Architecture Design of Q-Learning Accelerator for Intelligent Traffic Control System

Nana Sutisna, Zulfikar N. Arifuzzaki, Infall Syafalni, Rahmat Mulyawan, Trio Adiono
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
z.arifuzzaki@gmail.com

***Abstract*—Traffic congestion has been a major problem for almost every city around the globe. Most of those cities get their traffic even worse in rush hour. While the existing traffic light is operated using fixed-time intervals, many researchers have been developing an intelligent traffic light that works in real time based on a machine learning algorithm. However, it takes a lot of time to process the algorithm as the traffic is getting worse. In this paper, we propose a Q-Learning Accelerator Architecture for Intelligent Traffic Light Controller. It will allow a faster learning time computation. Furthermore, the controller will allow traffic lights to work adaptively in real time. This architecture is compatible with a 4-lane crossroad using 4 possible traffic light signals. Each signal represents a green light for one road, and red light for the others. This architecture has been successfully implemented on Zynq-7000 System-on-Chip by Xilinx.**

***Index Terms*—Q-Learning, Hardware Accelerator, System-on-Chip**

I. Introduction

Reinforcement Learning (RL) is an artificial intelligent formalism that allows an agent to learn from the interaction with the environment where it is inserted [1]. Reinforcement learning problems involve learning what to do-how to map situations to actions to maximize a numerical reward signal [1]. One of the most important breakthroughs in reinforcement learning was the development of an off-policy temporal different control algorithm known as Q-Learning [2].

Q-Learning is arguably one of the most applied representative reinforcement learning approaches and one of the off-policy strategies [3]. The development of Q-Learning enables the improvement of so many cases in real life, one of them being traffic light control systems. Most traffic light operations today are controlled by a fixed-time interval that is initiated when the traffic light is installed. This mechanism does not allow the traffic light to have the ability to adapt. The emerging Q-Learning would dramatically change the state of operation of traffic lights so that they could output the most effective signal to control the traffic in real time.

In [4], it has been successfully implemented a traffic light controller using reinforcement learning. The results have been attached and it shows that the algorithm works better than any other algorithm that has been developed.

Identify applicable funding agency here. If none, delete this.

However, Q-Learning is a very slow method that requires a long training time and high computing resources to obtain a practical convergence strategy [5]. It takes even longer time to train and higher computer resources if it is implemented in a complex traffic environment.

As software-based implementation would urge each algorithm instruction to be processed one by one. Furthermore, the q-learning algorithm cannot be processed in parallel. This contributes to a longer training time for the q-learning agent to introduce an optimum action for each state.

In this paper, we proposed a hardware accelerator architecture that would make the processing time faster. The previous related works are discussed in chapter 2 as the foundation for making the architecture. Chapter 3 describes the proposed hardware architecture. The result and experimental analysis are shown in chapter 4. The conclusion of this paper is summarized in chapter 5.

II. Related Works

Q-Learning [2] is a form of model-free reinforcement learning [6]. Since its convergence has been proved as shown in [7] for any agent's policy, it is widely used for a wide range of applications. Each Q-value in Q-matrix is then updated using equation [8].

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma max_A Q(s_{t+1}), A) \tag{1}$$

There has been a lot of work related to Q-Learning. It is shown in [3] that discussed classification and application of Q-learning comprehensively. They classify Q-Learning algorithms as single-agent and multi-agent algorithms.

In [9] has implemented a Q-Learning algorithm for FPGA. Their architecture has been simulated and synthesized for different scenarios and the number of states and actions were varied. Since the architecture takes a lot of area in a chip, it is then optimized in [10] by employing approximated multipliers instead of full multipliers. This work results in a smaller amount of hardware resources, faster computation, and dissipates less power.

In the case of utilizing Q-Learning for controlling traffic lights, [11] discussed optimizing single intersection performance using the Q-learning algorithm. It has been proven that their architecture decreases the average waiting vehicle by 17

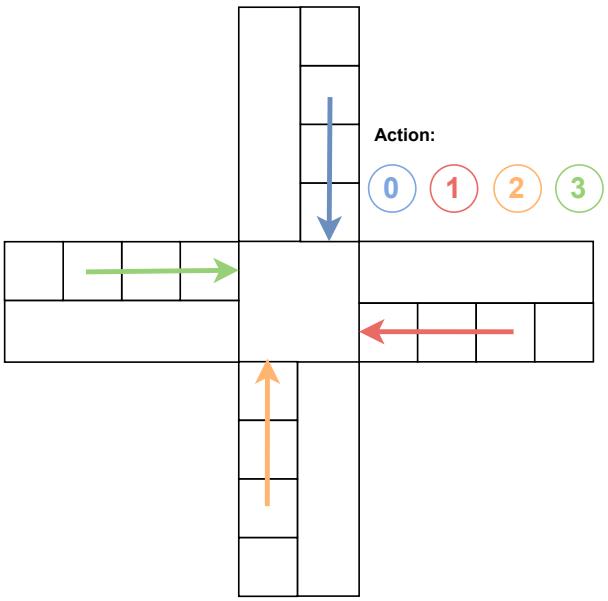

Fig. 1. Environment modelization and proposed action design

III. Proposed Architecture

A. Environment Modelization

The environment of this system is a 4-arm junction with traffic congestion. The length of the traffic congestion on each road is then characterized into 4 different levels as shown by figure 1.

We use a 4-segmentation because the system will only be compatible with a 4-arm junction. The 4-segmentation configuration is the minimum number of segmentation that allows us to determine which road with the maximum and minimum traffic congestion if the traffic congestion on each road is different.

Since each road is characterized into one single lane, then we design four possible actions that are shown in figure 1.

B. Proposed Architecture Overview

Figure 2 shows the proposed SoC architecture for the Q-Learning algorithm to control traffic lights. It is designed to be implemented in the Zynq-700 SoC chip by Xilinx. The chip consists of Processing System (which is an ARM Cortex A-9) and Programmable Logic (which is an FPGA). The PS part is utilized as an interface for the user to input parameters. The parameters will be explained in the following section below. On the other hand, PL is responsible for operating the whole Q-Learning process.

The Q-Learning processes could be broken down into two sequentially sub-processes, i.e.: the learning process and the adaptation process.

The learning process is when the agent explores and exploits its environment to determine the most effective action for each state where the agent is currently at. To carry this process out, the system needs some user inputs, i.e.: the seeds to feed linear shift feedback register (LSFR), and Q-learning parameters. The set data called Q-Matrix is created after the learning process is finished. This Q-Matrix represents the whole state-action pairs that describe the agent's final behavior after learning.

Meanwhile, the adaptation process is the process when the agent reads the Q-Matrix that has been produced through the learning process. The reading would output the chosen action based on state-action pairs written in Q-Matrix.

For some cases, there is a finite probability that the agent did not visit some states when learning. It results in a zero row Q-value for the entire action in the state-action pair in Q-Matrix. It is considered a learning bug. To overcome the bug, the agent must process brief-short learning during adaptation to decide the best action to be taken.

The architecture is shown in Figure 2 composed of four basic parts, i.e.: agent, environment, memory dan control unit. Those parts are explained in the following section:

C. Agent Modelization

This part consists of two modules, i.e.: the policy generator, which decides action; and the Q-Learning accelerator, which updates the Q-Value for the current state-action.

1) Policy Generator: The policy generator module (PG) represents the agent's behavior in deciding the next chosen action. It holds the mathematical equation below:

$$a_{t+1} = \begin{cases} max(Q(s_t, A)), & \text{if } x < \tau \\ \text{random}, & \text{otherwise} \end{cases} \quad (2)$$

where τ is a dynamic threshold for deciding whether the action is explorative (random action) or exploitative (greed action).

$$\tau = M_{episode} - episode \quad (3)$$

2) Q Learning Accelerator: Elements that build up the Q-Matrix are called Q-value $(Q(s_t, a_t))$, which is an estimation of how good it is to take the action a_t at the state s_t. At first, the Q-Matrix is initiated by a zero matrix. During the learning process, the agent must fill up the entire Q-Matrix with a finite Q-value. It is executed by a Q-learning accelerator module (QA) through updating mechanism that updates the Q-Value one at a time based on equation (1).

D. Environment Modelization

1) Environment Generator: To make the agent explores the entire possible states, it is necessary to randomize the starting environment as well as the debit of incoming vehicles. This randomization is carried out by the environment generator (EG) module. The internal structure of EG consists of two 64-bit LSFRs, each of which will output four 16-bit values. The first LSFR is used to randomize the starting environment L_0, while the other randomizes the incoming vehicle debit in_t.

2) State Decider: The state decider module (SD) is basically a converter that transforms the traffic jam information into the state s_t a form that is recognized by the agent. Inside SD, the state s_t is processed further to generate the goal signal. The goal signal will be responsible for incrementing the learning step that is being counted be the control unit

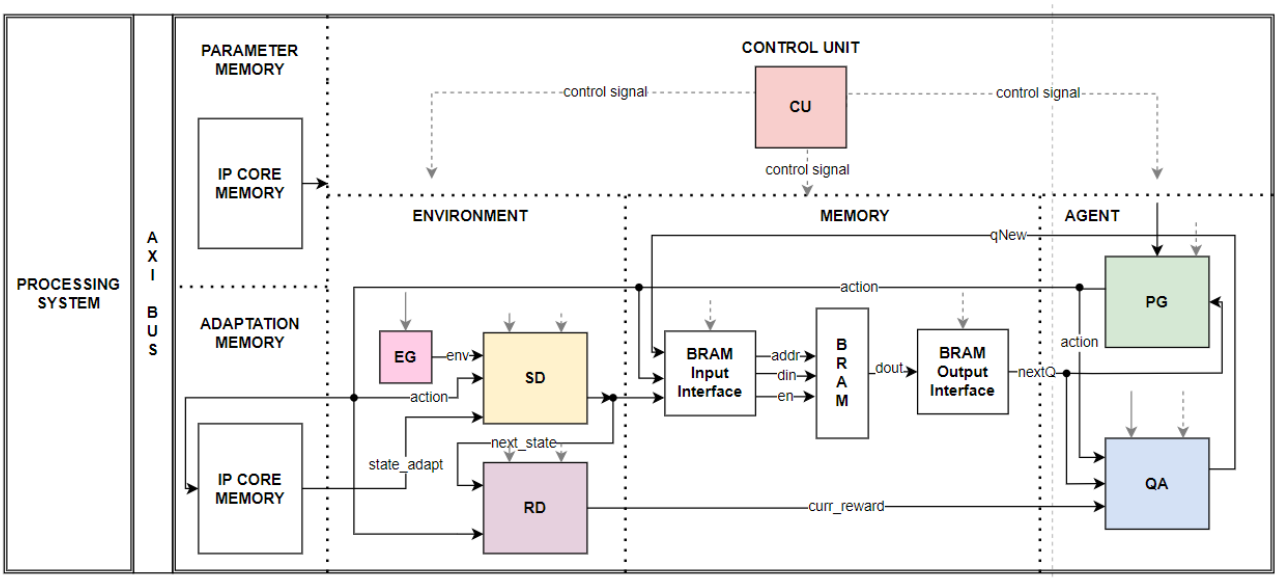

Fig. 2. System architecture

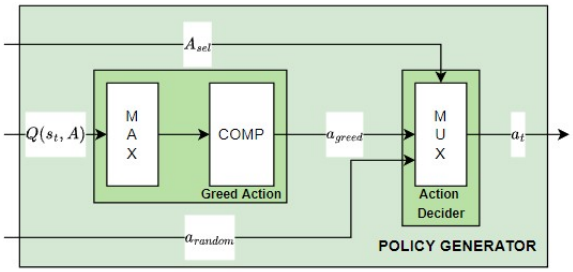

Fig. 3. Sub-block of policy generator

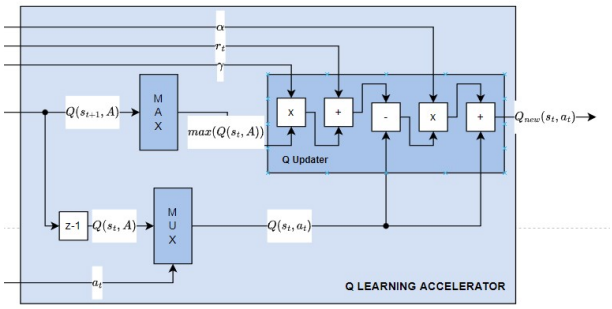

Fig. 4. Sub-block of q accelerator

module (the control unit module will be explained further in the section below).

At each learning step, SD (especially the State Calculator sub-module) needs to update the traffic jam L_t based on the randomly generated incoming vehicle flow debit $debit_i n_t$ and the previous traffic jam $L_(t-1)$. This environment updating mechanism is followed simultaneously by determining the corresponding state s_t.

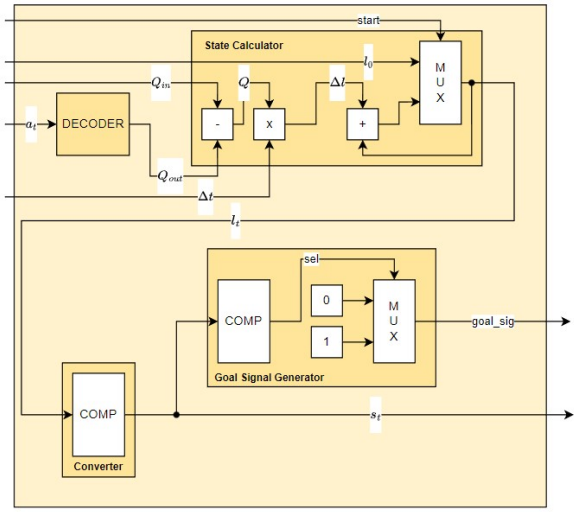

Fig. 5. Sub-block of state decider

3) Reward Decider: The reward decider module (RD) processes state s_t and action a_t to determine the reward for the agent $r_(t+1)$. Its architecture is shown in Figure 6. There are three possible rewards, i.e.:

- R_0 is chosen if the agent gives a green signal to the lane with the lowest congestion level
- R_1 is chosen if the agent gives a green signal to the lane with the congestion level lying between the highest and the lowest
- R_2 is chosen if the agent gives a green signal to the lane with the highest congestion level

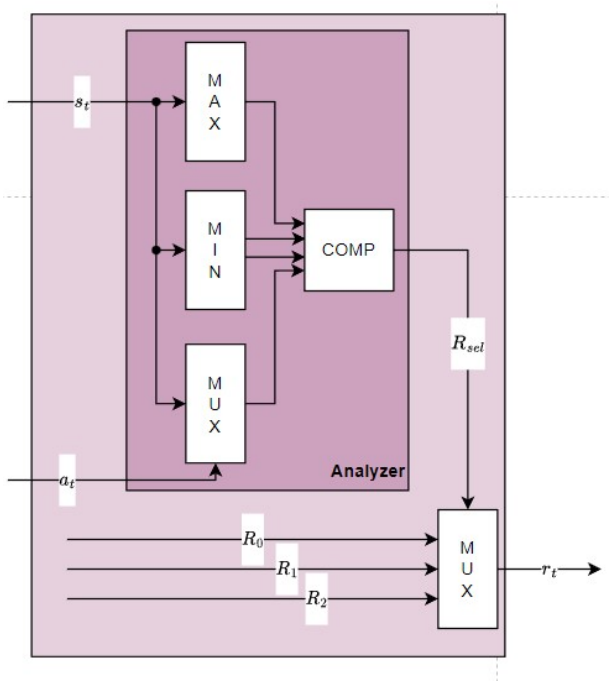

Fig. 6. Sub-block of reward decider

E. Memory Design

1) Action RAM Design: Q-Matrix is a matrix, where is the number of states and is the number of possible actions. The architecture is shown in Error! Reference source not found. operates with 4096 states and 4 actions. So, the size of the matrix that needs to be saved in memory is .

The memory architecture in Figure 7 was developed to get as less as possible RAMs utilized while maintaining the most efficient amount of parallelization to decrease the Q-learning processing time and system complexity.

The requirement above meets if the action-parallel RAM configuration is chosen. The configuration parallelizes Q-Matrix in terms of action, therefore the system needs only 4 RAMs to save the whole Q-Matrix.

BRAM Memory Generator by Xilinx is utilized as a basic building block of memory design. However, the BRAM supports up to only two ports to access the memory. Meanwhile, the learning process requires both two ports, each used to perform read and write operations. There is no remaining port for the PS to access the memory. Consequently, it is necessary to have 2 sets of memory, which contain the same data but are connected to different subjects. This is the reason why there are two sets of Action RAMS, i.e.: PS-PL Action RAMs and PL-PL Action RAMs. Hence, the system needs to double-write the Q-Value into two different RAMs.

2) BRAM Interfaces: The *read address* can have the same value as the *write address*. The read address and write address that shares the same value cause data collision if both are accessing the same BRAM. The data collision causes an undefined BRAM output. Therefore, it is needed to control the enable signal for both ports of BRAM to get rid of this. BRAM input interface is a sub-module that carries out the control to avoid data collision.

When the read address shares the same value as the write address, the BRAM input interface will control the enable signal so that the write-first operation is conducted. It means that the read operation will be shut down, allowing the write operation to BRAM. This causes BRAM to output the undesirable zero value.

Fortunately, the *write address* is the *read address* that has been delayed. It allows us to hold the previous output from the memory to replace the current memory output whenever the current output is the zero value in the cause of the write-first operation. The holding process is being conducted by the BRAM Output Interfaces sub-module.

IV. Experimental and Discussion

A. Performance Analysis

This system has been implemented in the Zynq-7000 System-on-Chip. The following data are obtained using Vivado EDA after the bitstream generation is finished. System-on-chip is an integrated circuit that contains a complete electronic system [12]. A "system" includes a microprocessor, memory, and peripherals [13]. Zynq-7000 by Xilinx products incorporates a dual-core ARM Cortex-A9 based Processing System (PS) and Xilinx Programmable Logic in a single device. This chip offers compact hardware design that supports both microprocessor-based design as well as FPGA design which supports parallel-fast hardware design.

The architecture has been successfully implemented to the Zynq-7000 System on Chip at a 100 MHz clock signal to meet the timing requirement. The summary for the timing analysis is shown in Table 3. I.

TABLE I
TIMING ANALYSIS RESULT FOR THE PROPOSED ARCHITECTURE

Timing variables	Quantity
Clock Frequency	100 MHz
Worst Negative Slack	1.153 ns
Worst Hold Slack	0.031 ns
Worst Pulse Width Slack	3.750 ns

The resource utilization used by the system is shown in table II. The combinational logic of the system is inferred as a look-up table in this design. While registers are used in sequential logic and to hold some data in a clock cycle period. The block RAM tile is used to save the entire Q-matrix from the Q-learning agent and Q-learning accelerator. The number of block RAM used is getting higher if the number of state to describe the environment is increased as well. Table 4 shows the resource utilization by the proposed architecture. Mostly, the design takes less than 20% of utilization for each resource. However, the block RAM Tile used for this design is quite high. It is caused by the high number of states that the design can handle, which is 4096 states. As the number of states goes higher, the block RAM utilization would be increasing as well.

TABLE II
RESOURCE UTILIZATION OF THE PROPOSED ARCHITECTURE

Resource	Utilization		
	Used	Available	Percentage
Slice LUTs	5200	53200	9.77%
Slice Registers	6745	106400	6.34%
F7 Muxes	125	26600	0.47%
Slice	2322	13300	17.46%
LUT as Logic	331	17400	1.90%
LUT as Memory	331	17400	1.90%
Block RAM Tile	34	140	24.29%

The resources and timing analysis above correspond to the power taken by the chip. The power analysis of the implemented system is shown in figure 7. The hardware accelerator system is implemented in a static device and it takes 0.148 W power to energize the hardware accelerator. While the dynamic device is referred to the processing system of the system-on-chip.

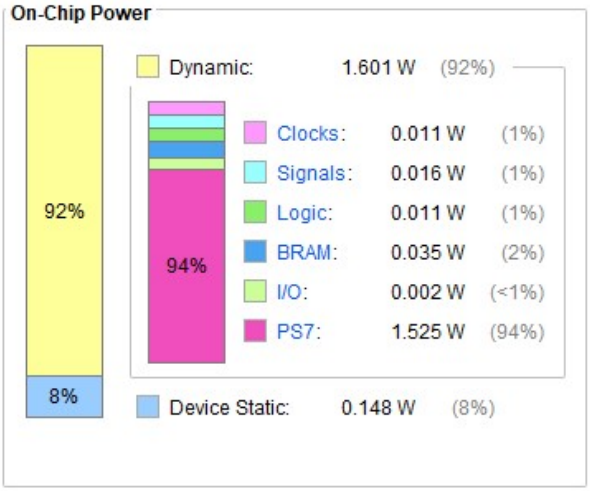

Fig. 7. Power analysis

B. Simulation Result

The simulation has been done by using python *pygame* simulation. For testing the performance, we compare our system to its fixed-time counterpart in terms of the resulting total number of waiting vehicles. The comparison is shown in figure 8, the red line indicates the total waiting vehicle if our system is utilized, and the blue line indicates the total waiting vehicle for its counterpart.

The fixed-time traffic control system results in a growth in the total number of vehicles in a junction indicating that it leads to worse traffic congestion over time. While the system that uses Q-learning shows better performance in handling traffic performance in terms of maintaining the optimum action.

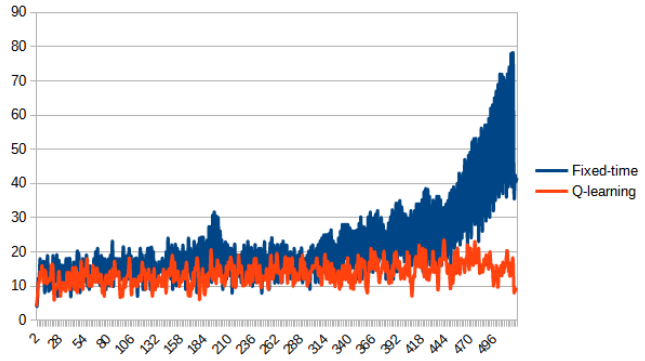

Fig. 8. Power analysis

V. CONCLUSION

The development of machine learning boosts so many improvements in every part of human life. It automates everything that was previously being done manually.

The hardware accelerator mostly takes an important role to the improvement of machine learning when it comes to a complex design. It enables to process the whole algorithm of machine learning faster than its software implementation counterpart.

This paper offers the architecture of a hardware accelerator that holds the machine learning algorithm to automate the traffic light controlling real-time. The architecture handles 32-bit data with the 4096 states and 4 possible actions. During the experiment, the architecture was implemented on the Pynq-Z1 board by Xilinx. The resource utilization and timing analysis of the design has been given.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2015.

[2] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/ chrisw/new_thesis.pdf

[3] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133 653–133 667, 2019.

[4] H. Wei, G. Zheng, H. Yao, and Z. Li, "Intellilight: A reinforcement learning approach for intelligent traffic light control," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2496–2505.

[5] D. Huang, H. Zhu, X. Lin, and L. Wang, "Application of massive parallel computation based q-learning in system control," in *2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*, 2022, pp. 1–5.

[6] C. Watkins and P. Dayan, "Q-learning. mach. learn," 1992.

[7] F. S. Melo, "Convergence of q-learning: A simple proof," *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[8] A. Gupta, P. P. Roy, and V. Dutt, "Evaluation of instance-based learning and q-learning algorithms in dynamic environments," *IEEE Access*, vol. 9, pp. 138 775–138 790, 2021.

[9] L. M. D. Da Silva, M. F. Torquato, and M. A. C. Fernandes, "Parallel implementation of reinforcement learning q-learning technique for fpga," *IEEE Access*, vol. 7, pp. 2782–2798, 2019.

[10] S. Spanò, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Matta, A. Nannarelli, and M. Re, "An efficient hardware implementation of reinforcement learning: The q-learning algorithm," *IEEE Access*, vol. 7, pp. 186 340–186 351, 2019.

[11] M. Rosyidi, S. Bismantoko, and T. Widodo, "Reinforcement learning with the classical q-learning algorithm for optimizing single intersection performance," in *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, 2020, pp. 1–5.
[12] G. S. May and S. M. Sze, *Fundamentals of Semiconductor Fabrication*. John Wiley & Sons, Inc, 2003.
[13] D. J. Greaves, "System on chip design and modelling," *University of Cambridge Computer Laboratory Lecture Notes*, p. 130, 2011.