

# Design Hardware Accelerator For Dyna-Q Algorithm To Solve 5x5 Maze Problem

Thien Le<sup>1</sup>   Hieu Nguyen<sup>2</sup>   Phuc Nguyen<sup>1</sup>   Duong Tran<sup>1 3</sup>

<sup>1</sup>Faculty of Computer Engineering  
University of Information Technology - VNUHCM

<sup>2</sup>Faculty of Computer Science  
University of Information Technology - VNUHCM

<sup>3</sup>Instructor

LSI Design Contest In OKINAWA 2021, March 2021



# Agenda

- 1 Introduction
- 2 Original Dyna-Q and Our modified algorithm
- 3 Implementation
- 4 Experiment



# 1. Introduction

## Reinforcement Learning (RL)

- A computational method of understanding and automating goal-directed learning and decision making.

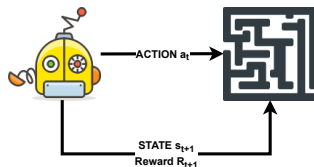


Figure: Agent Environment



# 1. Introduction

## Reinforcement Learning (RL)

- A computational method of understanding and automating goal-directed learning and decision making.
- Differs from the other kinds of learning by learning through trial and error (through a evaluating signal).

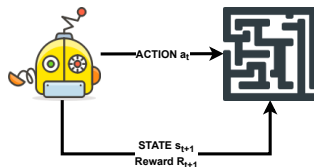


Figure: Agent Environment



# 1. Introduction

## Reinforcement Learning (RL)

- A computational method of understanding and automating goal-directed learning and decision making.
- Differs from the other kinds of learning by learning through trial and error (through a evaluating signal).
- Don't need correct label to train model.

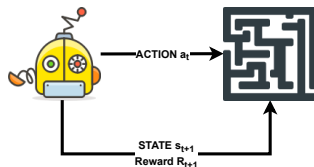


Figure: Agent Environment



## 2.1. Original Dyna-Q, Pseudo-code

**Input:**  $\alpha$  (Learning Rate),  $\gamma$  (discount factor),  $n$  ( of planning steps)

**Output:** Q-table

**Initialization:** Q-table  $Q(s,a)$ , Memory Buffer  $Model(s,a)$  for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}$ ;  $s \leftarrow$  starting state

While **not terminal state**:

    Choose execute action  $a$  with  $\epsilon - greedy(S, Q)$

    Observe reward  $r$  and next state  $s'$

$Q(S,A) \leftarrow Q(S,A) + \alpha * [r + \gamma * \max_a Q(s', a) - Q(s, a)]$

$Model(s,a) \leftarrow r, s'$

**Do  $n$  times** (planning steps):

        Randomly choose state, action  $s, a$  that exists in  $Model$

$r, s' \leftarrow Model(s, a)$

$Q(S,A) \leftarrow Q(S,A) + \alpha * [r + \gamma * \max_a Q(s', a) - Q(s, a)]$



## 2.2. Parameter Selection

$(\alpha, \gamma, \epsilon)$	Optimal	Solution (Avg.)	training steps (Avg.)
(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, 0.5, 0.05)	10	10.0	17.91
(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

The best-found parameter for Dyna-Q is 1.0 ( $\alpha$ ), 0.5 ( $\gamma$ ).



## 2.3. Modified Dyna-Q, Pseudo-code

**Initialization:** Q-table  $Q(s,a)$ ; Memory Buffer  $Model(s,a)$  for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}$ ;  $\epsilon \leftarrow 120$ ;  $s \leftarrow$  starting state;

While  $\epsilon > 0$ , do:

    If ( $random\_epsilon \leq \epsilon_t hreshold$ ) :

        action = random\_action

    Else:

        action = max\_direction

    Observe reward  $r$  and next state  $s'$

$Q(S,A) \leftarrow Q(S,A) + [R + \max_a Q(S',a) - Q(S,A)]$

$Model(s,a) \leftarrow r, s'$

    If  $\max_a Q(S',a) > 0$  Break;

$r, s' \leftarrow Model(p_s, p_a)$

$Q(S,A) \leftarrow Q(S,A) + [R + \max_a Q(S',a) - Q(S,A)]$

    If terminal state: Break;

    Else:  $\epsilon \leftarrow \epsilon - 10$





### 3. Implementation

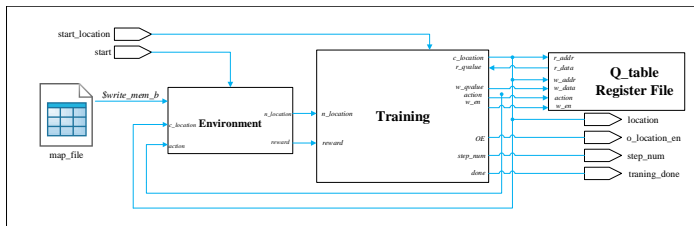


Figure: The system design of the machine

Our Design has 3 main parts

- Environment block.

### 3. Implementation

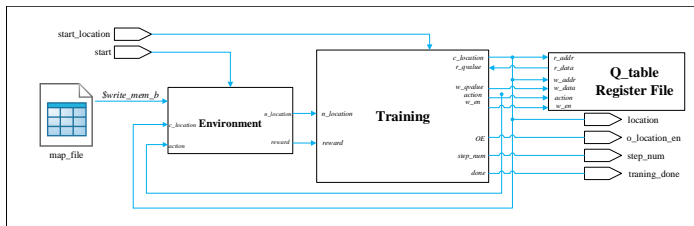


Figure: The system design of the machine

#### Our Design has 3 main parts

- Environment block.
- Training block.

### 3. Implementation

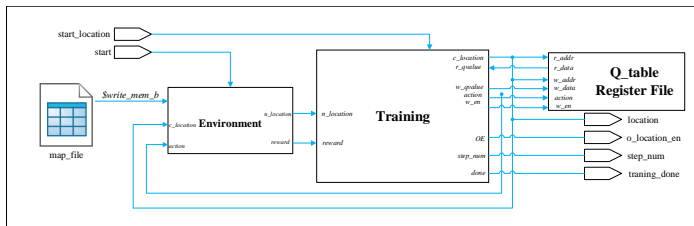


Figure: The system design of the machine

#### Our Design has 3 main parts

- Environment block.
- Training block.
- Q-table Block.

## 3.1. Environment block

### Environment Block

- 25 2-bit-registers: Storing the map from user.

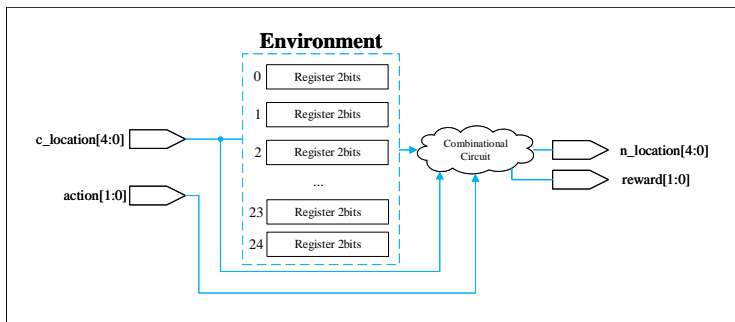


Figure: The design of Environment block



## 3.1. Environment block

### Environment Block

- 25 2-bit-registers: Storing the map from user.
- A combination circuit: Decode reward and next location.

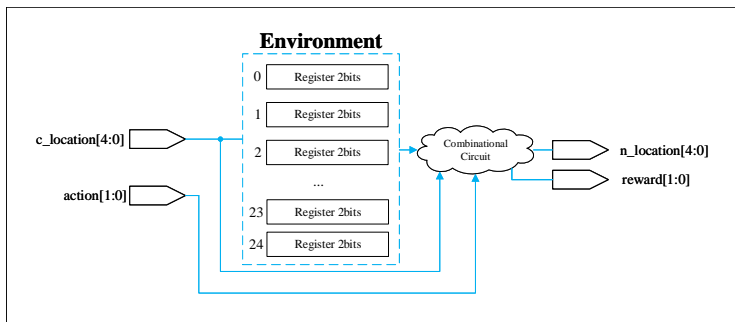


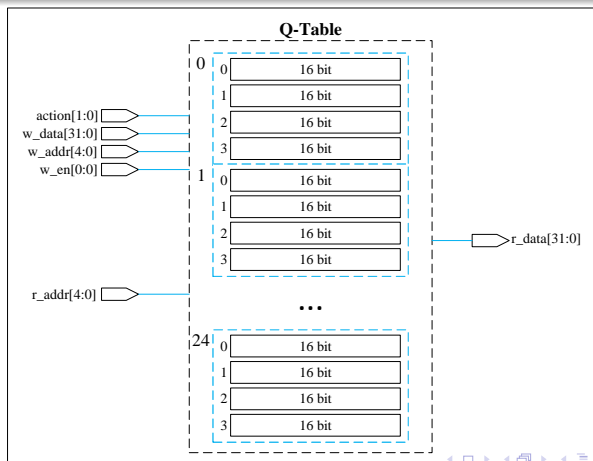
Figure: The design of Environment block



## 3.2. Q-Table block

### Q-Table

A register file containing 25x4 16-bit-registers associating with 25 squares and 4 action for each square.



### 3.3. Training block

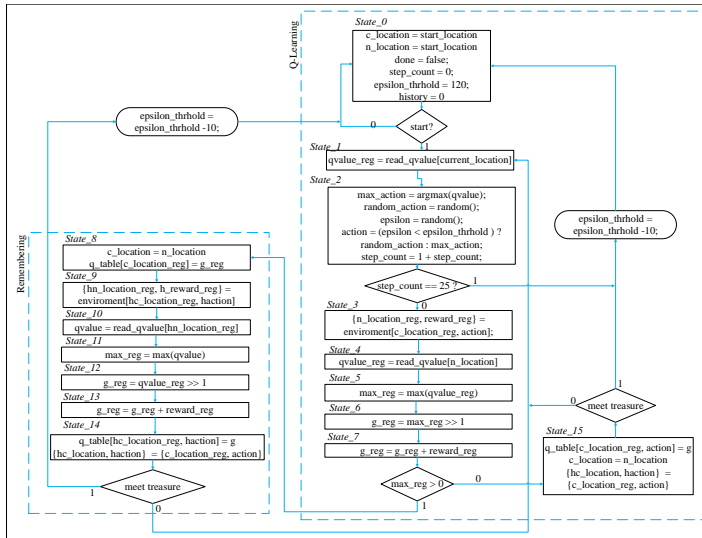


Figure: Algorithm Finite State Machine



### 3.3. Training block

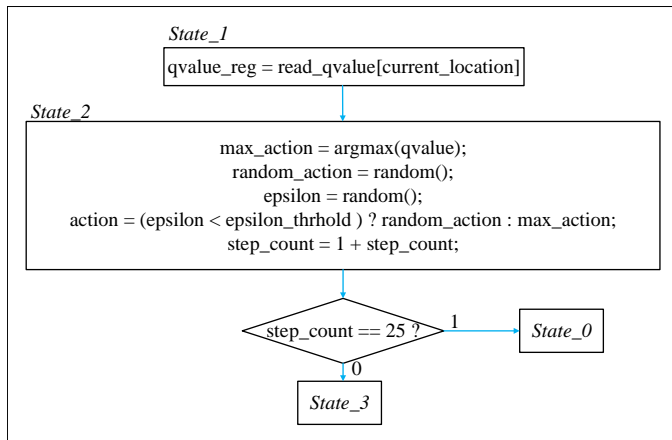


Figure: Algorithm Finite State Machine





### 3.3. Training block

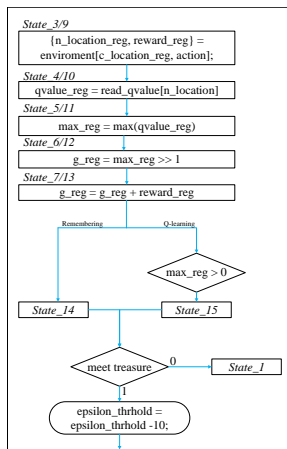


Figure: Algorithm Finite State Machine



### 3.3. Training block

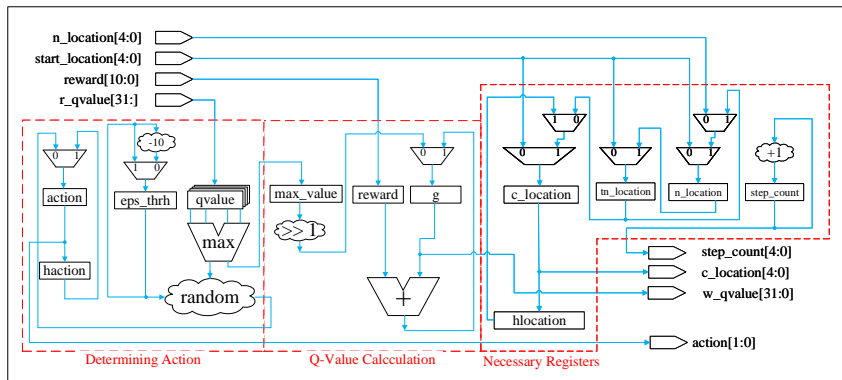


Figure: Datapath

## 3.4 Training block

### Other calculation blocks

- **Max block:** Determine which is max value in Q-value of four direction.
- **Shift-right-1 block:** Used for operating  $\ast 0.5$  operation.
- **Random Generator:** Used for generating action depend on epsilon threshold and direction of the max value.
- **Adder block:** 16-bit-adder.



## 4. Experiment

This section contains:

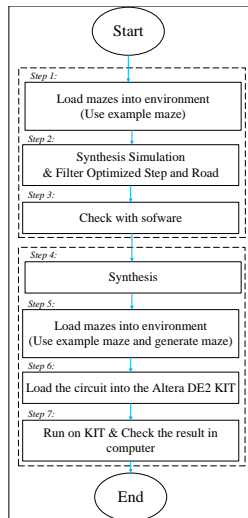
- Verification plan.
- Simulation result.
- Testing on FPGA board.
- Compare modified Dyna-Q and Q-learning.
- Compare the software and hardware version of Dyna-Q.



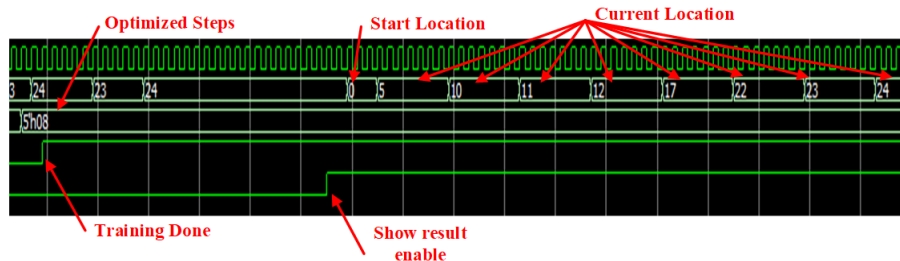
## 4.1. Verification Plan


### Our verification plan has 7 steps

- Step 1: Load mazes into environment block.
- Step 2: Simulation and filter out the optimized step and our solution.
- Step 3: Check these result with software version.
- Step 4,5,6: Synthesize, then repeat step 1 and install this accelerator to FPGA board.
- Step 7: Load the result to a computer for further observation. .



## 4.1. Simulation



S0	S1	S2	S3	S4
				
S5	S6	S7	S8	S9
				
S10	S11	S12	S13	S14
				
S15	S16	S17	S18	S19
				
S20	S21	S22	S23	S24
				

### Simulation Result

The optimized step and its solution match with result from LSI Design Contest 2021.



## 4.2. Testing on KIT

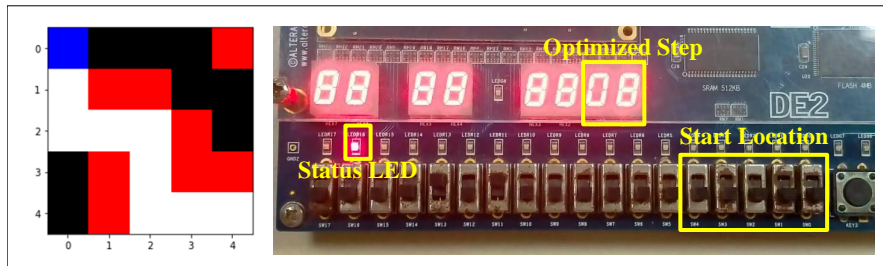


Figure: Testing on FPGA board

- We install our design into DE2 Education and Development board.
- This is our solution from example maze from Contest Committee.
- The FPGA show the optimized steps and send the solution to a computer through UART for observing.



## 4.2. Compare modified Dyna-Q and Q-learning

**Table:** Comparison between Dyna-Q and Q-Learning's hardware utilization in DE-2 Education and Development Board

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

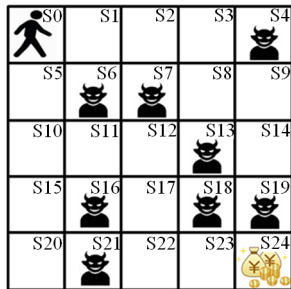
- The hardware utilization between Dyna-Q and Qlearning's is not different too much.





## 4.2. Compare modified Dyna-Q and Q-learning

**Table:** Comparison between Dyna-Q and Q-Learning execution time



Start at	Dyna-Q(ns)	Q-Learning(ns)
<b>0</b>	<b>4877.90</b>	<b>6138.30</b>
1	10328.70	8497.30
<b>2</b>	<b>15606.70</b>	<b>22565.50</b>
<b>3</b>	<b>23600.50</b>	<b>28332.70</b>
<b>8</b>	<b>79861.50</b>	<b>80666.50</b>
<b>9</b>	<b>88760.70</b>	<b>110173.50</b>
<b>14</b>	<b>403056.70</b>	<b>475482.50</b>
<b>15</b>	<b>1805.10</b>	<b>2059.70</b>
20	1607.30	1407.90

With the frequency equal 200ps in simulation, the execution time of Dyna-Q is less than Q-learning significantly.



## 4.2. Compare the software and hardware version of Dyna-Q.

**Table:** Comparison between Dyna-Q Algorithm in software and hardware version

Total case	10000 (100%)
The number of same cases	1661 (16.61%)
The number of different cases	8339 (83.39%)
Hardware better cases	5872 (58.72%)
Software better cases	2467 (24.67%)
Fail cases	0 (0%)

- We generate 10000 mazes randomly for verification and comparing 2 versions.
- In 10000 random maze, hardware version with some modifies gives us better solution (58.72%) than software version which is implemented standard algorithm with Python language.



## 5. Conclusion

### Our contribution

- A logical circuit design of modified Dyna-Q algorithm
- Our experiments show that our design performs better than Q-Learning agent, applying in the same problem

### Limitations

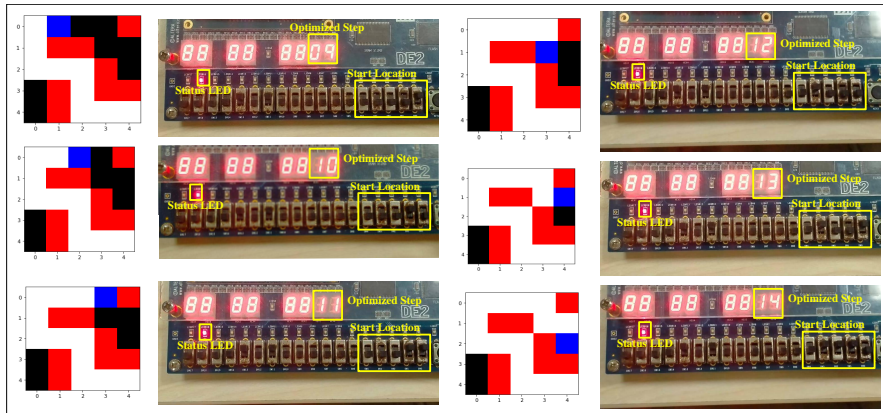
- Dyna-Q may perform well with small problems such as 5x5, but when maze grows too large (50x50, 100x100), our agent becomes unstable and takes long time to find optimal solution.
- The same problem may happen when we expand our state space by introducing adversarial elements such as enemies into our maze.



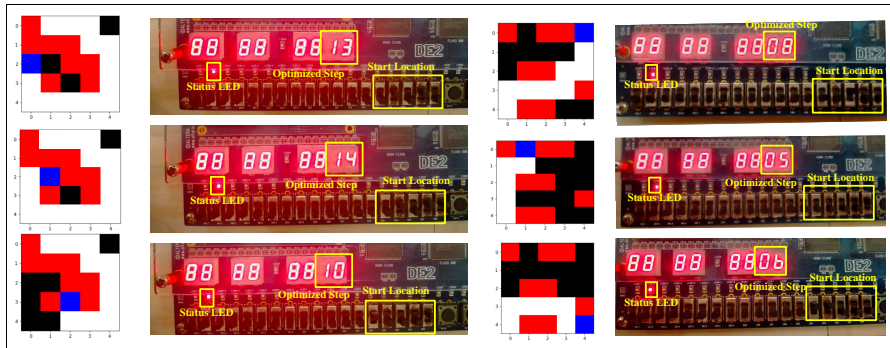
# Thanks for watching !!!



# Example: FPGA Board



# Example: FPGA Board



# Example: FPGA Board

