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

Zulfikar Nima Arifuzzaki  
Electrical Engineering Department   
Bandung Institute of TechnologyBandung, 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 over time. 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 fast Q-Learning Accelerator Architecture for Intelligent Traffic Light Controller. The controller would allow traffic lights to work fast in real-time. This architecture is compatible with a 4-lane crossroad. To control the traffic, there are 4 possible traffic light signals, each signal represents a green light for one lane, and red light for the others. This architecture was successfully implemented on Zynq-7000 System-on-Chip by Xilinx.

Keywords—Q-Learning, Hardware Accelerator, System-on-Chip

# 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. [2] One of the most important breakthroughs in reinforcement learning was the development of an off-policy temporal different control algorithm known as Q-Learning. [3] Its simplest form, *one step Q-Learning*, is defined by:

|  |  |
| --- | --- |
|  | (1) |

The development of Q-Learning enables the improvement of so many cases in real life, one of them is 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 current traffic in real-time.

The implementation of the Q-Learning algorithm takes a long time to process in terms of software implementation. The more complex the environment is, the longer the processing time would be needed. To fasten it up, it is essential to build a hardware accelerator that could drastically reduce the processing time.

*System-on-chip* is an integrated circuit that contains a complete electronic system. [4] A “system” includes a microprocessor, memory, and peripherals. [5] 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 minimum specifications are:

* The system needs to work well for crossroads (there are 4 lanes at the controlled intersection)
* To minimize the error, the system must work for 32-bit data Q-value
* To increase the accuracy, the system must operate with at least 300 states

# Related work

## Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA [6]

This paper proposes a parallel fixed-point Q-Learning algorithm architecture implemented on a field-programmable gate array (FPGA) focusing on optimizing the system processing time. The system is designed to operate with states and actions. The architecture was developed to parallelize as much as possible the algorithm execution to decrease the Q-Learning processing time.

The architecture delivers a good processing time to process the Q-Learning algorithm. However, because of the maximum parallelization, it takes a high number of resource utilization.

|  |
| --- |
| Table Resource utilization taken by the architecture to operate at N = 132 and Z = 4  Table  Description automatically generated |

Table 1 shows the resource utilization taken by the architecture to operate at and . The first row shows the number of resource utilization when the system runs 24-bit fixed-point data (10 first bits in a row used for the integer part, and the fractional part takes the remaining bits). It shows that the resource is too high, especially multiplier and LUTs.

The architecture would not support 32-bit data with the number of N at least 300 and the number of action Z = 4 (as

the requirements said to our system). Therefore, it is desirable to modify the architecture to reduce resource utilization.

Diagram

Description automatically generated

## An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm [7]

This paper offers an efficient hardware architecture that implements the Q-Learning algorithm, suitable for real-time applications. Its main features are low power, high throughput, and limited hardware resources.

The Q-Matrix is stored in Z Dual-Port RAMs, named Action RAMs. Each RAM contains an entire column of the Q-Matrix. The number of memory locations corresponds to the number of states N.

Bellman equations require multiplication operations to update Q-value. This paper offers an optimized multiplier that the state of its operation is approximated using the right barrel shifter. This approach will reduce the processing time as well as resource utilization.

|  |
| --- |
| Table Implementation results for Q-Matrix with 32-bit data and Z = 4  Table  Description automatically generated |
|  |

Table 2 shows the implementation result with 32-bit data and 4 actions. Based on the last row of the table above, the design offers a more effective resource utilization that the previous paper.

# Proposed Architecture

## Proposed Architecture Overview

Figure 1 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 the process when the agent explores and exploits its environment to find out the most effective action to be taken 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 1 the composed of four basic parts, i.e.: agent, environment, memory dan control unit. Those parts are explained in the following section:

## Agent Modelization

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

### PG - Policy Generator

|  |
| --- |
| Figure Policy generator module architecture |

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

|  |  |
| --- | --- |
|  | (2) |

which the select signal is defined as

|  |  |
| --- | --- |
|  | (3) |

where:

* : is th Q-Matrix row
* : is epsilon that determines the current behavior of an agent

### QA - Q-Learning Accelerator

|  |
| --- |
| Figure Q-Leaning accelerator module architecture |

Elements that build up the Q-Matrix are called Q-value (), which is an estimation of how good it is to take the action at the state . At first, the Q-Matrix is initiated by the 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).

## Environment Modelization

### EG - 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.

|  |
| --- |
| Figure Environment generator module architecture |

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 **,** while the other randomizes the incoming vehicle debit **.**

### SD - State Decider

|  |
| --- |
| Figure State decider module architecture |

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

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

### RD - Reward Decider

The reward decider module (RD) processes state and action to determine the reward for the agent . Its architecture is shown in Figure 6.

There are three possible rewards, i.e.:

* reward 0: is chosen if the agent gives a green signal to the lane with the lowest congestion level
* reward 1: is chosen if the agent gives a green signal to the lane with the congestion level lies between the highest and the lowest
* reward 2: is chosen if the agent gives a green signal to the lane with the highest congestion level

|  |
| --- |
| Figure Reward decider block architecture |

## Memory Design

|  |
| --- |
| Diagram, schematic  Description automatically generated  Figure Memory design architecture |

Figure 7 shows the memory configuration and its interfaces. Each component above will be explained by the following sub-sections.

### Action RAM Design

Q-Matrix is an 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 term 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.

### 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.

# Experimental Analysis

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.

Table 3 Timing analysis result for the proposed architecture

|  |  |
| --- | --- |
| **Timing variable** | **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 design cannot be fastened any further because there is a critical path in Q-Learning Accelerator Module on performing the Bellman equation operation, which takes around 8.147 ns delay (1.1317 ns logic delay and 6.830 ns net delay).

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 Resource utilization by 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 | 4869 | 53200 | 9.15% |
| 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. Figure 8 shows the on-chip power for the design implementation. Figure 8 shows the architecture usesof power. The rest power was taken for Processing System and device static.

Chart

Description automatically generated

Figure On-Chip power for the design implementation

# 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, Reinforcement Learning : A Special Issue of Machine Learning on Reinforcement Learning, Norwell: Kluwer, 1992. |
| [2] | R. S. Sutton dan A. G. Barto, Reinforcement Learning : An Intorduction, Cambridge: The MIT Press, 2015. |
| [3] | C. Watkins, Learning form Delayed Rewards, Cambridge : Cambridge University, 1989. |
| [4] | G. May dan S. M. Sze, Fundamentals of Semiconductor Fabrication, Atlanta: John Wiley & Sons, Inc, 2004. |
| [5] | D. J. Greaves, System on Chip : Design and Modelling, Cambridge: Easter Term, 2011. |
| [6] | L. Da SIlva, M. Torquato dan M. Fernandes, “Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA,” IEEE, Rio Grande, 2019. |
| [7] | S. Spano, G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, D. Giardino, M. Matta, A. Nannarelli dan M. Re, “An Efficient Hardware Implementation of Reinforcement Learning : The Q-Learning Algorithm,” IEEEAccess, Rome, 2019. |