Hardware Acceleration of Deep Reinforcement Learning Application in Traffic Signal Control

Introduction

Abstract

Conventional traffic light signals are pre-programmed with a specific timing schedule regardless of the real time traffic pattern, resulting in unnecessary wasting of time on the road. Adaptive traffic signal control (ATSC) systems, by detecting streams of vehicles at an intersection, can predict the best light signal pattern and adjust traffic light accordingly to avoid traffic delays[1]. To constantly detect traffic behavior and offer effective solutions, ATSC usually combines an area of machine learning, known as reinforcement learning, to aid its learning process and decision optimization. While novel reinforcement learning algorithms utilizing deep neural networks have evolved rapidly in recent years, their performance in terms of computational speed have generally been bottlenecked by the dense computations and high power demands. Field Programmable Gate Arrays (FPGAs) are specialized computing devices that provide customizable parallel data processing while efficiently using less energy. Although FPGAs have been explored in deep learning projects, limited attention has been paid to the potential of their applications in reinforcement learning.

Deep Reinforcement learning

Reinforcement learning, an area of machine learning, is the process of analyzing the current environment and suggesting suitable actions to maximize the long-term gain. General reinforcement learning algorithms like Q-learning often fall short when the state-action pairs grow large, taking longer time to simulate and sometimes unable to converge. Thus, deep neural networks, for their efficiency in processing large amount of data, are introduced to develop a new concept referred to as Deep Reinforcement learning. The Advantage Actor Critic (A2C) algorithm is a deep reinforcement learning algorithm where the learning agent selects an action according to policy π and updates the policy parameters later based on an advantage function $A(a_t,s_t) = Q(a_t,s_t) - V(s_t)$, where $Q(a_t,s_t)$ is the action value and $V(s_t)$ is the average state value.

FPGA

Many processor-based systems have been bottlenecked by high latency and energy demand when the amount of data needed to process increases. GPU implementations of neural network accelerators have been prevalent, but recent studies show that FPGAs have many potential advantages over GPUs in various aspects including scalability, power efficiency and customized parallelism [5]. Aside from operating

independently of a CPU driver, the high specialization of FPGAs to even bit-level customization makes them especially suitable as DNN accelerators since the inherent parallelism and major advancement in recent FPGA models offer power efficient platforms to process inputs synchronously to achieve high performance.

Objective

This project aims to implement a widely used reinforcement learning algorithm called the Advantage Actor-critic (A2C) method with Field Programmable Gate Arrays (FPGA) to design a time and energy-efficient ATSC system.

Method

Software prototype (problem definition, algorithm)

This project will study the use of the deep reinforcement learning model for a four-way intersection with the action space defined as A: {NSL, NS, EWL, ES}, where NSL stands for north-south protected left turn, NS for north-south through with permissive left turn, and EW stands for the east-west direction. Phase numbering and traffic signal pairing are based on NEMA standards [1] shown in Figure 1.

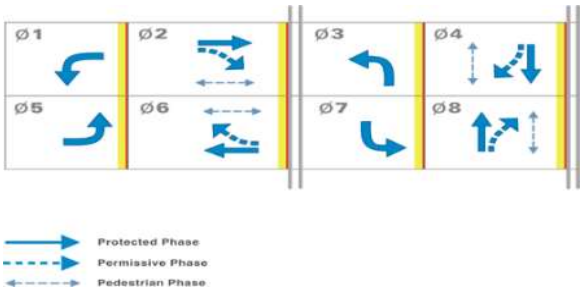


Figure 1. Phase Numbering. Diagram from [1].

The state is defined by lane occupancy passed as an 12x12 input matrix. A partial matrix for one road is shown below in Figure 2; a complete input matrix consists of four matrices, one for each road.

1	1	1	1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

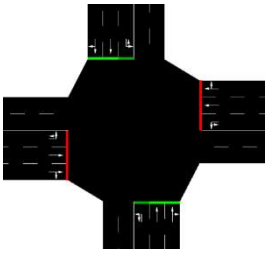


Figure 2. State representation and Intersection View

The reward in this project is defined by the difference of the total waiting time of all active vehicles at the start and the end of each simulation step *t*, where a positive reward is given to the agent for a decrease in waiting time. The action space consists of turning green light on for one of the two directions, each causes the traffic light to go through our phase from green to red as shown below.

	0(30s)	1 (5s)	2 (15s)	3 (5s)	4 (30s)	5 (5s)	6 (15s)	7 (5s)
EW-forward	G	Y	R	R	R	R	R	R
EW-left	g	g	G	Y	R	R	R	R
NS-forward	R	R	R	R	G	Y	R	R
NS-left	R	R	R	R	g	g	G	Y

*g = yield right of way on green light

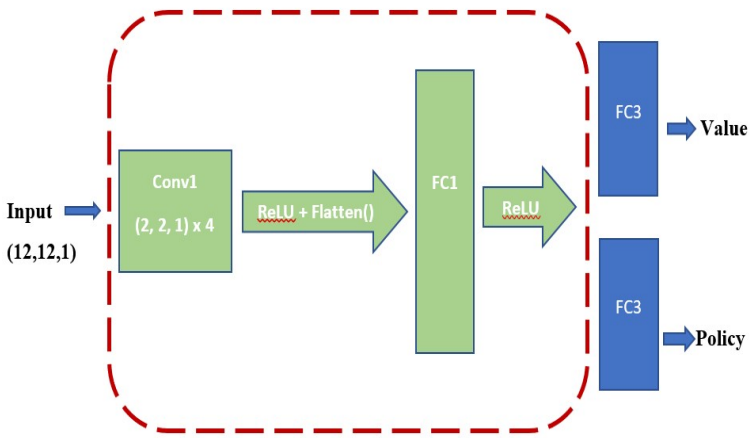


Figure 3. Model Structure

The neural network model used in this project is shown in Figure 3. This model has three layers, excluding the input and output layers. The input feature map is passed through one convolutional layer and one fully connected layer, the output is then passed through two fully connected layers with different activation functions to get the value and policy for the current state. Since the output feature map of the first fully connected layer is 48, the last computation step to get policy and value will not be offloaded to the hardware.

Hardware Implementation (FPGA architecture and system)

The top view of the hardware implementation is shown in Figure 4. The compute element of the design consists of four Multiply-Accumulators (MAC) working concurrently to generate outputs and three data buffers to store inputs and outputs of the MAC units.

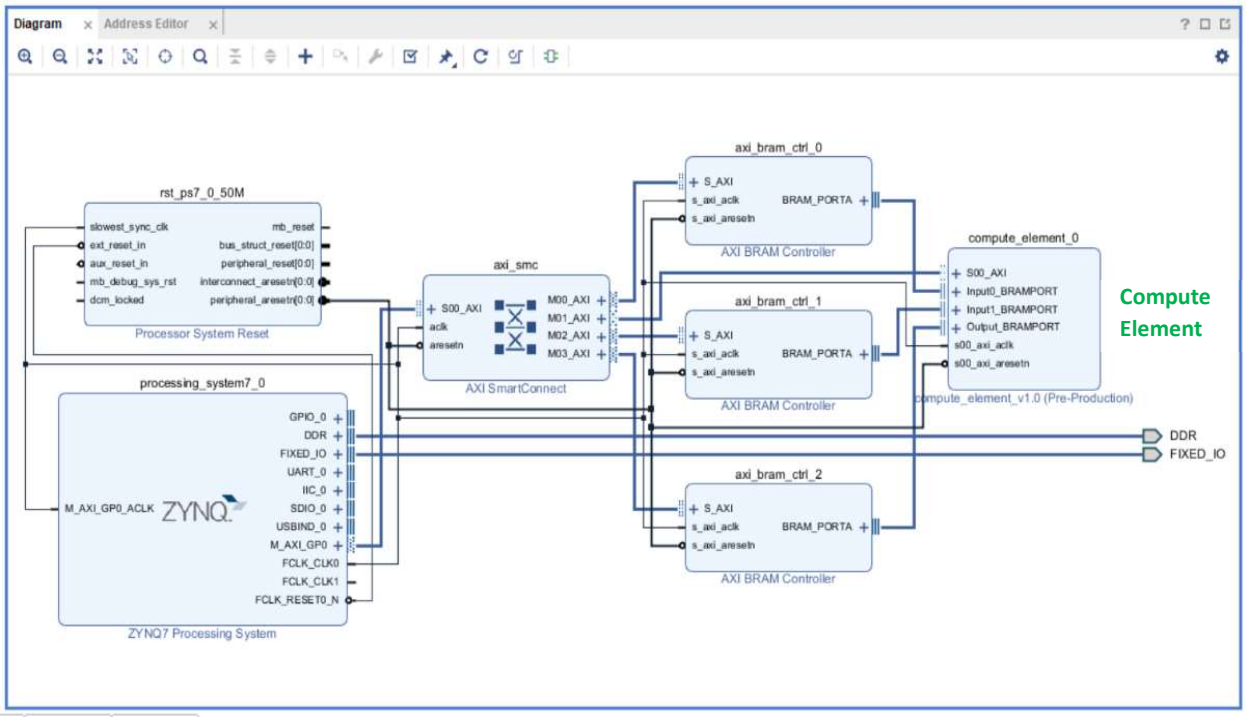
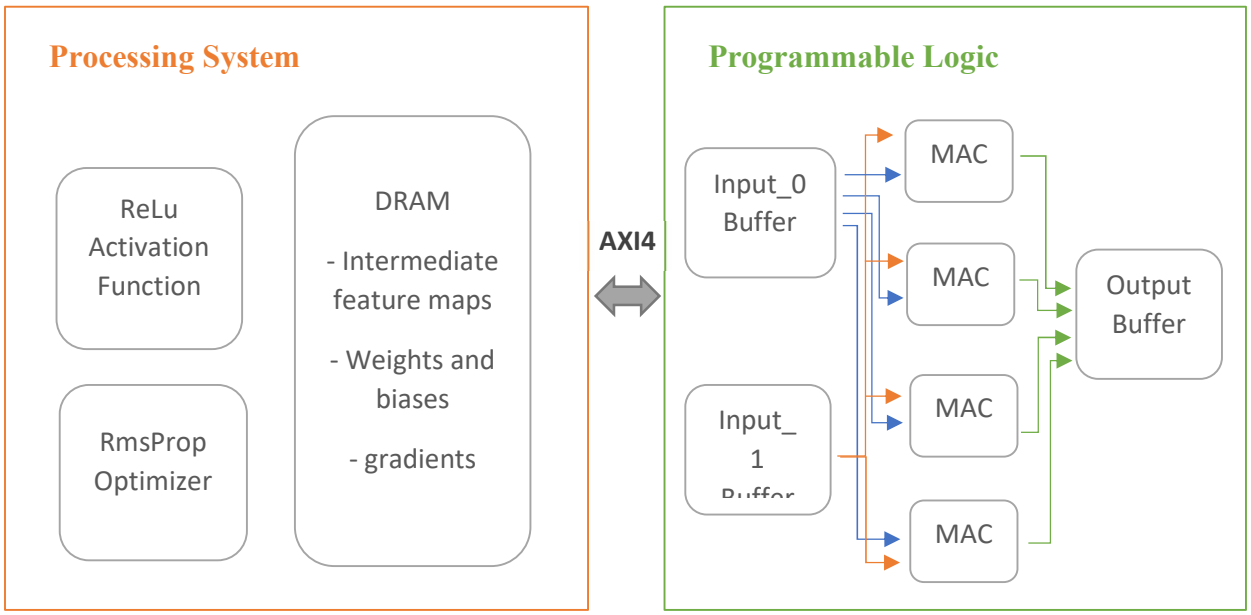


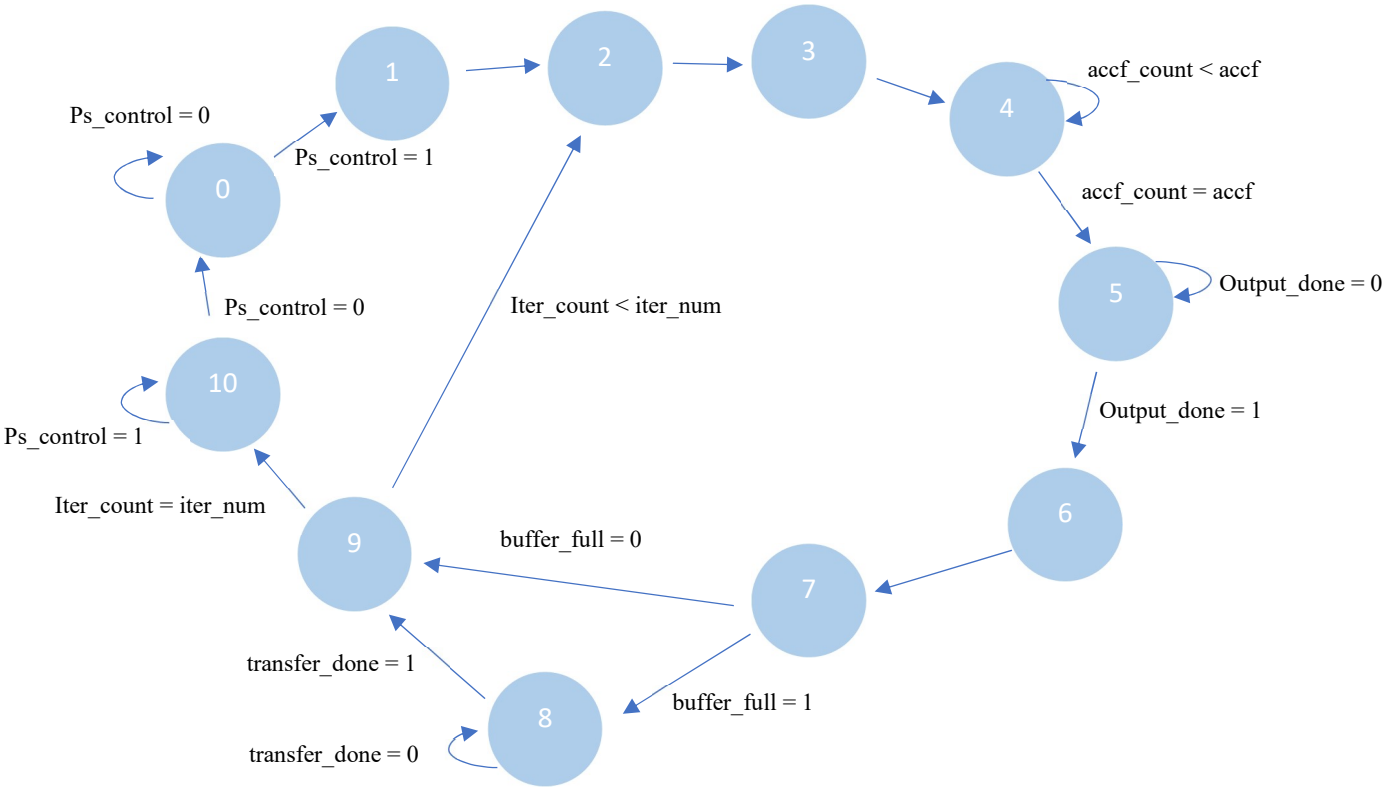
Figure 4. Hardware Implementation Top view



The top view of the processing system and programmable logic connection is shown above. The processing system takes from and sends data to the programmable logic, or the compute element in this case, for computation. It is also responsible for processing and storing data before and after the computation is done. Each MAC unit in the compute element takes inputs from the two input buffers. Input_0 is different for all MAC units while input_1 is shared. Each MAC unit is connected to one of the four on-chip BRAMs that make up the Output Buffer, which means four output values are generated during each iteration. Consequently, increasing the number of MAC units is expected to increase the system's parallelism.

There are five computation stages, two during inference and three during back propagation. At the start of each stage, the PS process data from DRAM and pass them to the on-buffers for calculation. If off-chip transfer is needed for this stage, the PS will wait for the PL to signal full buffer and make transfers accordingly; otherwise, the PS will wait for the PL to assert the done signal. Once computation for this stage is done, the PS load data from the Output Buffer and process them if necessary. For instance, the weight gradients will be used to update the weights through RMSprop optimizer and output feature map values need to be passed to the activation function.

The compute element data flow is controlled by a finite state machine and the state diagram is shown below with a brief explanation of each state.



Finite State Machine state definitions:

// state 0: wait for ps_control == 1 (PS start signal) to indicate start of a new phase

// state 1: initial set up for new stage, full_reg, accfreq, iternum, repeat, start_addr for each buffer

// state 2: start new iteration, pass start_addr to datapath module

// state 3: load inputs into mac units;

// state 4: assert valid_in; load next input values; assert valid_last if this is the last accumulation

// state 5: wait for final accumulation result

// state 6: write result to output buffer; check repeat_counter, update new_addr for next iteration

// state 7: check if transfer needed

// state 8: assert pl_full and wait for ps_control == 0 (transfer complete)

// state 9: compare iter_counter to see if this is the last iteration for this stage

// state 10: wait for acknowledgment from PS (ps_control == 0)

The on-chip input buffers use different data layouts for each computation stage. For example, below is the buffer layouts for the fully connected layer weights gradients calculation during back propagation. Input feature map values (y) for each episode in the current batch is stored in DRAM and loaded into Input_0 buffer sequentially for accumulation while the output feature map gradients are loaded into Input_1 buffer and fed to all four mac units simultaneously. For this stage, the accumulation number is the batch size, and the iteration number is 48*484/4 = 5808.