that generate good consequences. It is safe to say that learning from interaction is the basic foundation of theories about learning and artificial intelligence. Among all kinds of machine learning paradigms, Reinforcement Learning (RL) is the one that strives to model agent environment interactions.

Reinforcement Learning is a computational method of understanding and automating goal-directed learning and decision making. It differs from the other kinds of learning by its emphasis on learning through trial and error, which needs no exemplary supervision or complete models of the environment. A reinforcement learning framework typically involves agents and an enviroment, where agents can interact and learn to optimize an objective via carefully designed reward signals.

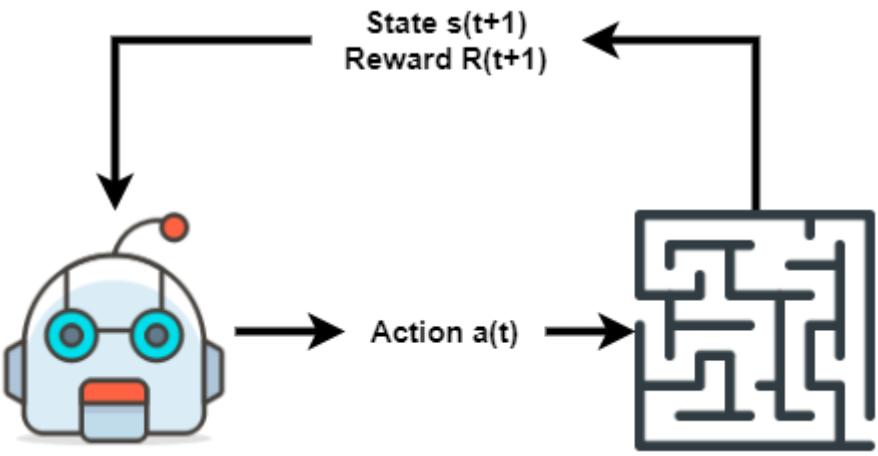


Figure 1.1: Reinforcement Learning mechanism

Markov Decision Process (MDP) is usually adopted to formalize these interactions between agents and environment, which can be described in terms of states, rewards and actions. Using MDP, practitioners of reinforcement learning hope to introduce sense of cause and effect, sense of uncertainty and nondeterminism, and the existence of explicit goals as essential features to an Artificial Intelligence (AI) problem. Additionally, the basic concepts of reinforcement learning often include two main components: Policy and value function. A policy, which maps from state to action, represents the strategy the agents take to solve the enviroments. The value function, which maps from state to a real number value estimating the goodness of that state in the long run, is responsible for the efficiency of searching the policy space, which can be prohibitedly large.

The very basics of reinforcement learning, such as Bellman equation, policy iteration, originates in the late 1950s, and through time, evolve and give birth to variety of successes in a wide range of fields: Alpha Zero, robotics, logistic, autonomous car.

However, Reinforcement Learning is yet a young study field and many interesting problems are open for future reasearch. Among these open problems, the study on methods to incorporate reinforcement learning from software simulation to large-scale real-world applications is excited above all. Through LSI design contest, we hope to make one small step closer to that goal.

1.2. Maze Problem

Environment: A 5x5 grid world filled with demons and a treasure.

Objective: Find the optimal path to goal with little information of the environment.

Rules:

+ If step into demon, turn back to the previous state.

+ The agent only has the information of the grid world size and its standing cell (no look ahead).

1.3. Dyna Q Learning

Dyna Q learning is among model-based planning reinforcement learning methods which learns Q function to estimate state value. Dyna Q learning is basically Q-Learning algorithm with extra planning steps.

Q Learning: The agent tries to learn a Q function (in the form of a table) to estimate the "quality" of each state action pair. In training phase, every time the agent takes an action, the environment returns next state and reward, which are used to update the agent's Q table following the equation 1.1.

$$Q(S, A)_{new} = Q(S, A) + \alpha*(R + \gamma*\max Q(S', A) - Q(S, A)) \qquad (1.1)$$

In the training phase, agent has to balance the trade off between exploration and exploitation using epsilon greedy update: every time the agent has to choose an action, there is probability epsilon that it chooses random action and probability 1 – epsilon that it follows the greedy Q-table (taking the action that returns that maximal value). The whole algorithm is summarised as following:

Step 1: Initialize Q table with random values.

Step 2: While playing the game execute the following loop.

a) Generate random number between 0 and 1 – if number is larger than the threshold epsilon select random action, otherwise select action with the highest possible reward based on state and Q-table.
b) Take action from step 2.a.
c) Observe reward r after taking action.
d) Update Q table based on the reward r using the update formula.

"Dyna" in Dyna Q Learning: This refers to a class of methods that keep track of the model of the environment during agent-environment interaction. Specifically,

the agent has to save an experience (in the form of state-action-next state-reward) to its memory each timestep. At the end of each timestep, the agent randomly samples a batch of these experiences and continue to update the Q table using Q learning fucntion. The whole algorithm is summarised as following:

Step 1: Initialize Q table with random values and empty replay buffer.

Step 2: While playing the game execute the following loop.

a) Generate random number between 0 and 1 – if number is larger than the threshold epsilon select random action, otherwise select action with the highest possible reward based on state and Q-table.
b) Take action from step 2.a.
c) Observe reward r after taking action.
d) Update Q table based on the reward r using the update formula.
e) Append tuple (State, Action, Next State, Action) to replay buffer.
f) Randomly sample n batches from replay memory.
g) For each batch, using the experience to update the Q table with update formula.

1.4. Why we chose Dyna-Q

Dyna Q learning has shown to be more effective in our maze problem, which means requiring less training step to find the optimal path than vanilla Q learning. To prove our hypothesis, we design an experiment of parameter search and compare Q-Learning with Dyna-Q.

We adopt the experiment pipeline as follows: First we choose parameter set to experiment with and use it to solve 5x5 maze problem. We keep track of length of best solution path and training episodes needed to solve the task. Due to the stochasticity of Reinforcement Learning, for each parameter set, we have to run 30 times and take average result for stability test.

Using the pipeline above, we compare Q-Learning and Dyna-Q with different values of epsilon (epsilon greedy policy parameter), gamma (discount factor) and alpha (leaning rate). Dyna Q have an additional parameter of the number of planning step (n-step Dyna-Q).

Table 1.1: Parameter comparation

(alpha, beta, epsilon)	**Best**	**Solution**	**Training**
(0.1, 0.9, 0.05)	10	10.02	26.0
(0.3, 0.9, 0.05)	10	10.0	19.28
(0.5, 0.9, 0.05)	10	10.0	18.43
(0.7, 0.9, 0.05)	10	10.0	18.04
(0.9, 0.9, 0.05)	10	10.0	18.11
(1.0, 0.9, 0.05)	10	10.0	17.91
(1.0, 0.5 0.05)	10	10.0	18.37
(1.0, 0.7, 0.05)	10	10.02	18.91
(1.0, 0.9, 0.05)	10	10.0	18.45
(1.0, 1.0, 0.05)	10	10.0	18.49

We observe the result and conclude the optimal parameters: alpha (1.0), gamma (0.5).

Chapter 2. System Design

In Chapter 1, we have already described the Dyna-Q algorithm and why we choose this algorithm instead of Q-Learning algorithm. This chapter will discuss about:

– Overview about the system.

– The purpose of each block in design and their input/output.

– The connection between these blocks.

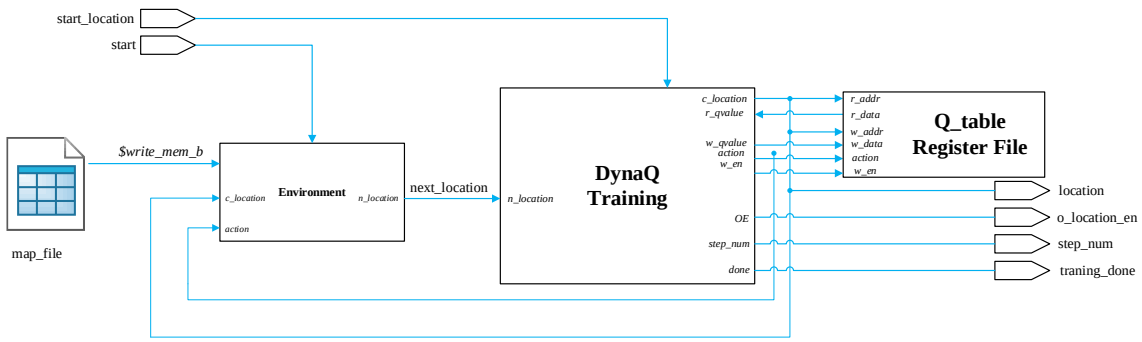


Figure 2.1: Dyna-Q's algorithm system deign

Block diagram of Dyna-Q's algorithm is shown in Figure 2.1. This system contains three main blocks and their main purpose of these is described in Table 2.1.

Table 2.1: The purpose of each block in system design

Block name	Purpose
Environment	– Read the environment data which is involved in the *map_file.txt* file. – Receiving some signals from *Dyna-Q Training* and send back the *next location* and *reward*.
Dyna-Q Training	– Depend on q-value from Q-Table Register File to produce an action.

	– Calculating new *q-value* and store into *Q-Table Register File.*
Q-Table Register File	– Storing the *q-value*.

2.1. Environment block

As the Table 2.1, the most important of this block is determine next location and reward, then send it to *Dyna-Q Training* block through *n_location* signals. To implement this task, the design of this block is in Figure 2.2.

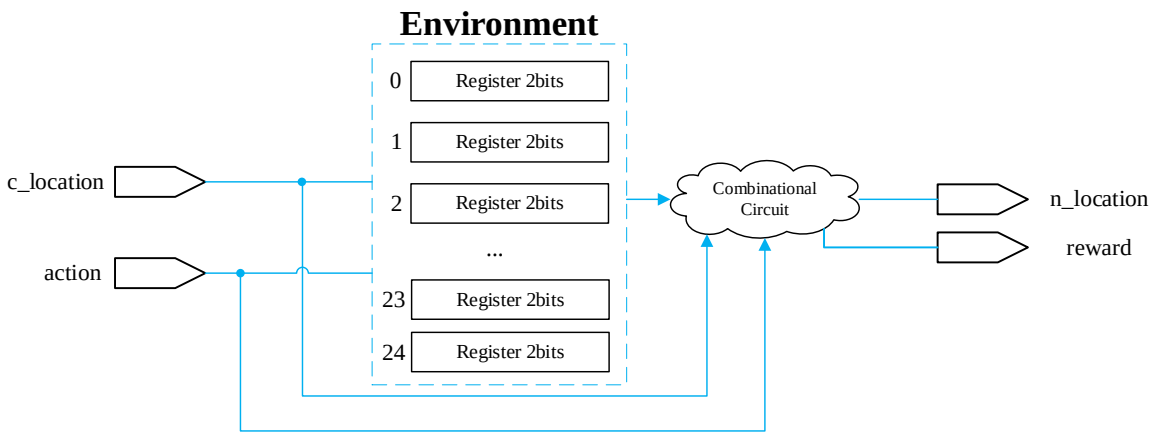


Figure 2.2: The inside of Environment block

Each signal is described in Table 2.2.

Table 2.2: Describing signals of Environment block

Signals	Direction	Length (bits)	Description
c_location	Input	5	Standing for *current location.*
action	Input	2	Action of the hero. Maybe go up down, left, right.
n_location	Output	5	Standing for *next location* which is location of hero after hero will take the action.
reward	Output	2	Reward for each action.

As Figure 2.2, the Environment block contains a set of 25 registers with each length is 2 bits which is read from *map_file.txt* by *$write_mem_b* task. These 2 bits will be decoded as the *Table 2.3*.

Table 2.3: Decode value in Environment block

Value	**Meaning**
00	Nothing
01	x
10	Demon
11	Treasure

As the Table 2.3, the demon is decoded as *2'b10*, treasure as *2'b11* and Normal Road as *2'b00*.

The reward in this circuit is also decoded as Table 2.4.

Table 2.4: Reward decode value

Meet	**Reward**
Demon	10
Out of the map	10
Normal Road	00
Treasure	01

According to Table 2.4, when our hero meets a Demon or run out of the map the reward it receives is *2'b01*. When the square has nothing (Normal Road) the reward is *2'b00* means nothing. When the square contains the treasure, the reward will be *2'd01*.

By accessing this set of registers through *c_location* and *action* signals, we will have an information whether this action is led our hero to a demon's location or not. If it does the *n_location* will equal *c_location*, else the *n_location* signal will be

calculated by *c_location* and *action* signal. Another task of this combinational circuit is to determine whether the hero is out of the map or not and give to next block the *n_location* signal truly.

2.2. Q-Table block

Similar with the Environment block, Q-Table is a set of 25x4 registers 16 bits which contain the q-value of 4 actions for each location. The design of this block is in Figure 2.3 and each signal is described in Table 2.5

Table 2.5: Describing signals of Q-Table block

Signal	**Direction**	**Length (bits)**	**Description**
action	Input	2	Action of the hero. Maybe go up down, left, right. Used for selecting which register is written in block that is select is *w_address*.
w_address	Input	5	Standing for *write address* which is used for selecting which block register will be select.
w_data	Input	5	Standing for *write data*. This is new q-value will store in the register which is select by action and w_address.
w_en	Input	1	Write enable signal for register file.
r_address	Input	5	Standing for read *data*. This signal is used for select which block register will send to Dyna-Q Training block for calculation.
r_data	Output	16x4	Standing for *read data* which is associate with re r_address.

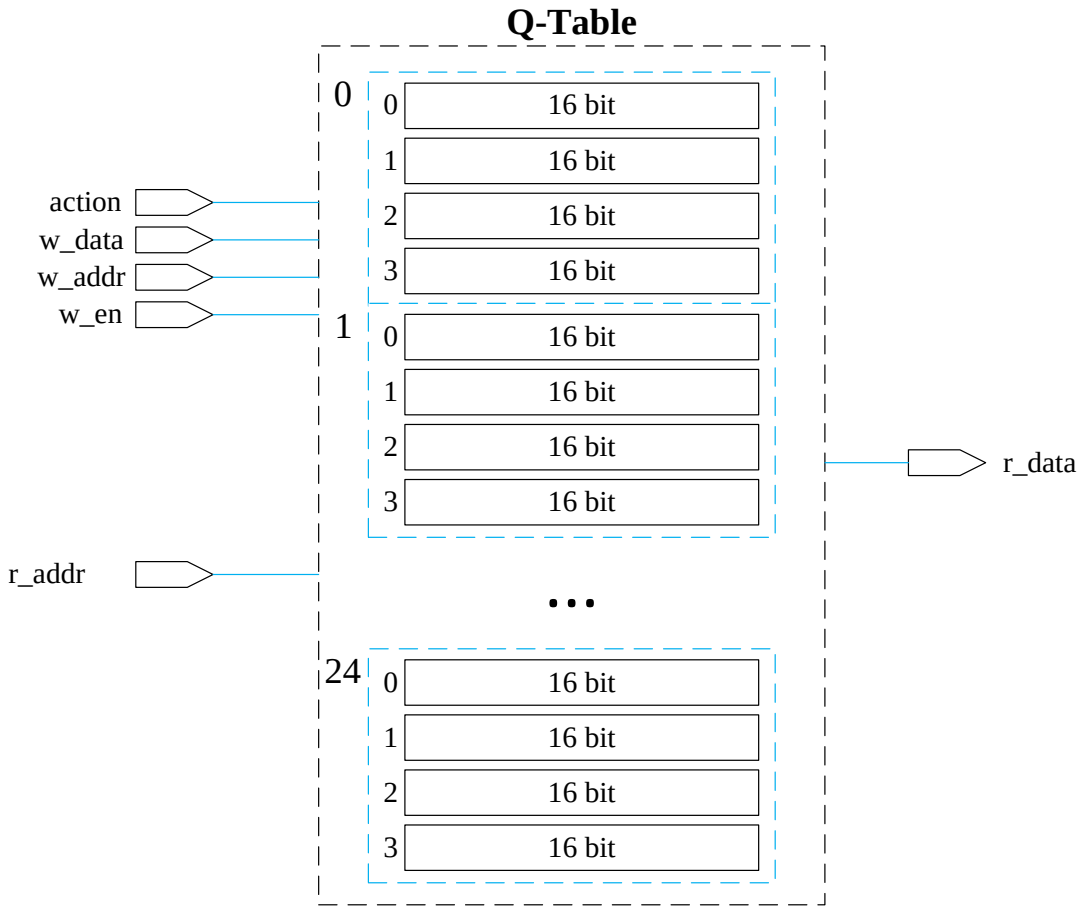


Figure 2.3: The inside of Q-table block

2.3. Dyna-Q Training block

The main purpose of this block is to calculate the q-value and update Q Table block to find the best road to the treasure. To do this task, this block is design as an application specific integrated circuit (ASIC) which contains controller and data-path. The design of this block is shown in Figure 2.4 and its signals is described in Table 2.6.

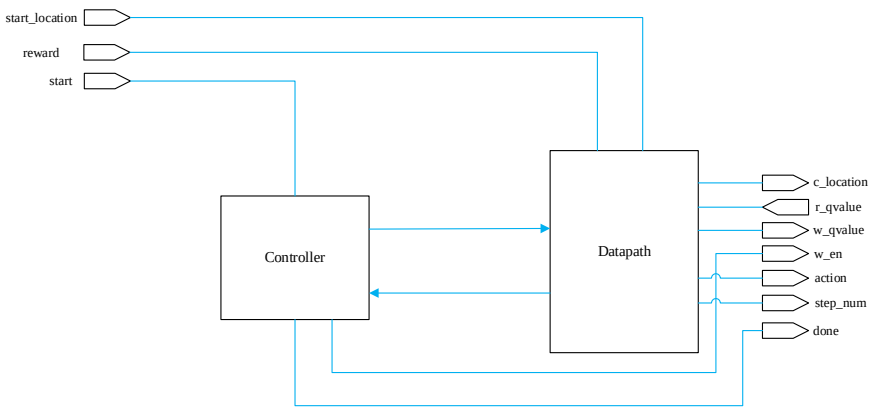


Figure 2.4: An ASIC's structure

Table 2.6: Describing signal of Dyna-Q Training block

Signal	**Direction**	**Length (bits)**	**Description**
start_location	Input	5	Start location which is got from user.
start	Input	1	When this signal is active, this block begins to run and find the best route.
c_location	Output	5	Standing for *current location* which is location of hero.
r_qvalue	Input	16x4	Standing for *read q-value* which is read data from Q-table.
w_qvalue	Output	16	Standing for *write q-value* which is new value of q-register associate with *c_location* and *action*.
w_en	Output	1	Standing for *write enable* which is active if this block needs to write to Q-table.
action	Output	2	The direction of hero will do which is depended on r_qvalue or randomly.

step_num	Output	4	Standing for *number of step* which is the number of step that hero needs to solve the maze.
done	Output	1	This signal is active when the hero find the best route.

Chapter 3. Dyna-Q Design

In Chapter 2, we have already discussed about the system design of Dyna-Q's accelerator. In this chapter we will discuss more detail about the Dyna-Q Training block which is the main and the most important of this accelerator. Things we will discuss are:

- Algorithm State Machine (ASM) of Dyna-Q Training.
- The datapath of Dyna-Q Training.
- The controller of Dyna-Q Training.

3.1. Algorithm State Machine (ASM)

In the Figure 3.1, the ASM contains 16 states and the purpose of each state is described in Table 3.1.

Table 3.1: Describing each state in ASM

State	Task
State_0	– Used for initial some register in this accelerator. – From State_0 to State_1 we need to active the start signal.
State_1	– Read data from Q-Table associate with *current state* and write it into *qvalue_reg*. – State_1 go to State_2 without any signal.
State_2	This state has things to do: – Find the direction that have max value of *qvalue_reg* which is contains 4 *qvalue* of 4 direction which is associate with *current location*. – Generate a random direction and an epsilon number.

	– Compare generated epsilon number which epsilon threshold value and select random direction or max direction. – Plus 1 on *step count*. State_2 will go to State_3 when the *step_count* less than 25, if not the State_2 will go to State_1 to start a new episode. 25 is called *step_limitation*.
State_3	– Send *c_location* and *action* to environment and it will send back next location and reward.
State_4	– Read data from Q-Table associate with *n_location* and write it into *qvalue_reg*.
State_5	– Find the max value of *qvalue_reg* which is contains 4 *qvalue* of 4 direction which is associate with *n_location*.
State_6	– In true Dyna-Q algorithm, *g = max_reg * gamma*. In this case, gamma is equal 0.5, *so g = max_reg >> 1* is used to reduce the hardware utilization.
State_7	– Do the addition operation between *reward* and *g*. Beside copy the *n_location_reg* into *tn_location_reg*. – Check the value in *max_reg* to remembering's mechanism. If the n_location reach a location which have *qvalue* > 0, then it will update *q_register* associate with *hc_location* and *haction*.
State_8	– Store the *qvalue* in to Q-Table.
State_9	– Send *hc_location* and *haction* to environment and it will send back next location *(hn_location)* and reward *(h_reward).*
State_10	– Read data from Q-Table associate with *n_location* and write it into *qvalue_reg*.

State_11	– Find the max value of *qvalue_reg* which is contains 4 *qvalue* of 4 direction which is associate with *next location*.
State_12	– In true Dyna-Q algorithm, *g = max_reg * gamma*. In this case, gamma is equal 0.5, *so g = max_reg >> 1* is used to reduce the hardware utilization.
State_13	– Do the addition operation between *reward* and *g*.
State_14	– Store the qvalue in to Q-Table by *hc_location* and *haction*. – Store *c_locati*on and *action* into *hc_location* and *haction*. – If it reaches the treasure then go to State_0 and done the episode or go to the State_1 for next step.
State_15	– Store the qvalue in to Q-Table by *c_location* and *action*. – Store *c_locati*on and *action* into *hc_location* and *haction*. – If it reaches the treasure then go to State_0 and done the episode or go to the State_1 for next step.

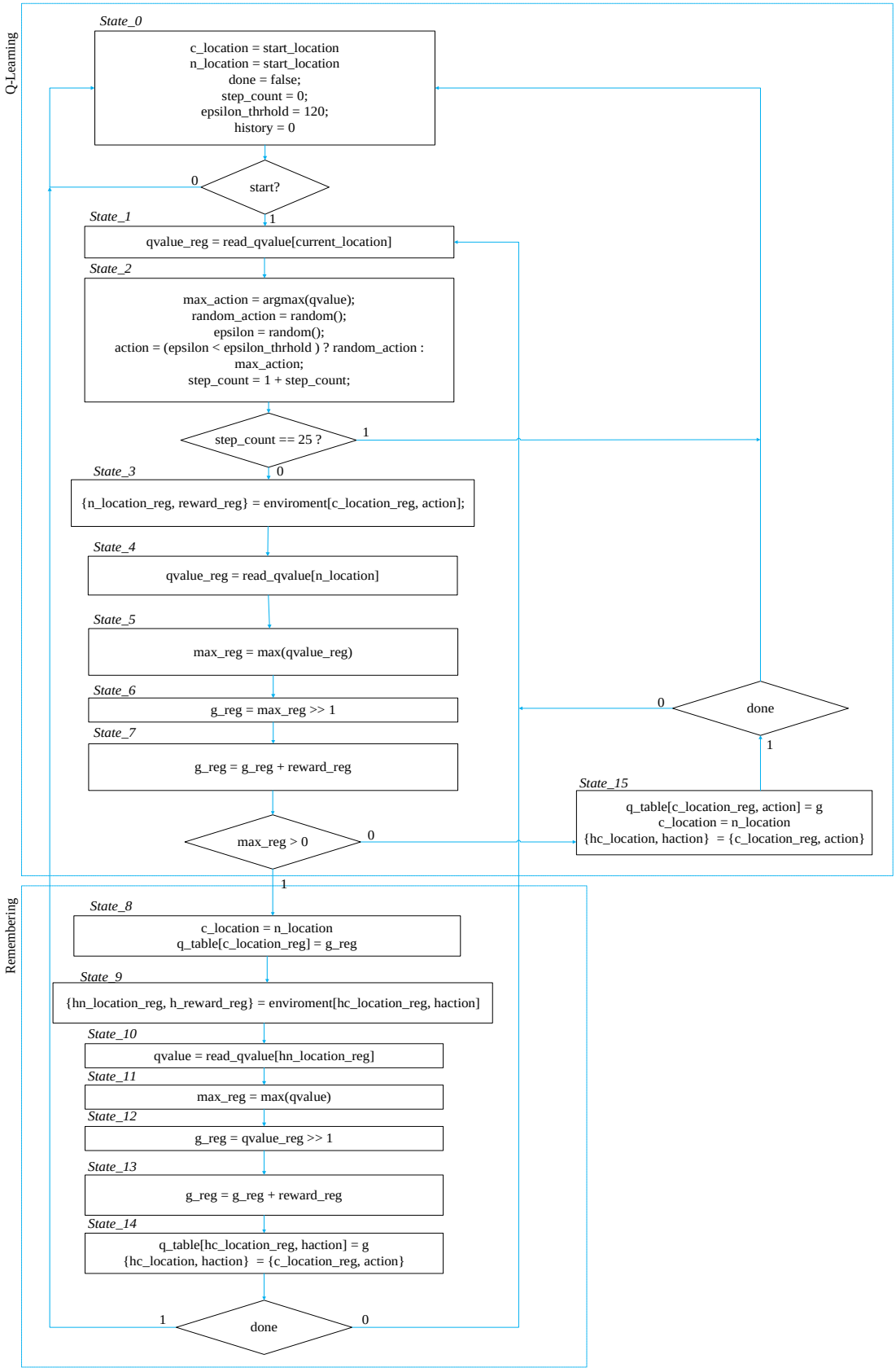


Figure 3.1: The ASM of Dyna-Q Algorithm

3.2. Datapath

Figure 3.2 described the data-path of this accelerator.

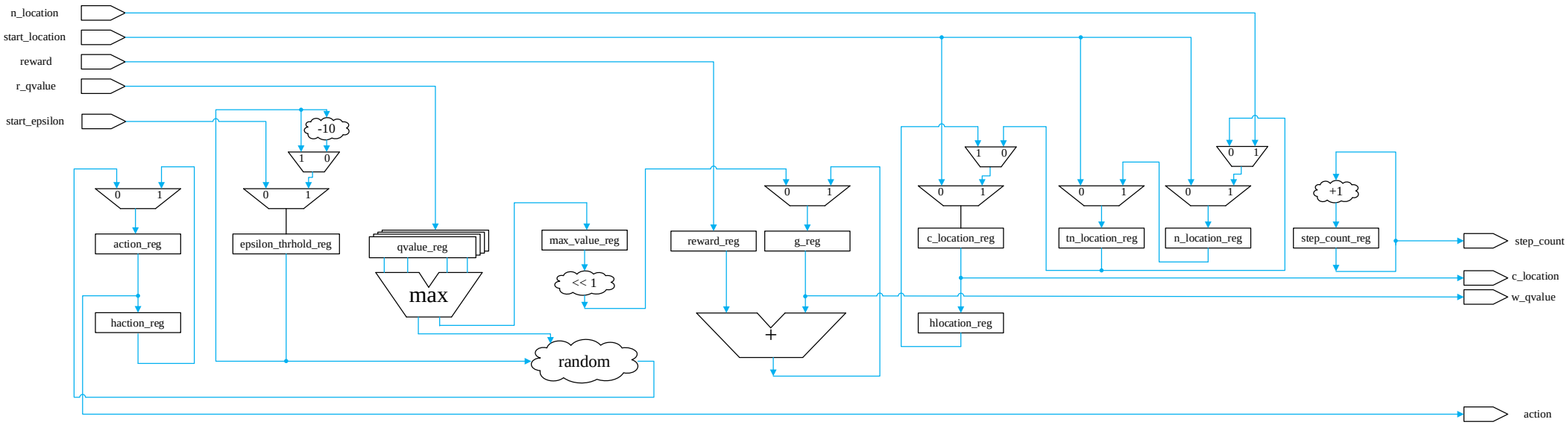


Figure 3.2: The data-path of Dyna-Q's accelerator

In this data-path, all blocks in this accelerator are simple and there is some main block:

– *max* block: Determine max value of 4 value in *qvalue_reg* and its direction.

– *random* block: Generate a random action and random epsilon to find the truly action.

The main purpose of data-path is to calculate the Q-Learning equation (equation 1.1) and the hardware in data-path support for this purpose and it calculates the $Q(S, A)_{new}$ step by step according to Figure 3.1 and the flow of this hardware is described in Table 3.2.

Table 3.2: Describing the Data Flow for the data-path

State	Data Flow
0	– All the register is stored the initial value synchronously. One of them can take some input signal to store or take a constant. For example: + c_location_reg = start_location. + n_location_reg = start_location. + step_count_reg = 0. + epsilon_threshold_reg = start_epsilon = 120.
1	– This state is used for read data from Q-Table. The read data from Q-Table through *r_value* signal go into the data-path then is stored directly to *qvalue_reg*.
2	– Determine the max direction associate with 4 *r_value*: + The *r_value* in *r_value_reg* go to the *max* block. Then give us the direction of the max value which is called *max_action* and send thí value to *random* block. – Beside *random* block also generates a *random_action* and *random_epsilon.* Then it use this data to determine the true action by the equation: (*random_epsilon* < *epsilon_threshold*) *? random_action : max_action.* Then store it into *action_reg*. – It also increases the *step_count* value by 1.

3	– In State 3, the data-path send *c_location* and *action* to read *n_location* and *reward* from the *Enviroment* block. These signals go through input of this block and is stored into 2 registers: *n_location_reg* and *reward_reg*.
4	– The same as the State_1, this state also read the Q-Table but the read address is *n_location* from *n_location_reg.*
5	– This find the max value of 4 *qvalue* which has been read in State_4. The data in *qvalue_reg* is sent to max block and the max value is stored to *max_reg*.
6	– This step takes the value in *max_reg* then shit right 1 bit then store the result in to *g_reg*.
7	– This stage takes the value in *g_reg* and *reward_reg* which have been calculated or read in previous state. Then it add these signal by the addition block. The output of the addition block is written back the *g_reg*.
8	– This state used for store the $Q(S, A)_{new}$ which is calculated in State_7 and is store in *g_reg* in to Q-table. The value in c_location_reg, *action_reg* and *g_reg* is send to Q-Table through *c_location*, *action* and *w_qvalue* output. – Beside this state also update the *c_location* reg by storing the *n_location* into it.
9	– In State 9, the data-path send *hc_location* and *haction* to read *n_location* and *reward* from the *Enviroment* block. These signals go through input of this block and is stored into 2 registers: *hn_location_reg* and *hreward_reg*.

10	– This state also read the Q-Table but the read address is *hn_location* from *hn_location_reg*.
11	– This find the max value of 4 *qvalue* which has been read in State_10. The data in *qvalue_reg* is sent to max block and the max value is stored to *max_reg*.
12	– This step takes the value in *max_reg* then shit right 1 bit then store the result in to *g_reg*.
13	– This stage takes the value in *g_reg* and *reward_reg* which have been calculated or read in previous state. Then it adds these signals by the addition block. The output of the addition block is written back the *g_reg*.
14	– This state used for store the $Q(S, A)_{new}$ which is calculated in State_13 and is stored in *g_reg* in to Q-table. The value in *h*c_location_reg, *haction_reg* and *hg_reg* is send to Q-Table through *c_location*, *action* and *w_qvalue* output.
15	– This state used for store the $Q(S, A)_{new}$ which is calculated in State_7 and is stored in *g_reg* in to Q-table. The value in c_location_reg, *action_reg* and *g_reg* is send to Q-Table through *c_location*, *action* and *w_qvalue* output.

3.3. Controller

Controller is the main part as well as Datapath. This part is used for controlling the Datapath to function normally. The operation of controller is described in Table 3.3.

Table 3.3: Describing for controller

State	Control description
0	– Enable store the register that need to initial. – Select initial value. – Checking the *start* signal. If *start* signal is active, it will go to the State 1.
1	– Enable to write to the *qvalue_reg*.
2	– Enable to write to the *max_action* register. – Enable to write to *action_reg*. – Checking the *step_count* register. If the *step_count* is equal 25 then go to the State 0 to start a new episode. If not, going to the State 3 to continue the process.
3	– Enable to write to the *n_location_reg* and *reward_reg*.
4	– Enable to write to the *qvalue_reg*.
5	– Enable to write to the *max_reg*.
6	– Enable to writte to *g_reg*.
7	– Enable to writte to *g_reg*. – Checking *max_reg*. If max_reg > 0 then go to State_8, if not go to State_15.
8	– Enable to write to *c_location* register. – Active the *w_qvalue_en*.
9	– Enable to write to the *hn_location_reg* and *hreward_reg*.
10	– Enable to write to the *qvalue_reg*.

11	– Enable to write to the *max_reg*.
12	– Enable to writte to *g_reg*.
13	– Enable to writte to *g_reg*.
14	– Active the *w_qvalue_en*. – Enable to write to the *hc_location* and *haction* registers.
15	– Active the *w_qvalue_en*. – Enable to write to *c_location* register. – Enable to write to the *hc_location* and *haction* registers.

Chapter 4. Result Evolution

4.1. Verification Flow

Figure 4.1 show the Verification flow which has 7 steps from run in software (Questa Sim) to run on Altera DE2 KIT with the support of a Python code on computer.

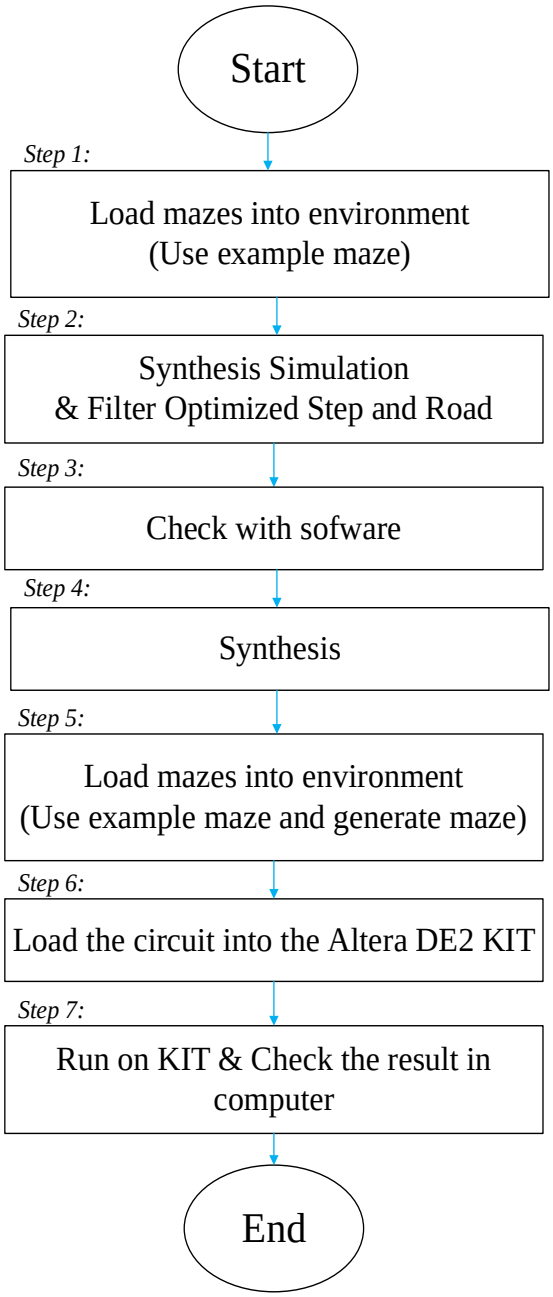


Figure 4.1: The verification flow

Table 4.1 describes detail about each step in the flow.

Table 4.1: The description of each step in Verification flow

Step	**Description**
Step 1: Load mazes into environment.	– By create a text file (map_file.txt) hold the example map information from LSI Design Contest 2021 and load it into Environment's register file by *$write_mem_b* task.
Step 2: Synthesis simulation & Filter Optimized step and Road.	– Run the circuit in QuestaSim. – Filter the step and Road
Step 3: Check software.	– Run this algorithm in software version. – Compare the result in hardware and software.
Step 4: Synthesis	– Run synthesis this design.
Step 5: Load mazes into environment.	– Use the example maze or generate random maze to test
Step 5: Load the circuit into Altera DE2 KIT.	– Load this design into the Altera DE2 Board.
Step 6: Run the circuit and Check the result in computer.	– Training this machine and this will send the road with optimized step to the computer and we can check more easier than waveform. – Before training, we can config the start location which help us can check the circuit more deeply without generate too much mazes.

As Table 4.1 and Figure 4.1, the verification flow is from check in waveform to KIT and the maze is randomized by Python. That why's this process is reliable.

4.2. Simulation

Figure 4.2 shows the training process which is reduce epsilon threshold when the hero reaches the treasure and when epsilon is zero the training process is done. When the process is done, it needs a signal to start generate step.

```
VSIM 53> run
# Epsilon 120 : DONE !!!
# Epsilon 110 : DONE !!!
# Epsilon 100 : DONE !!!
# Epsilon 90 : DONE !!!
# Epsilon 80 : DONE !!!
# Epsilon 70 : DONE !!!
# Epsilon 60 : DONE !!!
# Epsilon 50 : DONE !!!
# Epsilon 40 : DONE !!!
# Epsilon 30 : DONE !!!
# Epsilon 20 : DONE !!!
# Epsilon 10 : DONE !!!
# Epsilon 0 : DONE !!!
```

Figure 4.2: The training report

Figure 4.3 shows waveform when the training process is done and show the result.

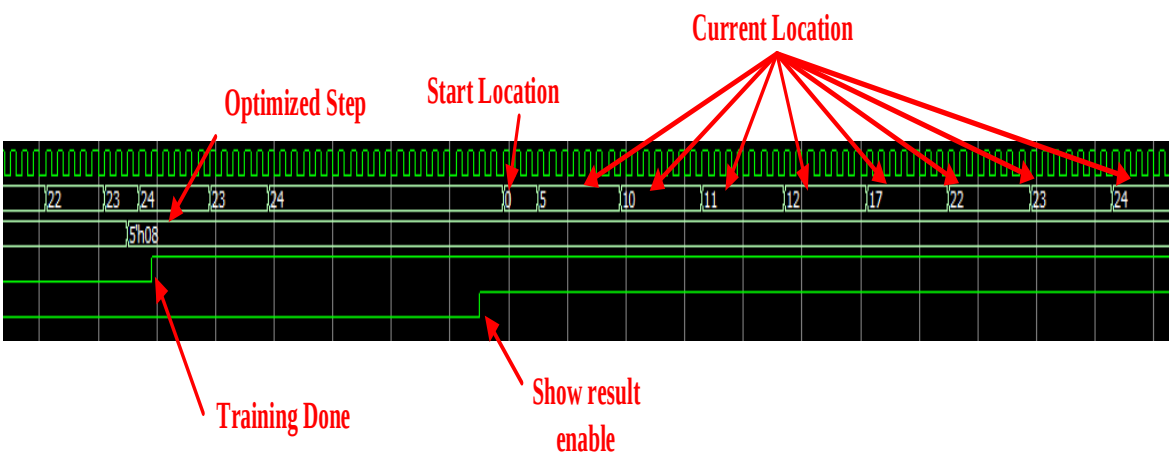


Figure 4.3: Pre-Synthesis Waveform

4.3. Synthesis & Critical path speed

In this project, we use Altera DE2 Education and Development board to run this implement. The hardware utilization is shown in Table 4.2.

Table 4.2: Hardware utilization and Max Frequency of Dyna-Q and Q-learning accelerator

	Dyna-Q	**Q-learning**
Logic Utilization (in ALMs)	2,403 / 33,216 (7 %)	2,325 / 33,216 (7 %)
Total register	1763	1741
Max Frequency	143.64 MHz	156.99 MHz

As you can see in Table 4.2, the hardware utilization between Dyna-Q and Q-learning's is not different too much, but the result may be not the same. In Table 4.3 and Table 4.4, as easy to see that the Dyna-Q algorithm's result much better than Q-learning's.

Figure 4.4 show the critical path of this accelerator. The critical path is the read Q -Table path which is State 1, State 4, State 10 in the ASM (Figure 3.1).

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
3	0.999	0.999	RR	CELL	1	IOC_X0_Y18_N2	clk|combout
4	1.117	0.118	RR	IC	1	CLKCTRL_G3	clk~clkctrl|inclk[0]
5	1.117	0.000	RR	CELL	1763	CLKCTRL_G3	clk~clkctrl|outclk
6	2.138	1.021	RR	IC	1	LCFF_X37_Y19_N15	training|training_controller|current_state[4]|clk
7	2.675	0.537	RR	CELL	1	LCFF_X37_Y19_N15	dynaQ_training:training|dynaQ_training_con...er_v2:training_controller|current_state[4]
3	⌄ 9.575	6.900					data path
1	2.925	0.250		uTco	1	LCFF_X37_Y19_N15	dynaQ_training:training|dynaQ_training_con...er_v2:training_controller|current_state[4]
2	2.925	0.000	RR	CELL	29	LCFF_X37_Y19_N15	training|training_controller|current_state[4]|regout
3	3.257	0.332	RR	IC	1	LCCOMB_X37_Y19_N10	training|training_datapath|o_action[0]~1|datab
4	3.676	0.419	RR	CELL	20	LCCOMB_X37_Y19_N10	training|training_datapath|o_action[0]~1|combout
5	3.947	0.271	RR	IC	1	LCCOMB_X37_Y19_N4	training|o_next_location_data[0]~0|datad
6	4.097	0.150	RR	CELL	1	LCCOMB_X37_Y19_N4	training|o_next_location_data[0]~0|combout
7	4.363	0.266	RR	IC	1	LCCOMB_X37_Y19_N26	training|o_next_location_data[0]~1|datac
8	4.638	0.275	RR	CELL	338	LCCOMB_X37_Y19_N26	training|o_next_location_data[0]~1|combout
9	5.680	1.042	RR	IC	1	LCCOMB_X42_Y19_N16	q_table|Mux55~2|datab
10	6.100	0.420	RR	CELL	1	LCCOMB_X42_Y19_N16	q_table|Mux55~2|combout
11	6.360	0.260	RR	IC	1	LCCOMB_X42_Y19_N26	q_table|Mux55~3|dataa
12	6.798	0.438	RR	CELL	1	LCCOMB_X42_Y19_N26	q_table|Mux55~3|combout
13	7.506	0.708	RR	IC	1	LCCOMB_X40_Y19_N6	q_table|Mux55~4|datab
14	7.922	0.416	RR	CELL	1	LCCOMB_X40_Y19_N6	q_table|Mux55~4|combout
15	9.216	1.294	RR	IC	1	LCCOMB_X36_Y14_N16	q_table|Mux55~15|datac
16	9.491	0.275	RR	CELL	1	LCCOMB_X36_Y14_N16	q_table|Mux55~15|combout
17	9.491	0.000	RR	IC	1	LCFF_X36_Y14_N17	training|training_datapath|qvalue_register[0]|r_data[8]|datain
18	9.575	0.084	RR	CELL	1	LCFF_X36_Y14_N17	dynaQ_training:training|dynaQ_training_dat...egister_reset:qvalue_register[0]|r_data[8]

Figure 4.4: The critical path of Dyna-Q

4.4. Performance of Dyna-Q vs Q-Learning

To more specific, Table 4.3 and Table 4.4, by config epsilon condition to find the best decision when make a decision in action randomly or not. There are 2 possible case we think that is the best epsilon condition. Beside config the start location is the way to change the hero's situation which is the same as the change the map by change the hero's start location. *"Episode"* column shows the episode that done the training process and "At time (ns)" column is at this time the accelerator done the training process. The cycle in Table 4.3 and Table 4.4 is *100ps*.

Table 4.3: The comparation of Dyna-Q and Q-Learning's result (1)

Epsilon Condition	**Start Location**	**Optimized Step**	**Dyna-Q**	**Q-Learning**
			At time (ns)	**At time (ns)**
Epsilon equal 120 and is subtracted for 10 for each time reach the treasure	1	8	4877.90	6138.30
	2	9	10328.70	8497.30
	3	10	15606.70	22565.50
	4	11	23600.50	28332.70
	9	12	79861.50	80666.50
	10	13	88760.70	110173.50
	15	14	403056.70	475482.50
	6	7	3853.70	3393.50
	11	6	1284.70	1055.30
	16	7	1805.10	2059.70
	21	8	1607.30	1407.90
	12	5	850.10	636.50
	13	4	475.90	298.50

Table 4.4: The comparation of Dyna-Q and Q-Learning's result (2)

Epsilon Condition	**Start Location**	**Optimized Step**	**Dyna-Q**	**Q-Learning**
			At time (ns)	**At time (ns)**
Epsilon equal 120 and is subtracted for 20 for each time reach the treasure	1	8	5452.30	Cannot Solve
	2	9	11399.50	
	3	10	24088.50	
	4	11	24105.30	
	9	12	81745.90	
	10	13	98328.70	397293.10
	15	14	364627.50	445762.90
	6	7	2792.50	2675.90
	11	6	972.50	1258.30
	16	7	1766.50	Cannot Solve
	21	8	1016.70	1538.30
	12	5	522.10	426.30
	13	4	291.70	212.70

With the condition is *"Epsilon equal 120 and is subtracted for 10 for each time reach the treasure"* in Table 4.3, this means 100% the accelerator select the action randomly in the first and reduce 12% for each time it reaches the treasure. In this case, Q-learning maybe solve the 5x5 maze problem better than Dyna-Q learning. Because Dyna-Q take some cycles to remember and this process take time. In the case of Dyna-Q, it can solve the maze with any start location with the time depend on the complication of the maze.

With the condition is *"Epsilon equal 120 and is subtracted for 20 for each time reach the treasure"* in Table 4.4, this means 100% the accelerator select the action randomly in the first and reduce 24% for each time it reaches the treasure. In

this case, Q-learning can not solve the maze in some case. Dyna-Q it can solve the maze with any start location with the time depend on the complication of the maze.

Through Table 4.3 and Table 4.4, Q-learning is not good enough to solve 5x5 maze or even cannot find the road to the treasure. To compare the Dyna-Q in 2 condition, these conditions have the advantage and disadvantage which is shown in Table 4.5

Table 4.5: The comparation between two epsilon condition.

No	Condition	Advantage	Disadvantage
1	*"Epsilon equal 120 and is subtracted for 10 for each time reach the treasure"*	Can solve complicated mazes in the less time.	Take more time when solve simply mazes.
2	*"Epsilon equal 120 and is subtracted for 20 for each time reach the treasure"*	Can solve complicated mazes in the less time even less than condition 2 in some case.	Maybe more risk because the epsilon reduces too fast.

4.5. Implementation and Testing

Figure 4.5 show the implement and testing flow which need to interface with board.

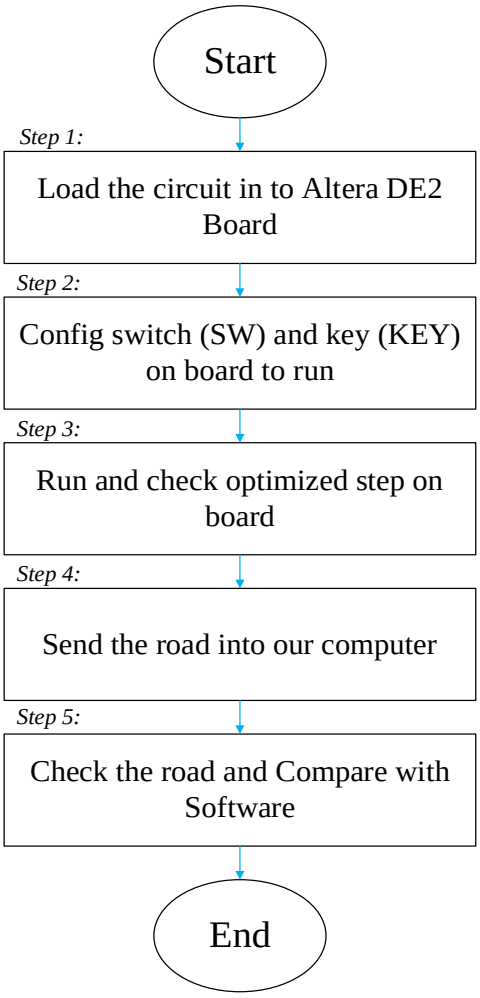


Figure 4.5: Implement and Testing flow

Each step is described in Table 4.6.

Table 4.6: Describing each step in Implement and Testing Flow

Step	**Description**
1	Load the design to DE2 Board through JTAG capable.
2	Config the SW to choose start location and press KEY to run the design.
3	After run, check LED 7 SEG and status LED.
4	Press a KEY to send the road to computer.
5	See the road on the screen.

To test this machine, we use Python to generate some random maze in loading its into the DE2 Education and Development which contains our accelerator. After it done the training, it will send back the optimized step into our computer through and show it in the screen.

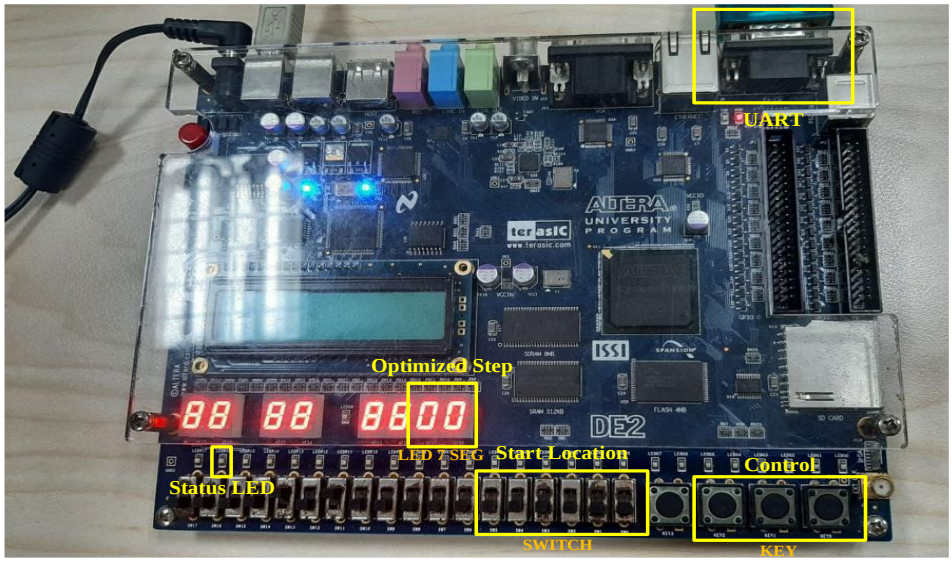


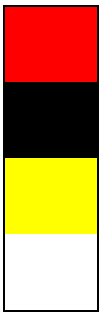
Figure 4.6: KIT test

In addition, some part in this DE2 KIT is used, there are:

- UART connection: Used for send back the road to our computer.
- SWITCH [4:0]: Used for config the start location. With these switches we can test one maze with many *start_location* easily.
- LED [16]: This led will be on when the hardware has done the training section and ready to send the road with lowest step to our computer.
- LED 7 SEG [0] and LED 7 SEG [1]: This will show the optimize step.
- KEY [2:0]: Control the machine. They are
    + KEY[0]: Reset.
    + KEY[1]: Start the training process.
    + KEY[2]: Send the road to the computer.

4.5.1. Demo maze from LSI Contest 2021

In the next section, we will use some color to represent for the square's information. There are:



Demon's Square.

Normal Square.

Treasure Square.

The Optimized Road.

Figure 4.7 show the example maze with red square is demon and yellow square is treasure.

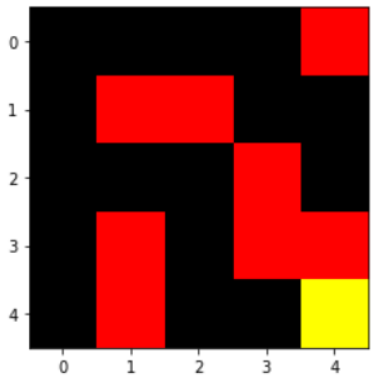


Figure 4.7: Maze from LSI Contest 2021- Intermediate

As the Figure 4.8, the start location is 0 means the value of SW[4:0] is **5'd0**. Running the machine by press some KEY[0] then KEY[1], we have the result in Figure 4.8. The LED 7 SEG[1:0] part show 08, this means to solve this maze we need 8 actions. Beside the LED[16] is on means it ready to send the computer.

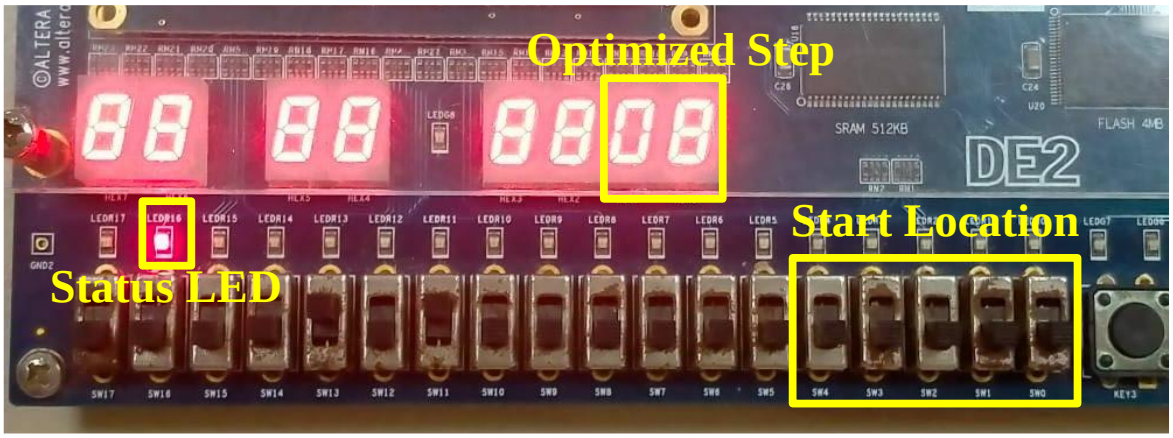


Figure 4.8: The configuration on the DE2 KIT and result

After press KEY[3], the software will take these data from the KIT through UART communication and show it in our computer screen.

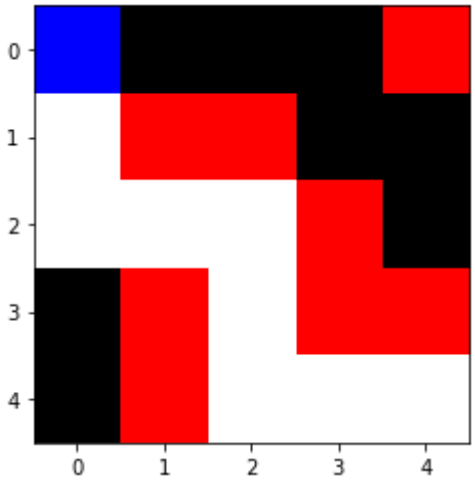
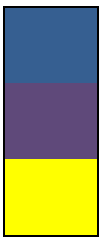


Figure 4.9: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 0) – Hardware version

After we have the result in Figure 4.9. We need to check this road with the result of software. Its use another color set to representation for the map. There are:



Demon's Square.

Normal Square.

The Optimized Road.

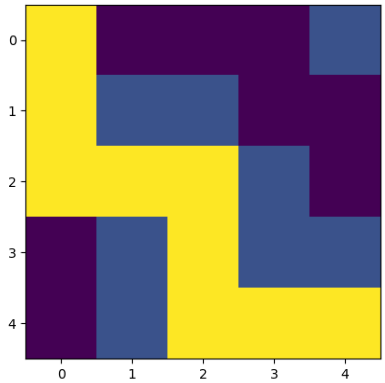


Figure 4.10: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 0) – Software version

We also test this maze with a list of start location and there result in next Figure.

Figure 4.11: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 1) – Hardware version

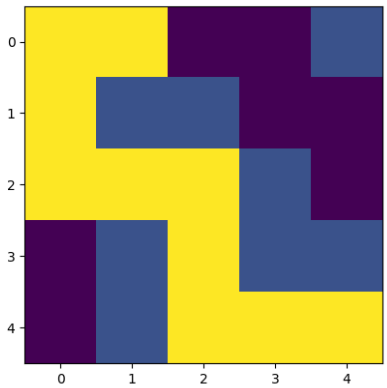


Figure 4.12: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 1) – Software version

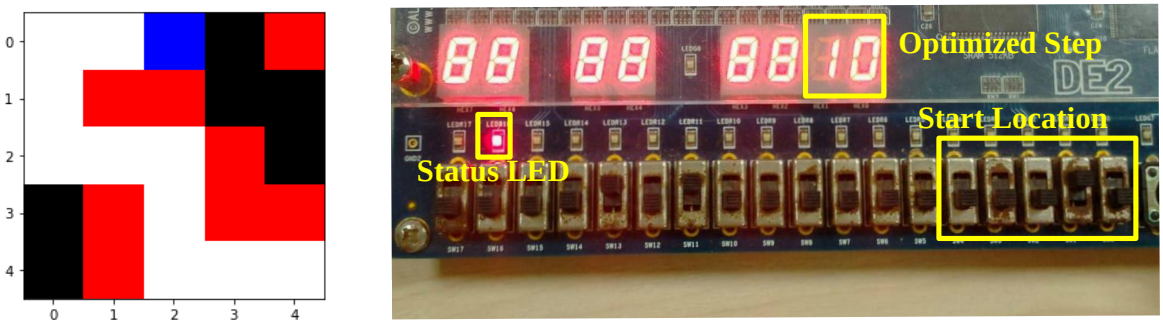


Figure 4.13: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 2) – Hardware version

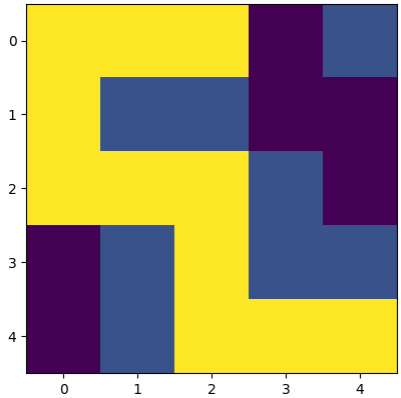


Figure 4.14: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 2) – Software version

Figure 4.15: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 3) – Hardware version

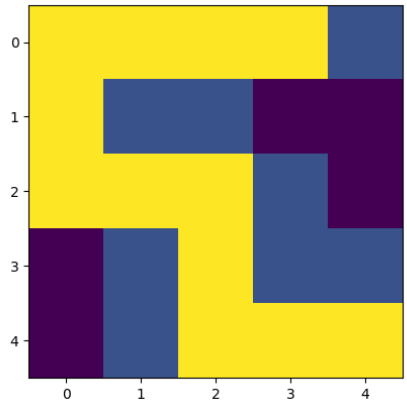


Figure 4.16: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 3) – Software version

Figure 4.17: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 8) – Hardware version

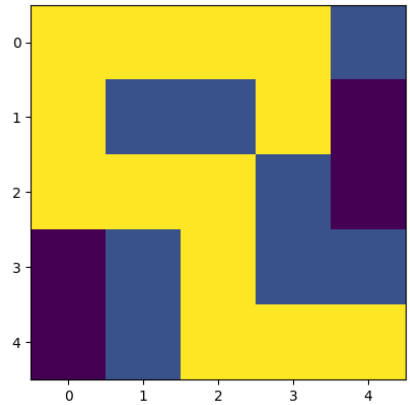


Figure 4.18: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 8) – Software version

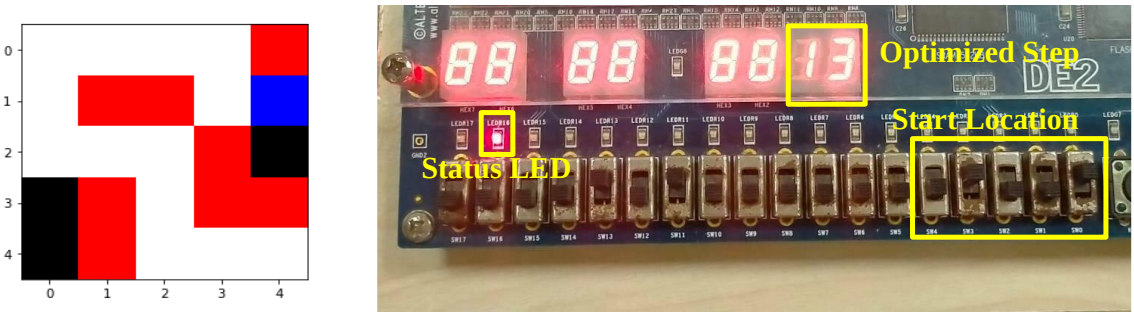


Figure 4.19: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 9) – Hardware version

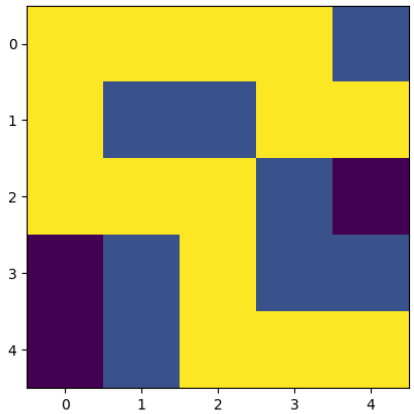


Figure 4.20: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 9) – Software version

Figure 4.21: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 14) – Hardware version

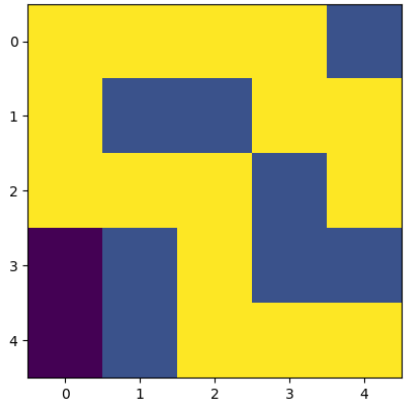


Figure 4.22: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 14) – Software version

Figure 4.23: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 5) – Hardware version

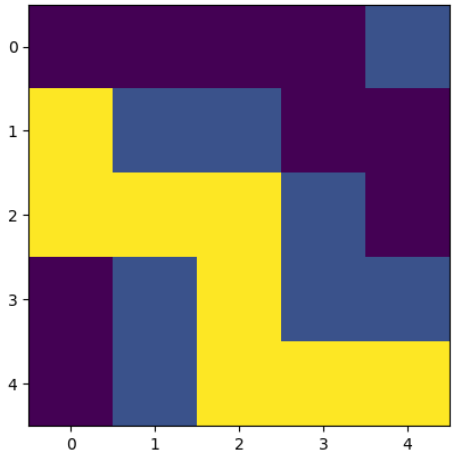


Figure 4.24: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 5) – Software version

Figure 4.25: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 10) – Hardware version

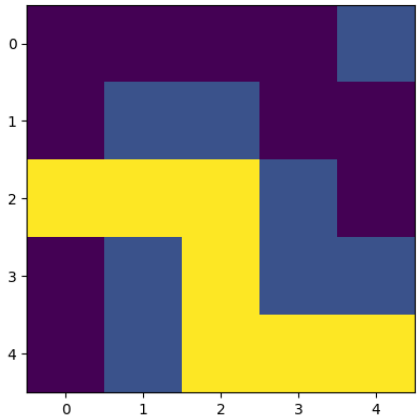


Figure 4.26: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 10) – Software version

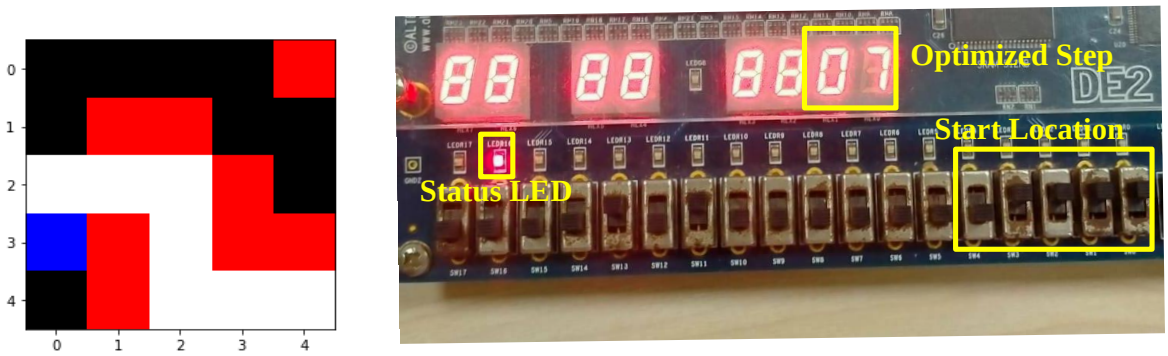


Figure 4.27: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 15) – Hardware version

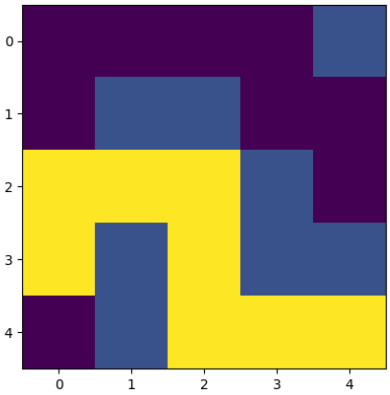


Figure 4.28: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 15) – Software version

Figure 4.29: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 20) – Hardware version

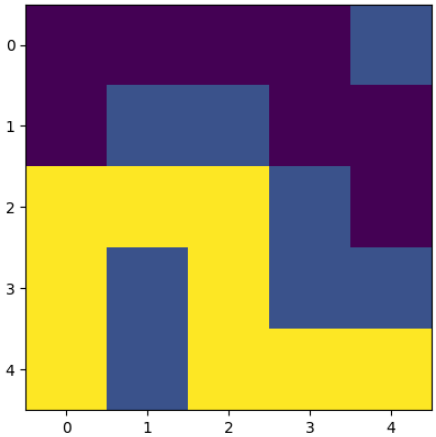


Figure 4.30: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 20) – Software version

Figure 4.31: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 11) – Hardware version

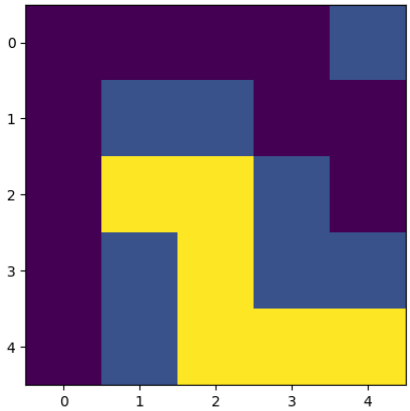


Figure 4.32: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 11) – Software version

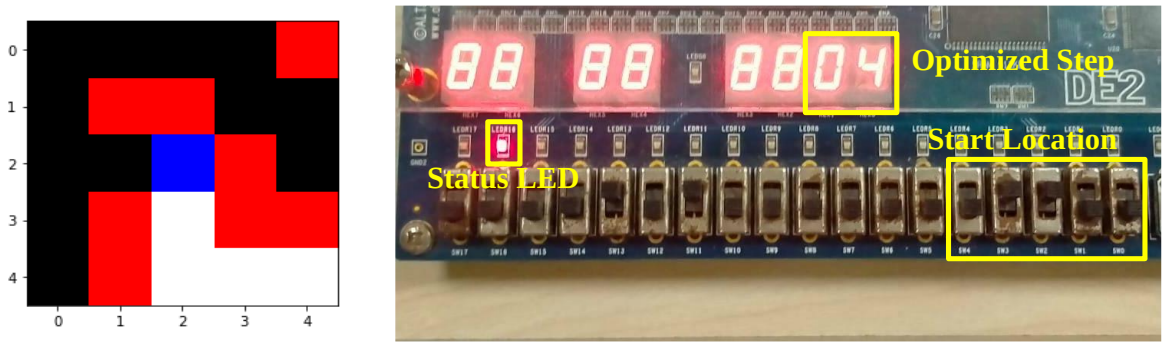


Figure 4.33: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 12) – Hardware version

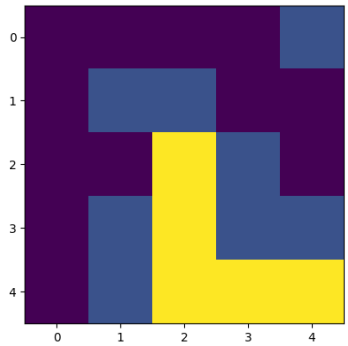


Figure 4.34: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 12) – Software version

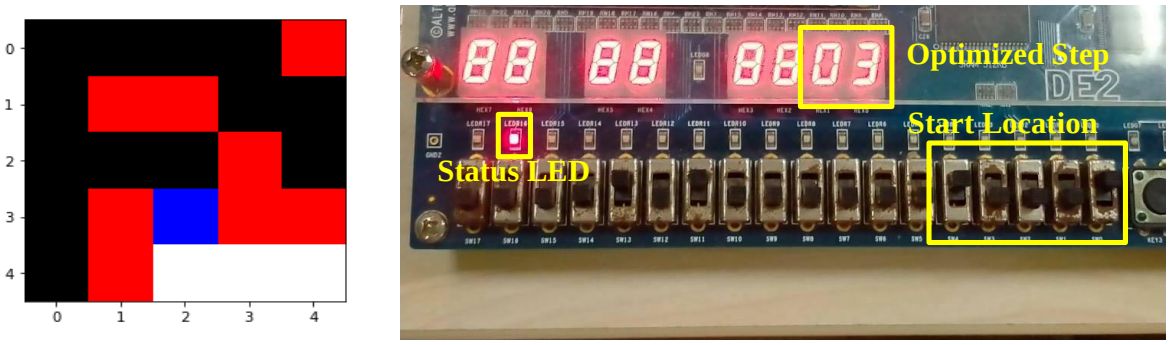


Figure 4.35: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 17) – Hardware version

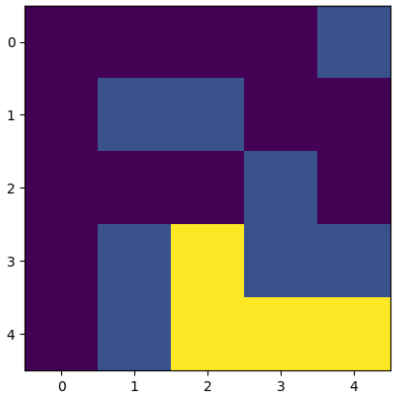


Figure 4.36: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 17) – Software version

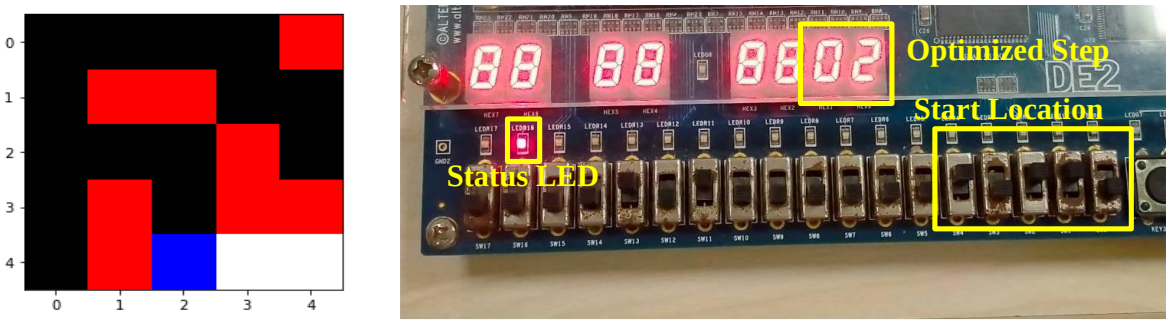


Figure 4.37: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 22) – Hardware version

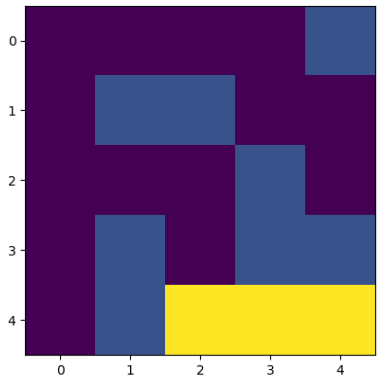


Figure 4.38: Optimized road of the maze from LSI Contest 2021- Intermediate (Start Location is 22) – Software version

4.5.2. Random maze

By using Python, we generate some random maze to test this machine.

4.5.2.1. Maze 1

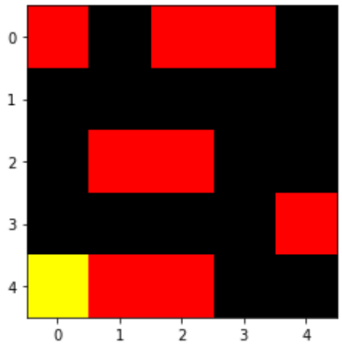


Figure 4.39: Random maze (1)

Figure 4.40: Optimized road of the random maze 1 (Start Location is 4) – Hardware version

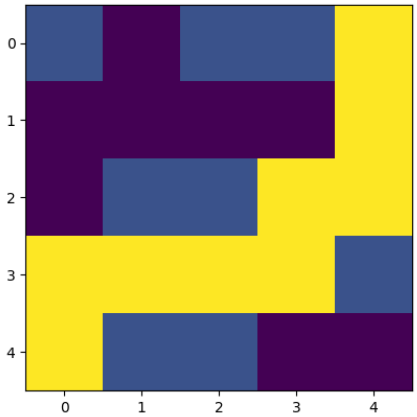


Figure 4.41: Optimized road of the random maze 1 (Start Location is 4) – Software version

Figure 4.42: Optimized road of the random maze 1 (Start Location is 1) – Hardware version

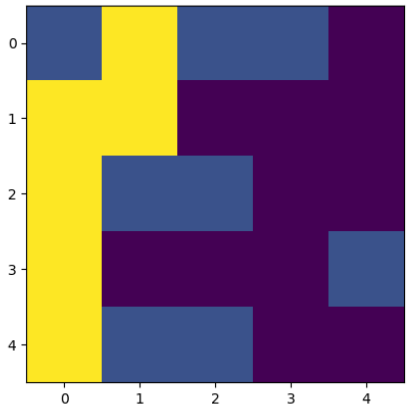


Figure 4.43: Optimized road of the random maze 1 (Start Location is 1) – Software version

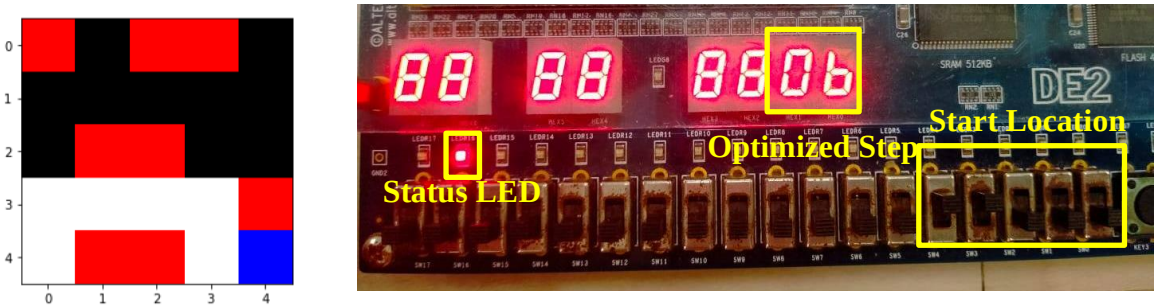


Figure 4.44: Optimized road of the random maze 1 (Start Location is 24) – Hardware version

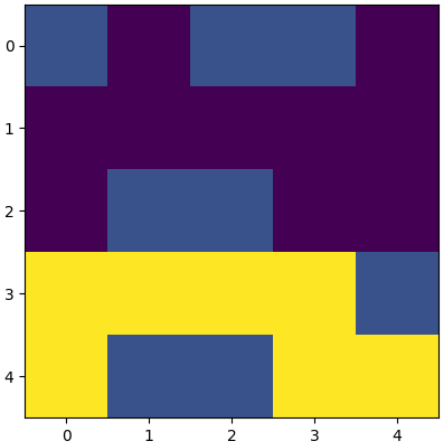


Figure 4.45: Optimized road of the random maze 1 (Start Location is 24) – Software version

4.5.2.2. Maze 2

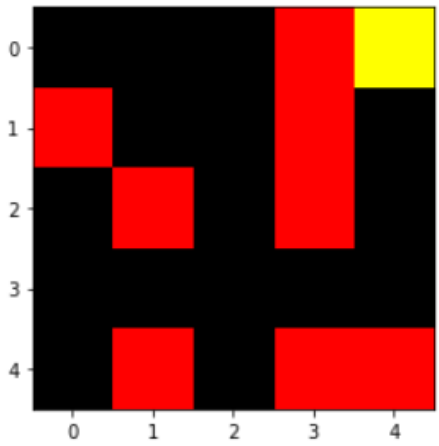


Figure 4.46: Random maze (2)

Figure 4.47: Optimized road of the random maze 2 (Start Location is 0) – Hardware version

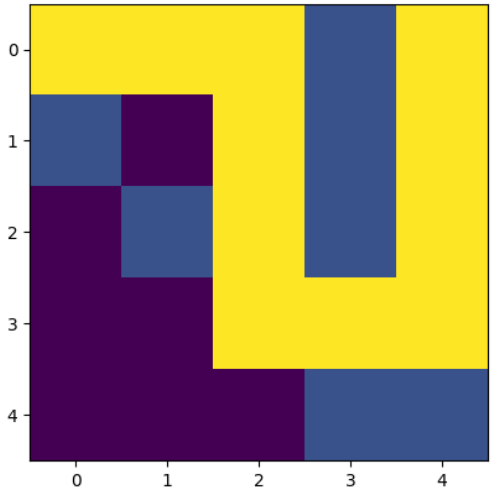


Figure 4.48: Optimized road of the random maze 2 (Start Location is 0) – Software version

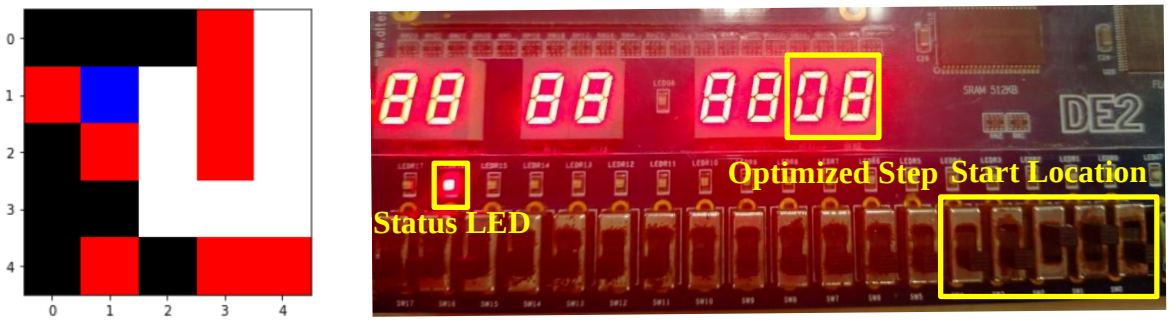


Figure 4.49: Optimized road of the random maze 2 (Start Location is 6) – Hardware version

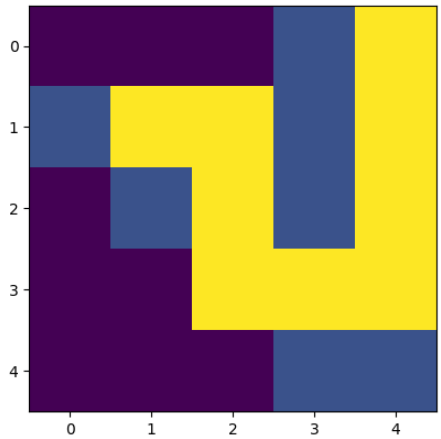


Figure 4.50: Optimized road of the random maze 2 (Start Location is 6) – Software version

Figure 4.51: Optimized road of the random maze 2 (Start Location is 10) – Hardware version

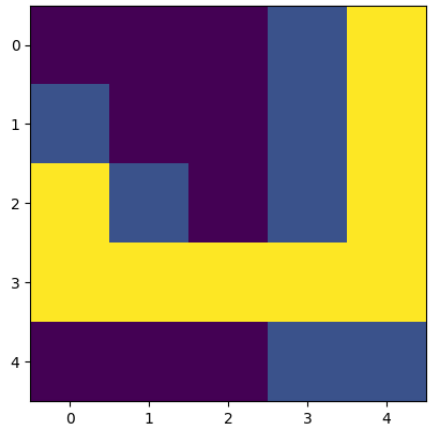


Figure 4.52: Optimized road of the random maze 2 (Start Location is 10) – Software version

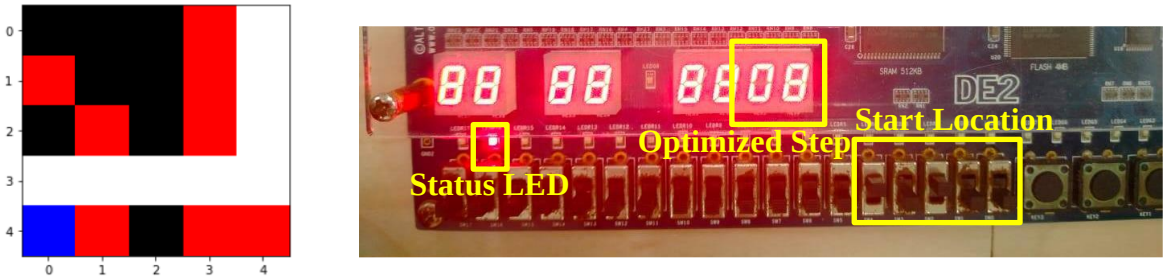


Figure 4.53: Optimized road of the random maze 2 (Start Location is 20) – Hardware version

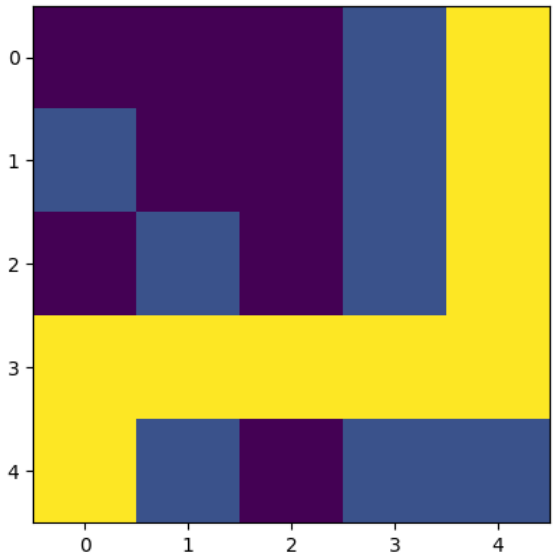


Figure 4.54: Optimized road of the random maze 2 (Start Location is 20) – Software version

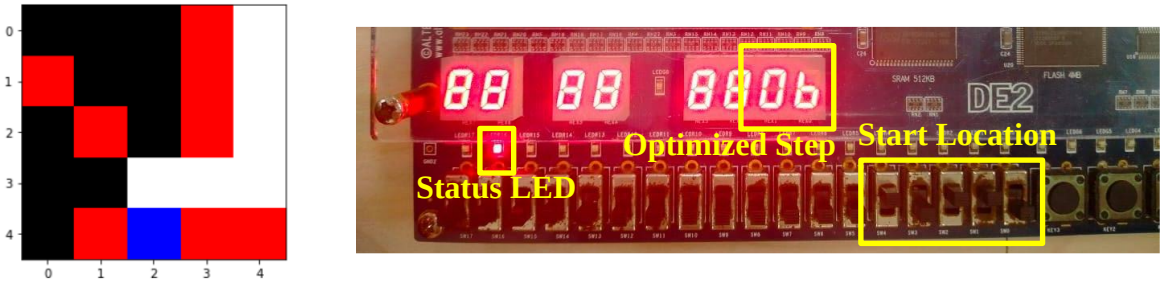


Figure 4.55: Optimized road of the random maze 2 (Start Location is 22) – Hardware version

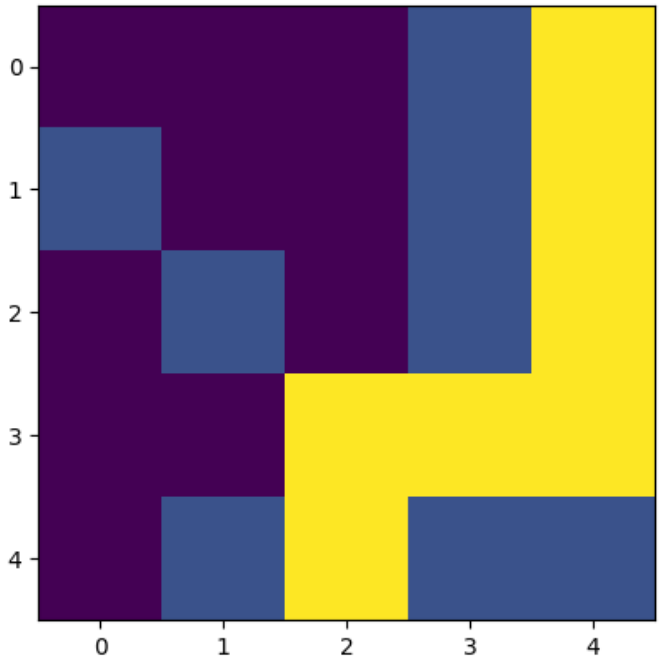


Figure 4.56: Optimized road of the random maze 2 (Start Location is 22) – Software version

4.5.2.3. Maze 3

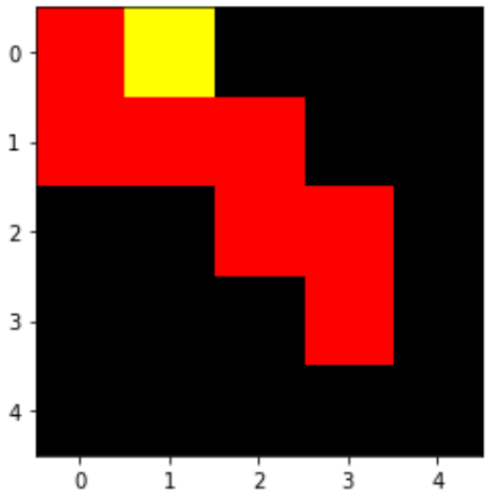


Figure 4.57: Random maze (3)

Figure 4.58: Optimized road of the random maze 3 (Start Location is 10) – Hardware version

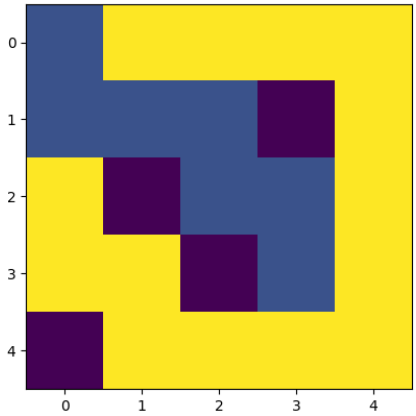


Figure 4.59: Optimized road of the random maze 3 (Start Location is 10) – Software version

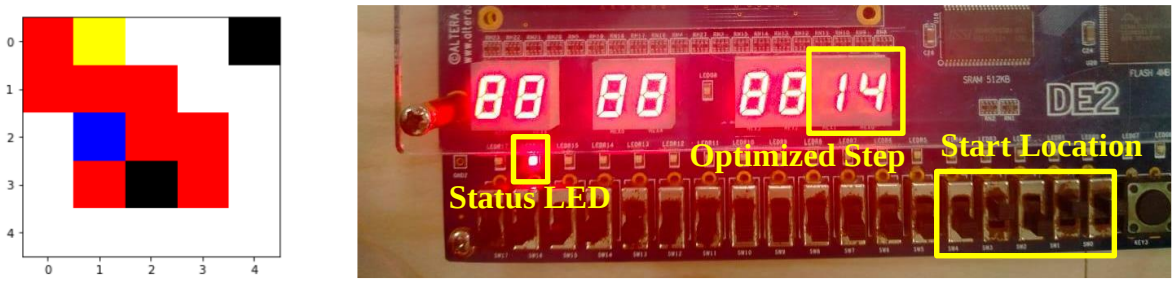


Figure 4.60: Optimized road of the random maze 3 (Start Location is 11) – Hardware version

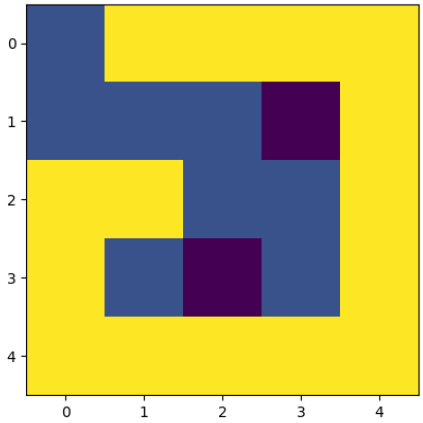


Figure 4.61: Optimized road of the random maze 3 (Start Location is 11) – Software version

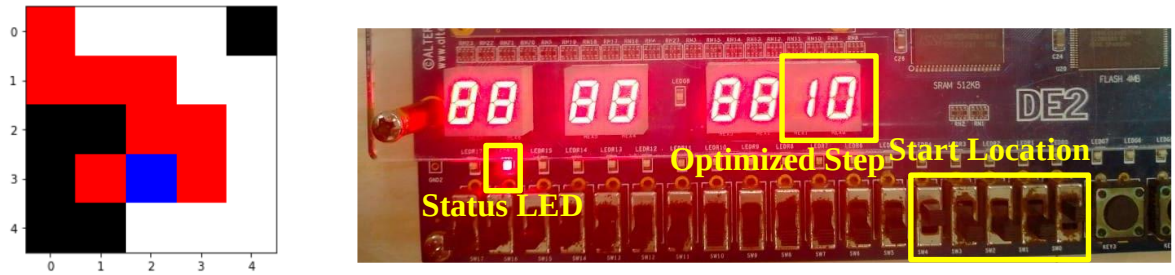


Figure 4.62: Optimized road of the random maze 3 (Start Location is 17) – Hardware version

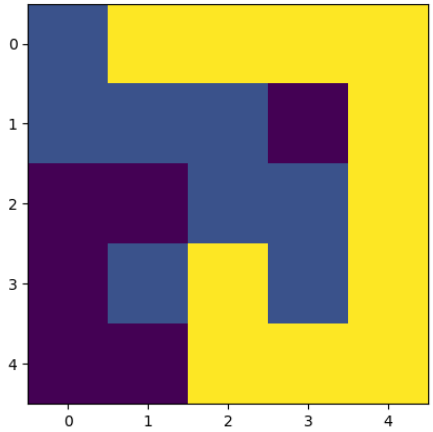


Figure 4.63: Optimized road of the random maze 3 (Start Location is 17) – Software version

4.5.3. A lot of tests

In this section, we generate about 2000 random mazes. Then we will run on the Hardware and Software parallel. After 2000 test cases, we compare the optimized step of 2 versions. If they are different, we will write it into the *output.txt* which is attached to the project folder. Figure 4.64 and Figure 4.65 is some example same and different cases respectively.

```
                        HW_ROAD                          HT   SW_ROAD                          ST
Same step count     0: 1_2_3_4_9                         5   1_2_3_4_9                         5
Same step count     1: 5_6_11_12_13_18_23_22             8   5_6_11_12_13_18_23_22             8
Same step count     2: 5_6_7_12_17_22_23                 7   5_6_11_16_21_22_23                7
Same step count     3: 5_6_11_16                         4   5_6_11_16                         4
Same step count     4: 1_2_3_4                           4   1_2_3_4                           4
Same step count     5: 1_6_7_12_13_18_23_22_21_16       10   1_6_7_12_13_18_23_22_21_16       10
Same step count     6: 5_6_7_8_9_14_19_18_17             9   5_6_7_8_9_14_19_18_17             9
Same step count     7: 1_2_7_12_17_16                    6   1_2_7_12_17_16                    6
Same step count     8: 1_2_7_12                          4   1_6_11_12                         4
Same step count     9: 5_6_11_12                         4   5_6_11_12                         4
Same step count    10: 5_10_15_20                        4   5_10_15_20                        4
Same step count    11: 5_10_11_16_17                     5   5_10_15_16_17                     5
Same step count    12: 5_6_7_12_13_18_23_24              8   5_6_7_12_13_18_23_24              8
Same step count    13: 5_6_11_12_13                      5   5_10_11_12_13                     5
```

Figure 4.64: The comparation result between Hardware and Software version – Same cases

	HW_ROAD	HT	SW_ROAD	ST	
Different case	66: 5_10_15_16_17_18_19	7	5_10_15_16_17_18_23_24_19	9	Hardware Win
Different case	92: 1_6_7_8_9	5	1_6_11_12_13_14_9	7	Hardware Win
Different case	93: 5_6_11_16_17	5	5_6_11_16_21_22_17	7	Hardware Win
Different case	153: 5_10_15_16_17_12	6	5_6_7_12	4	Software Win
Different case	194: 1_2_7_12_13_14	6	1_2_7_12_17_18_19_14	8	Hardware Win
Different case	199: 5_10_15_16	4	5_10_15_20_21_16	6	Hardware Win
Different case	202: 5_10_11_12	4	5_10_11_16_17_12	6	Hardware Win
Different case	239: 5_10_11_12_13_18	6	5_10_15_20_21_22_23_18	8	Hardware Win
Different case	376: 1_2_3_8_13_14_19_18	8	1_6_7_12_13_18	6	Software Win
Different case	405: 5_10_15_16_17_18_19	7	5_10_15_16_21_22_23_24_19	9	Hardware Win
Different case	463: 1_2_7_12_13_14_19	7	1_6_7_12_17_18_23_24_19	9	Hardware Win
Different case	469: 1_2_7_8_13_14	6	1_2_7_8_13_18_19_14	8	Hardware Win
Different case	535: 1_6_11_12	4	1_6_11_16_17_12	6	Hardware Win
Different case	551: 5_6_7_8_13_18_23_22	8	5_6_7_12_17_22	6	Software Win
Different case	627: 5_10_15_16	4	5_10_15_20_21_16	6	Hardware Win
Different case	657: 5_6_11_12_13_18_19	7	1_6_11_12_13_18_23_24_19	9	Hardware Win
Different case	685: 5_10_15_16_17	5	5_10_15_20_21_22_17	7	Hardware Win

Figure 4.65: The comparation result between Hardware and Software version – Different cases

In 2000 case was tested, there are 62 cases that the hardware's result and software's result are different from optimized path. After analyzing, we figure out that the difference come from the difference between *random actions*. However, the rate of the difference is only 3.1%.

In conclusion, Hardware version maybe give us a better result than Software version.

Chapter 5. Conclusion

Our team success in the implement of the Dyna-Q accelerator which can solve the 5x5 maze problem with the lowest hardware utilization by reducing the Dyna-Q's equation in equation 1.1 and modify the Dyna-Q's remembering mechanism. In addition, to show the advantage of Dyna-Q than Q-Learning, we also build a Q-Learning accelerator.

5.1. Appeal points and originality

Before start implement this accelerator, our team spend to find the best parameter for this algorithm which we believed is reduce the hardware of this accelerator. There are *alpha* (learning rate) and *gamma* (discount) in the equal equation 1.1. By doing this we have equation 5.1.

$$Q(S, A)_{new} = Q(S, A) + R + (\max Q(S', A) \gg 1) - Q(S,A) \qquad (5.1)$$

That's why we have an equation have no multiplication and division. That why's our speed is improve and the hardware utilization is reduce.

Beside we also implement another verification way which we bring our team a lot of benefit for testing and more fun by showing the road with optimized steps on our computer screen. In normal way, most candidates will only show the result on the waveform or display the result in log window then check the result. In the difference way, we load the circuit on the KIT, the KIT will train the hero to find the best way. When the training is done, a status LED will be on to notice us that the training process is done and show the optimized step in LED 7 SEG. Then, we just need to press the KEY to enable the KIT send the result to our computer through UART. With Python, we receive that data and show it in the screen by colorful way which is easy to check the result and more fun.

5.2. Limitation

This accelerator is easy to implement with high performance, but hard to extend. It means, with the bigger maze, for example 6x6, 7x7, 8x8 or even 100x100, Dyna-Q is a bad choice because it needs a Q-Table which increases when the size of maze increase. In this project, with 5x5 maze, Q-Table take 5x5x4 = 100 registers that why's the maze with unlimited size is impossible. With bigger size, we need to apply more higher algorithm of Reinforcement Learning.

In addition, the execution is too long. We also can improve this algorithm to reduce the execution time in the next version.

5.3. Recommendation

To deal with larger maze problem, for example 100x100, 200x200, etc, we need to apply a Neural Network to solve any maze size as Figure 5.1.

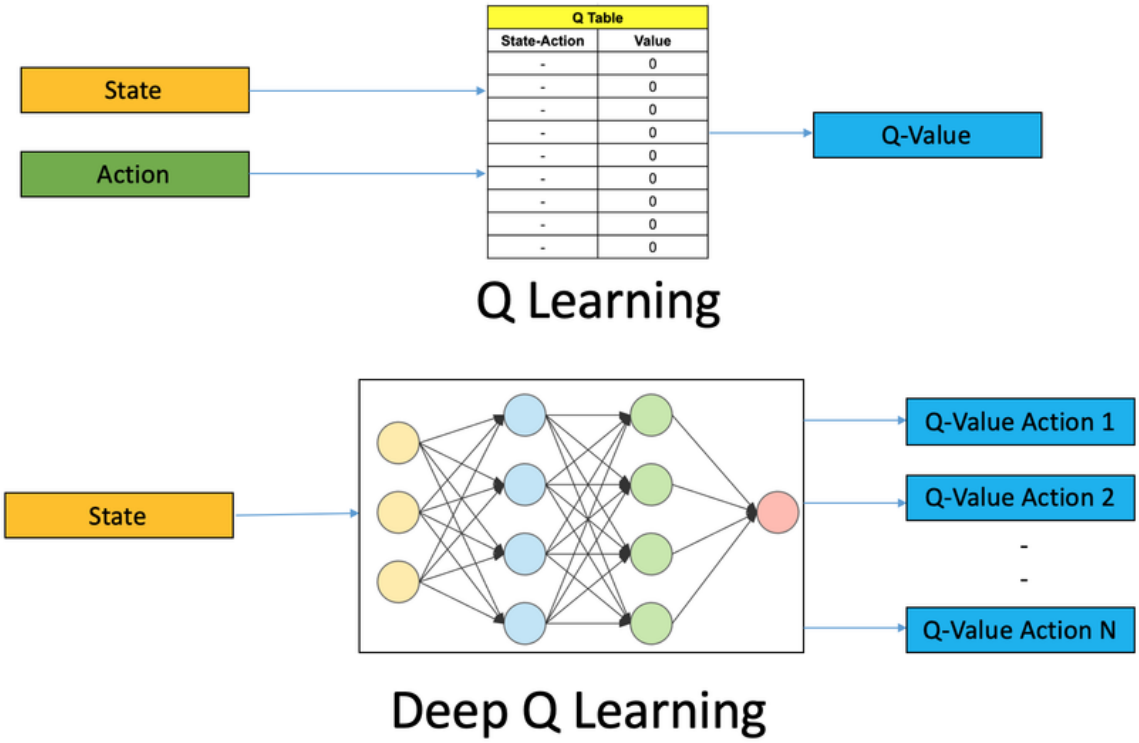


Figure 5.1: Q-Learning vs. Deep Q Learning.

In the Figure 5.1, Q-learning as well as Dyna-Q algorithm need a Q-Table to store result, so it is not suitable for large maze. Deep Q-Learning is not like that. It only needs to store a network which is unchanged with any map's size.