# ADAPTIVE ARCHITECTURAL ENHANCEMENTS FOR 16-BIT MACHINE LEARNING APPLICATION-SPECIFIC PROCESSORS

By

SAMANTHA DEGENERO-SCHEIB

Bachelor of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
2017

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2018

ADAPTIVE ARCHITECTURAL ENHANCEMENTS FOR 16-BIT MACHINE LEARNING APPLICATION-SPECIFIC PROCESSORS

Thesis Approved:

Dr. James E. Stine

Thesis Advisor

Dr. Keith A. Teague

Dr. Sabit Ekin

ACKNOWLEDGMENTS

I would like to acknowledge the Air Force Research Laboratory for funding this research under contract FA8750-1-0261.

Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: SAMANTHA DEGENERO-SCHEIB

Date of Degree: DECEMBER, 2018

Title of Study: ADAPTIVE ARCHITECTURAL ENHANCEMENTS FOR 16-BIT MACHINE LEARNING APPLICATION-SPECIFIC PROCESSORS

Major Field: ELECTRICAL ENGINEERING

Abstract: This paper presents architectural enhancements that allow for the use of digital signal processing operations, such as convolution and multiply-and-accumulate operations, that can be applied to machine learning applications on a 16-bit processor while maintaining low power and area consumption. The proposed design focuses on implementing hardware for a type of machine learning called the convolutional neural network. In addition to the convolution and multiply-and-accumulate operations that can be computed with the addition of the proposed architectural enhancements, common case instructions such as multiplication and addition can be utilized without creating additional hardware. Included in this paper are the motivation for the enhancements, background on the building blocks necessary to modify the processor, the implementation of the architectural enhancements, and the verification and evaluation of the proposed design against other machine learning architectures using simulation and synthesis, respectively.

TABLE OF CONTENTS

Chapter Page

**I INTRODUCTION** . . . 1

1.1 Motivation . . . 1
1.2 Organization . . . 3

**II BACKGROUND** . . . 4

2.1 Machine Learning . . . 4
2.2 Convolutional Neural Networks . . . 5
2.3 Multiply and Accumulate Unit . . . 6
2.4 LC-3 . . . 7

**III METHODOLOGY** . . . 13

3.1 Modifying the Hardware . . . 15
3.2 Utilizing the Microsequencer . . . 15
3.3 CNN Architecture in Depth . . . 16
3.4 LC-3 Assembler . . . 19

**IV RESULTS** . . . 21

4.1 Assembly Usage of Hardware . . . 21
4.2 Simulation and Verification . . . 22
4.3 Synthesis . . . 23

**V CONCLUSIONS** . . . 25

5.1 Future Improvements . . . 26

**REFERENCES** . . . 27

**APPENDICES** . . . 29

APPENDIX A: Additional Figures and Tables . . . 29
A.1 Topographical Synthesis . . . 29
A.2 Simulation . . . 29
A.3 Finite State Machine - Instructions . . . 29
A.4 Finite State Machine - Interrupts . . . 29

LIST OF TABLES

Table Page

4.1 Cycles Per Instruction (CPI) . . . . . 23
4.2 Comparison of M2L2C-3 to Pre-Existing Architectures . . . . . 24

1.1 Topographical Synthesis Results . . . . . 30

LIST OF FIGURES


| Figure | | Page |
|---|---|---|
| 2.1 | LeNet-5 Architecture | 6 |
| 2.2 | LC-3 ISA | 9 |
| 2.3 | LC-3 Datapath | 10 |
| 3.1 | Flow Diagram: Adding Instructions to the LC-3 | 14 |
| 3.2 | M2L2C-3 Processor Datapath with Modifications Highlighted | 16 |
| 3.3 | CNN Hardware | 17 |
| 3.4 | CNN Micropipelined Control Path | 18 |
| 3.5 | M2L2C-3 ISA | 20 |
| 1.1 | Verification of CNN instruction in ModelSim | 31 |
| 1.2 | Updated LC-3 FSM adapted for use in the M2L2C-3 - Instructions | 32 |
| 1.3 | Unmodified LC-3 FSM utilized in the M2L2C-3 - Interrupts | 33 |

# CHAPTER I

# INTRODUCTION

Adaptive architectural enhancements are important in today's computer design because many applications require or need specific instructional-level support for extra features. One important enhancement for architectures is the use of machine learning [1]. Although machine learning is pervasive, especially in applications such as Internet-of-Things (IoT), there is less discussion on microarchitectures and extensions to Instruction Set Architectures (ISAs) for machine learning (ML). Some microarchitectures have been proposed [2] as well as architectural studies [3]; however, there have been a limited number of suggestions on small microarchitectures, such as microcontrollers, that might help machine learning. In addition, many of these microarchitectural changes can be quite large and involved. This paper aims to present architectural enhancements that are simple yet effective in augmenting application-specific architectures with machine learning capabilities.

## 1.1 Motivation

The overall motivations for the machine learning microarchitectural modifications lie in the applications it can be used for. Simplicity and effectiveness drive the motivations that affect the implementation of the modifications to the microarchitecture. The improvements that satisfy this motivation focus on low power and area consumption as well as efficiency. A low overall power and area consumption is accomplished by reusing the added hardware for multiple purposes, while efficiency is achieved through the implementation of the instruction in hardware, as it is predominantly

faster to implement an instruction in hardware than it is to do so in software. This is because software instructions are implemented by hardware and must go through the overhead of getting to the hardware layer as well. The end product of the motivations is a 16-bit microprocessor, capable of machine learning, which is named the Modified Machine Learning Little Computer 3 (M2L2C-3) [4]. This is accomplished by adding an instruction to implement convolution in hardware to the microarchitecture, which allows for the computationally complex portion of a convolutional neural network to be implemented in hardware.

Machine learning applications of small microarchitectures, such as microcontrollers, can benefit a variety of fields, as machine learning already does on larger architectures, but with the added benefit of consuming less area and power. With a low power consumption machine learning microarchitecture, complex data could be both collected and analyzed on device. For example, Kyong Lee and Naveen Verma propose a processor to handle high-order machine learning signal analysis functions for patient-adaptive monitoring and other medical applications.[5] Applications within the medical field are not the only applications machine learning microarchitectures can be applied to. IoT applications such as traffic monitoring, soil condition tracking, and animal health monitoring could benefit from low power machine learning microarchitectures as well, as it can be power consuming to continuously transmit data to be analyzed.[6]

In order to help quantify the extensions, a small 16-bit architecture called the Little Computer 3 (LC-3) is implemented and adapted to handle Convolutional Neural Network (CNN), Multiply and Accumulate (MAC), and multiply operations without the creation of additional hardware for each new operation. This is accomplished by adding a small amount of hardware that can be used to implement the desired instruction, in this case convolution for a use within a CNN, and reusing that hardware for other instructions such as the common case instructions of addition and multipli-

cation, as well as the MAC and other operations, simplifying the amount of hardware needed to accomplish all of the added functionality.

## 1.2 Organization

The rest of the paper is organized as follows. First, relevant background information will be provided covering the components used in the modifications as well as the base microarchitecture. This includes information on machine learning, convolutional neural networks, multiply and accumulate units, and the Little Computer 3. Following the background information is the methodology. Within the methodology, different aspects of the modifications will be covered, such as the hardware, the microsequencer, the finite state machine (FSM) , and the assembler associated with the Little Computer 3. The results are described with relation to simulation in Modelsim, verification, and synthesis in the next chapter. The final chapter will discuss the conclusions drawn from the results of implementing the methodology.

# CHAPTER II

# BACKGROUND

In order to create the 16-bit machine learning application-specific processor, a working 16-bit processor is needed. The processor that the modifications are applied to is the Little Computer 3 created by Yale N. Patt and Sanjay J. Patel [7]. The machine learning modifications proposed enable the processor to compute the convolution equation for a convolutional neural network, a type of machine learning, using one assembly instruction enabled by the hardware. This chapter provides some background on the processor as well as the proposed hardware modifications. The background for the hardware modifications is broken into three different sections, Machine Learning, Convolutional Neural Networks, and the Multiply and Accumulate Unit.

## 2.1 Machine Learning

Machine learning - a common topic today, especially among conferences - provides a way for computers to solve problems through different methods of learning, such as supervised and unsupervised learning. Through machine learning, programs can complete a variety of tasks, such as image, language, and speaker recognition. Supervised learning works by training a program by providing it with a large number of diverse inputs and the correct outputs associated with the inputs. Once sufficient training has taken place, the program can be used to complete the task it was trained for.

Training is an important part of supervised machine learning. In image recognition, in order to identify an image, the program must first be provided with a variety

of images from different angles and distances and a label associated with each image. One method of supervised learning that is useful for image recognition is the convolutional neural network.

## 2.2 Convolutional Neural Networks

Modern Convolutional Neural Networks are based on work from the research paper by Yann Lecun et al. [8]. In the aforementioned paper, the authors propose a neural architecture called LeNet-5, used for recognizing hand-written digits and words. The name is inspired from signal and systems theory in processing signals using the convolution operator or:

$$y[n] = x[n] * h[n] = \sum_{k} x[k] \cdot h[n-k] \tag{2.1}$$

The architecture of the LeNet-5 can be seen in Figure 2.1 [8] and can be expressed with 4 layers, convolution, subsampling, fully connected layers, and Gaussian connections. In the convolution stage, filters are convolved with the input image to produce feature maps according to the filter used. The subsampling layer then produces feature maps half the size of the previous feature maps produced through the convolution layer by performing a local averaging and sub-sampling. The fully connected layer consists of several layers of hidden nodes. In the full connection layers, the output of every node is connected to the input of every node in the next layer with computation occuring within each node. Finally, an output probability is produced in the final layer of the CNN. Further information on LeNet-5 can be found in the aforemention research paper by Yann Lecun et al. [8]

Convolutional Neural Networks have improved vastly since the creation of the LeNet-5. Such improvements include expanding into the topics of computer vision (CV) and natural language processing (NLP). CV applications include facial recogni-

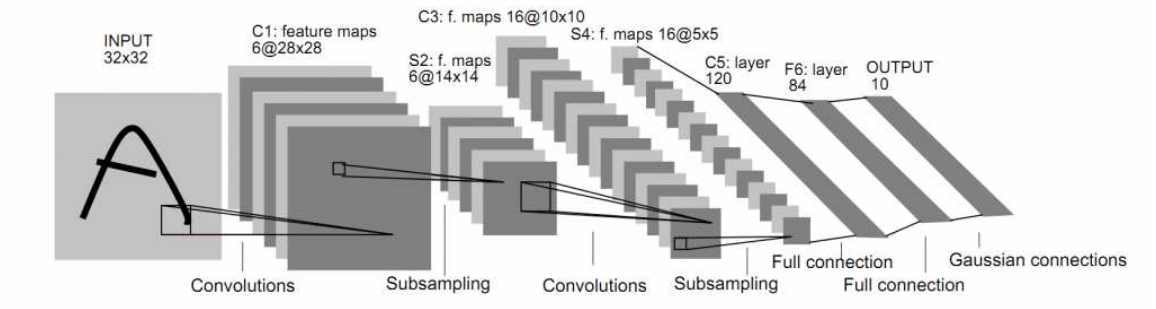


Figure 2.1: LeNet-5 Architecture [8]

tion, scene labeling, image classification, action recognition, human pose estimation, and document analysis, while NLP applications cover topics such as speech recognition and text classification [9]. CV and NLP can consequently be applied to a variety of fields, such as autonomous vehicles [10], health care monitoring [11], and many other areas.

### 2.3 Multiply and Accumulate Unit

Although convolution is extremely useful, it can be computationally complex for its implementation in hardware. Reusability of hardware is taken into consideration in order to make the addition of the CNN hardware more advantageous. This is accomplished by using a multiply and accumulate unit, which can be utilized for multiplication, addition, and the computation of digital signal processing (DSP) operations, such as convolution, filtering, and computation of the Fast Fourier Transform [12]. Because the MAC has similarities to a CNN, it can also be used in a CNN operation [13]. This can be seen within the following two equations that consist of multiplication and addition over N iterations.

$$z_{ij} = y_{ij} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} x_{i+m,j+n} w_{mn} \tag{2.2}$$

$$Output_{CNN} = \sum_{n=0}^{N} A_n B_n + C_0 \tag{2.3}$$

Equation 2.2 shows the equation for a 2-Dimensional MAC operation; breaking this equation down to one dimension produces Equation 2.3. Equation 2.3 has been written using the variable names that are provided to and result from the proposed assembly CNN operation, where $A_n$ and $B_n$ are the values to be convolved, $C_0$ is the initial offset, and N is the number of times to compute the summation within the convolution equation.

## 2.4 LC-3

The LC-3 micrarchitecture has a 16-bit Instruction Set Architecture (ISA), shown in Figure 2.2, which is made up of a 4-bit opcode and 12 remaining bits for hard-coded values and operands such as source registers, destination registers, and offsets. The architecture of the LC-3 that the modifications build upon and are evaluated against can be seen in Figure 2.3 [7].

The LC-3 is a Von Neumann architecture, also called a Princeton Architecture, meaning that it is made up of 5 main functional units. In addition to the 5 functional units, the LC-3 architecture also includes a bus, which can hold one 16-bit value per clock cycle, and 7 registers that can be used as operands, such as the source and destination registers, in the ISA. Within the Von Neumann architecture, the first is the memory unit, which allows the LC-3 to store and load data. In addition to the main memory block, the LC-3 also contains two memory registers, the memory address register (MAR) and the memory data register (MDR), which hold the addresses and data of an instance of memory respectively. The second and third units are the input and output (I/O) units. The I/O units allow for connections to exist between the processor and I/O devices such as keyboards, monitors, mice, and more. This component is not implemented in the M2L2C-3 or

the synthesized version of the LC-3 used for comparisons. The fourth unit is the processing unit. For the LC-3, this is the arithmetic logic unit (ALU), responsible

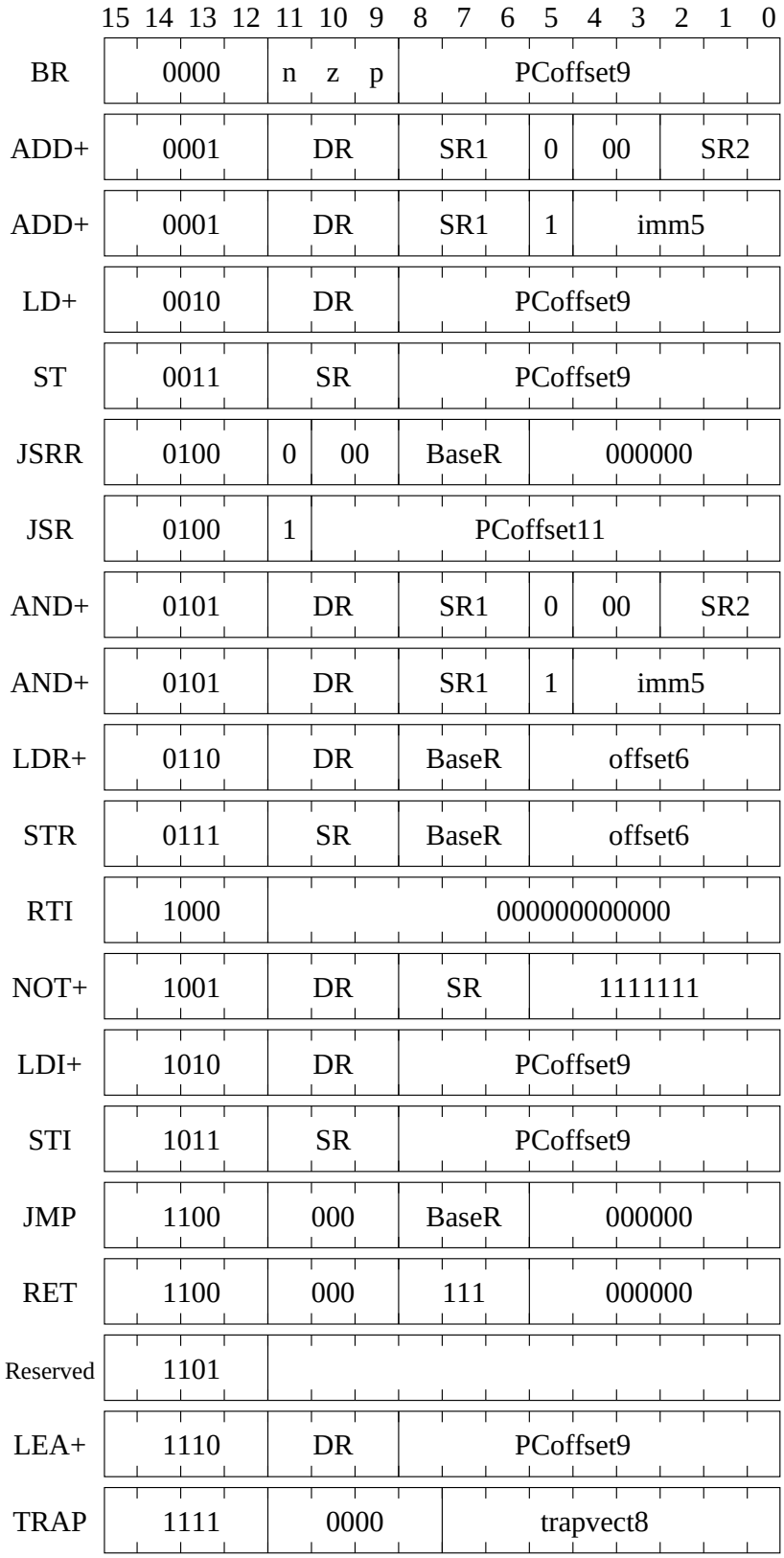


Figure 2.2: LC-3 ISA

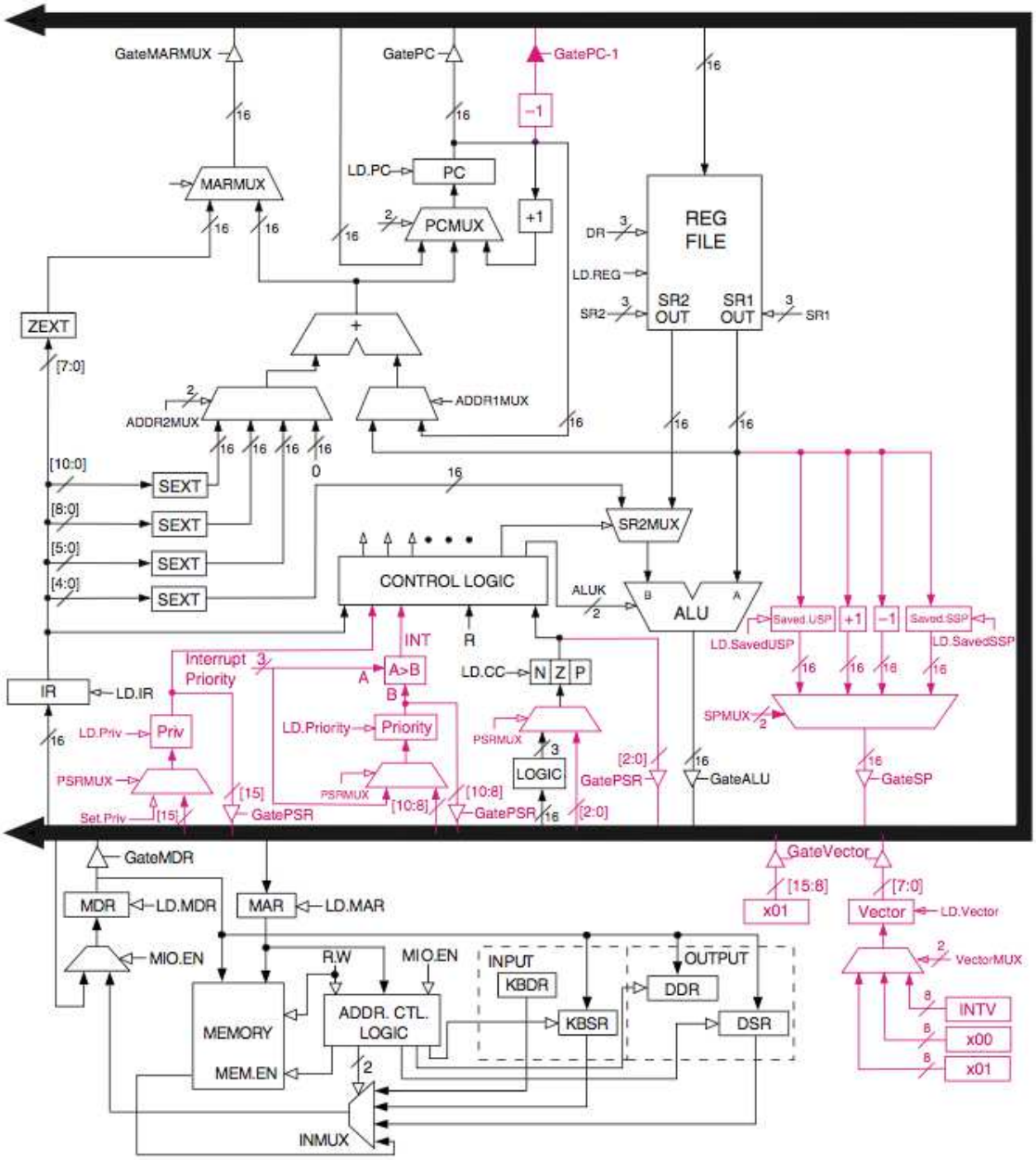


Figure 2.3: LC-3 Datapath [7]

for computing addition, bitwise AND instructions, and bitwise NOT instructions. The final unit is the control unit, which is made up of the instruction register (IR) and the program counter (PC). The IR and PC allow the processor to remember the current location within a program and allow for each instruction to be called and evaluated in the instruction cycle, which breaks down and evaluates each instruction

in steps.[7]

The instruction cycle consists of 6 possible steps, fetch, decode, evaluate address, fetch operands, execute, and store result, 2 of which (fetch and decode) are required for every instruction. The instruction cycle can be explained best using the finite state machine of the LC-3. The FSM, depicted in Figures 1.2 and 1.3, can be summarized as a sequence of states. To move from one state to another, the transitioning requirements must be satisfied, for example, to move from state 32, which takes in an instruction, to state 1, the ADD operation, the operand of the instruction must be 0x0001. The states within the FSM complete different steps of the instruction cycle. The fetch step is the process of collecting the instruction to complete. This occurs in states 18, 33, and 35. When combined, these states collect the address pointed to by the PC, increment the PC, pull the memory from the incremented address, and store it in the IR, using the memory address register and the memory data register to store the address between pulling it from the PC and storing it in the IR. The decode step occurs in state 32. As described previously, the decode step takes the opcode of the instruction taken from the IR, and moves to the state that matches the value of the opcode. The evaluate address step is used to determine and store address values. For example, within the LD instruction, in state 2, the address is determined by computing the sum of the PC and a 9-bit offset value provided when the instruction is called; the address is then stored in the memory address register. The fetch operands step can be seen within state 1. This step is used to pull the data from an operand, in this case source register 1 (SR1), in order to use the data from the operand. The execute step is used for instructions containing computations such as those found in the ALU. The store results step is used to save data into registers. An example of this step can be seen in state 27, where data from the memory data register is stored into a destination register.[7]

If the user wants to modify, remove, or add an instruction, both the FSM and the

microsequencer must be altered. The microsequencer is used to change the values of the control signals, depicted in Figure 3.2 as hollow arrows, for each state in the FSM. This is done by updating and running a Python file containing the control signals and their associated states. The FSM is modified by changing where each state points to in order to rearrange, add, or remove the states that make up instructions.

# CHAPTER III

# METHODOLOGY

The motivations for a low overall area and power consumption drive the changes towards reusing the added hardware by first creating the hardware for the desired CNN instruction and then utilizing that hardware to provide the functionality for other instructions, such as the common case instructions of multiplication and addition, without the addition of unnecessary hardware. Focusing on the reuse of hardware resulted in a CNN instruction (which can also be utilized as a multiply and accumulate operation), the hardware for a multiplication instruction if the FSM and microsequencer are modified, and the capability to solve DSP calculations, such as convolution, without additional hardware beyond that required of the MAC unit. Figure 3.2 removes the I/O components and highlights the hardware modifications made to the LC-3 to provide the capability to implement the CNN, MAC, and multiply operations.

The flow diagram for the changes to the LC-3, the FSM, the microsequencer, and the assembler can be seen in Figure 3.1. The process to add an instruction begins with the datapath of the LC-3. If additional hardware is needed to implement the new instruction, it must be added to the datapath. Once the hardware has been modified, the FSM is modified, if needed, to include the proper set of states for the instruction to go through. After checking the FSM for changes, the microsequencer may need to be changed as well. If this is the case, the appropriate control values should be set for each state and the microsequencer should be recompiled. Following the addition of the necessary hardware, states, and control values, the instruction is tested. If it does

not work as expected, the instruction is debugged in Mentor Graphics Corporation (MGC) Modelsim in order to determine what part of the architecture needs modified before testing it again. If the instruction does work, the assembler is adjusted to include the instruction. The following sections cover, in depth, the changes made to the LC-3 to produce the M2L2C-3 containing the CNN instruction.

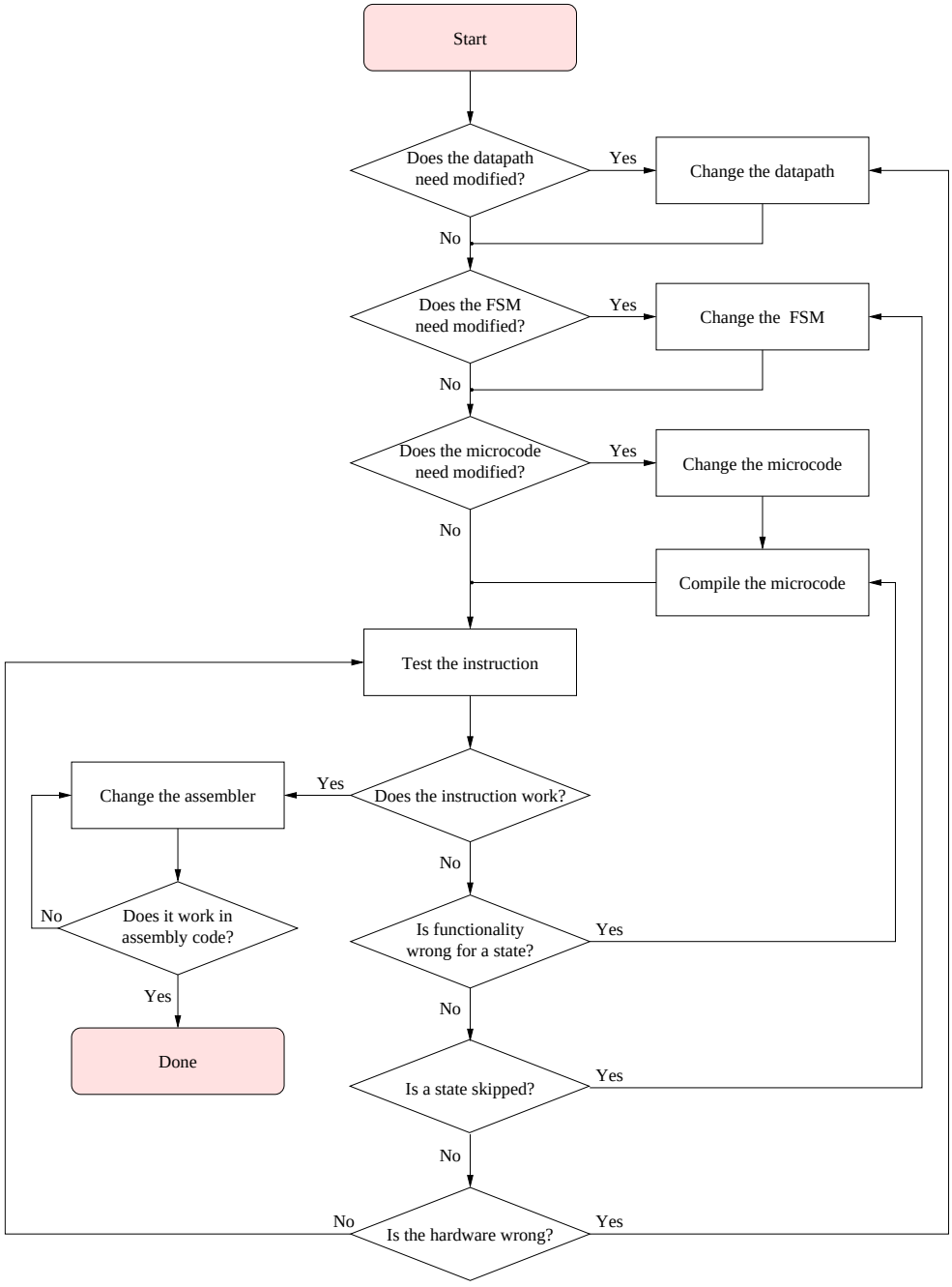


Figure 3.1: Flow Diagram: Adding Instructions to the LC-3

## 3.1 Modifying the Hardware

The LC-3 ISA (Figure 2.2) can only write to one register and can read from two per instruction. When implementing a CNN or a MAC, more than 2 values may be supplied for multiplication; for this reason, and because the LC-3 only accepts 2 inputs, the data supplied to the CNN instruction for convolution must be provided in the form of 2 vectors of data and must be preloaded into memory. The vectors are loaded into memory in alternating order before running the instruction. This is shown further in Chapter IV. In order to handle the vectors without calling the CNN instruction for each iteration, the user stack pointer (USP), designated as register 6 (R6) by the LC-3, is utilized.

Most CNN architectures have several computational elements associated with them. These involve the convolution, pooling, and fully-connected layers to form a full 3-step architecture. The hardware modifications used to create the M2L2C-3 affect only the convolution step within the CNN architecture. The basic structure of the modification is shown more in depth and with relation to Equation 2.3 in Figure 3.3.

## 3.2 Utilizing the Microsequencer

To optimize the control, the LC-3 and M2L2C-3 utilize microprogramming, or a microsequencer, to handle the implementation of the control signals for each state within the overall FSM structure (Figures 1.2 and 1.3 in Appendix 5.1) and to allow easy updates to the microarchitecture. Updates can include modifying, removing, adding, or moving instructions within the FSM. The specific micropipelined control diagram for the CNN instruction is highlighted in Figure 3.4. It contains 6 states specific to the CNN instruction as well as the first 4 states used to determine which instruction has been called.

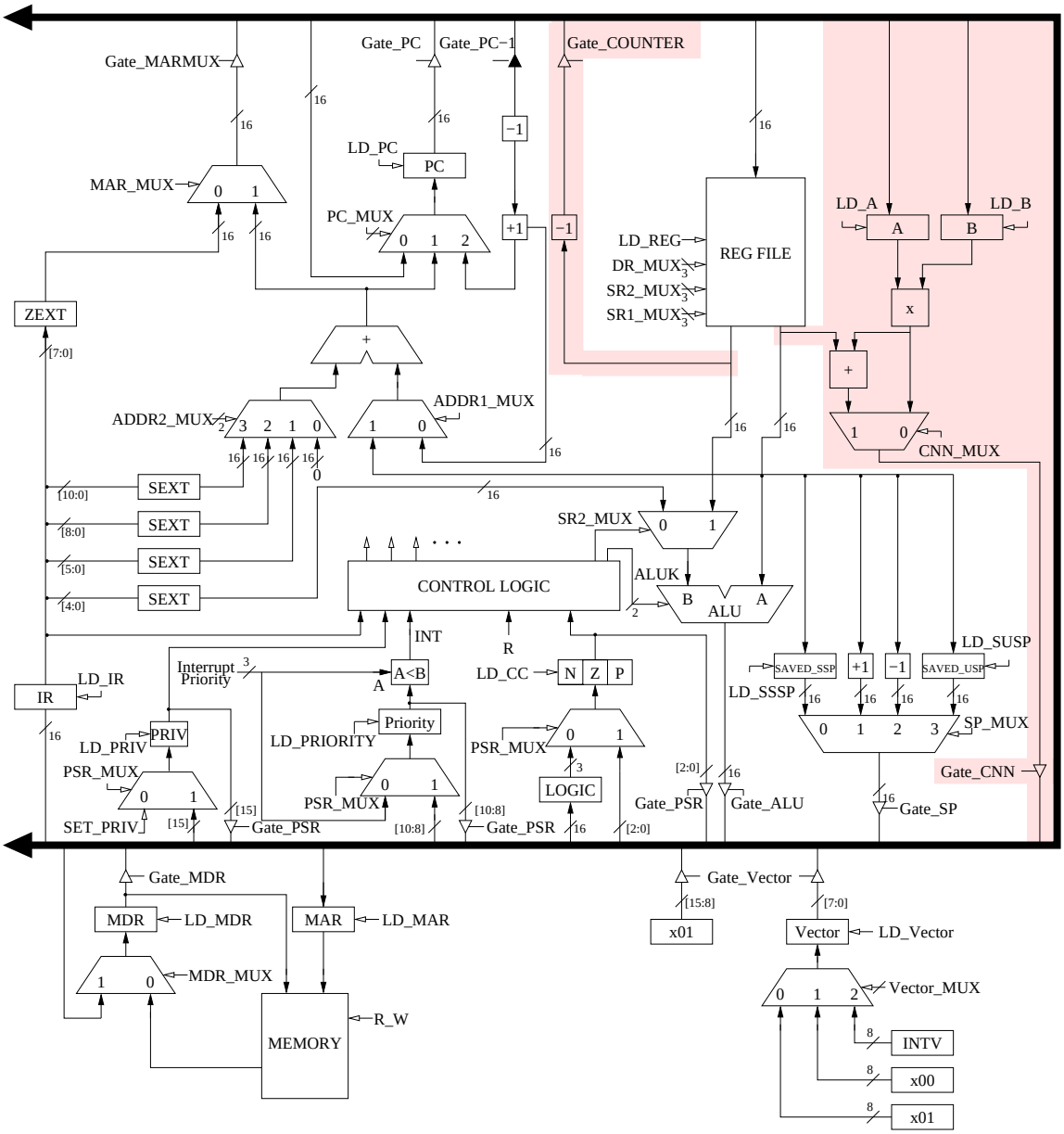


Figure 3.2: M2L2C-3 Processor Datapath with Modifications Highlighted

### 3.3 CNN Architecture in Depth

Before calling the CNN instruction, the user stack pointer (USP), carry in value (C), and the number of iterations (N) must be initialized in memory. The USP should be pre-loaded with the memory address that points to the first value of vectors A and B. The values of N and C come from Equation 2.3, where the value of N provides the upper limit of the summation of the convolution equation and the value of C

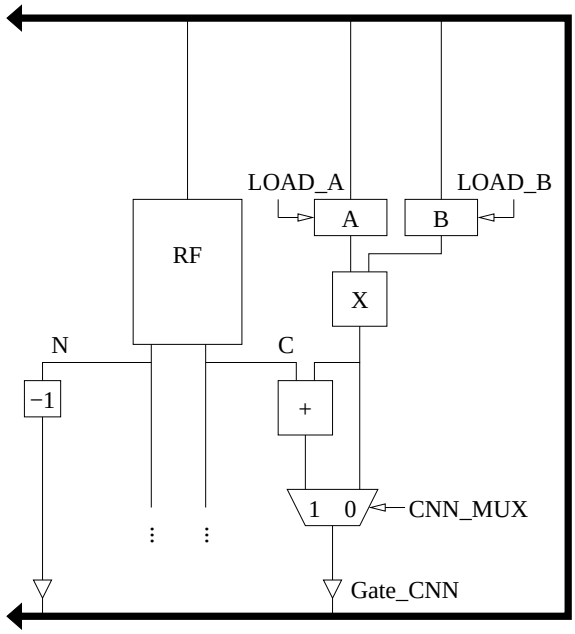


Figure 3.3: CNN Hardware

holds the initial offset to add to the summation during the initialization. During the instruction, N is decremented until it reaches 0, setting the condition code register (CCR) with each pass. The CCR holds a 3-bit value with each bit representing the flag for whether the value of the previous computation produces a positive, negative, or zero value. The condition code (CC) value determines whether the instruction computes another cycle or ends and move to the next instruction. In subsequent loops following the start of the instruction, C holds the total value produced by the previous iterations of the summation.

When the CNN instruction is called, it starts at state 5, the beginning of 6 states (highlighted in red) that make up the CNN instruction loop, as seen in Figure 3.4. In state 5, the memory location of the initial value of vectors A and B is loaded into the MAR and the USP by incrementing the previous USP value provided prior to running the instruction. The next state, 46, loads the memory of the address collected by the MAR into the MDR and repeats the functionality described in the previous state. State 53, loads the data stored in the MDR into the register that holds the current

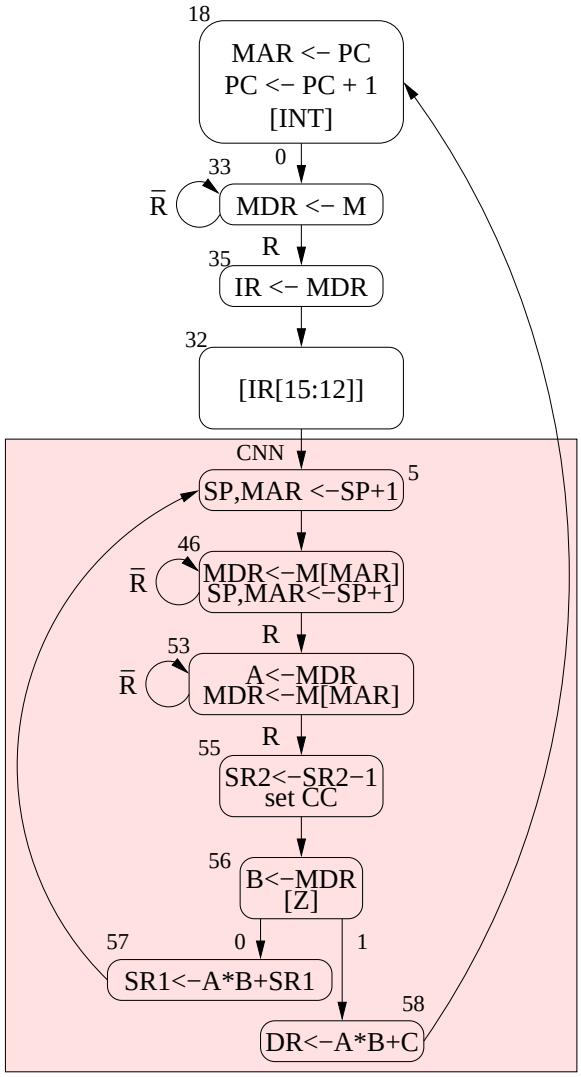


Figure 3.4: CNN Micropipelined Control Path

value from vector A and repeats the process to store data into the MDR. State 55 is responsible for decrementing the value of N and setting the CCR so that the cycle can end when the proper number of iterations is completed. The instruction ends when N reaches 0. In state 56, the process used to load register A is repeated for register B and the Z bit of the CCR is checked to determine whether the value produced should be stored in a temporary register and continue for another cycle (state 57) or in the destination register to return the final value (state 58).

### 3.4 LC-3 Assembler

The functionality to call the CNN instruction in assembly was added by modifying the LC-3 assembler. In order to get the assembler to accept the modifications made to the ISA in assembly, the files associated with the LC-3 assembler and simulator were modified to include the addition of the CNN instruction, and then recompiled.

Figure 3.5 shows the ISA of the M2L2C-3 produced by the modifications made to the datapath, FSM, microsequencer, and the assembler. This allows the user to call the CNN instruction from assembly while still having additional room for more instructions.

The inclusion of a function operand allows for the addition of more instructions while still maintaining the 16-bit architecture of the M2L2C-3. The instructions that contain the opcode 0x0001, ADD and NOT, have been modified to include a function operand, seen in bits 3 and 4, which allows for multiple instructions per opcode. Setting the function operand to 0x00 selects the ADD instruction, while setting it to 0x01 selects the NOT instruction.

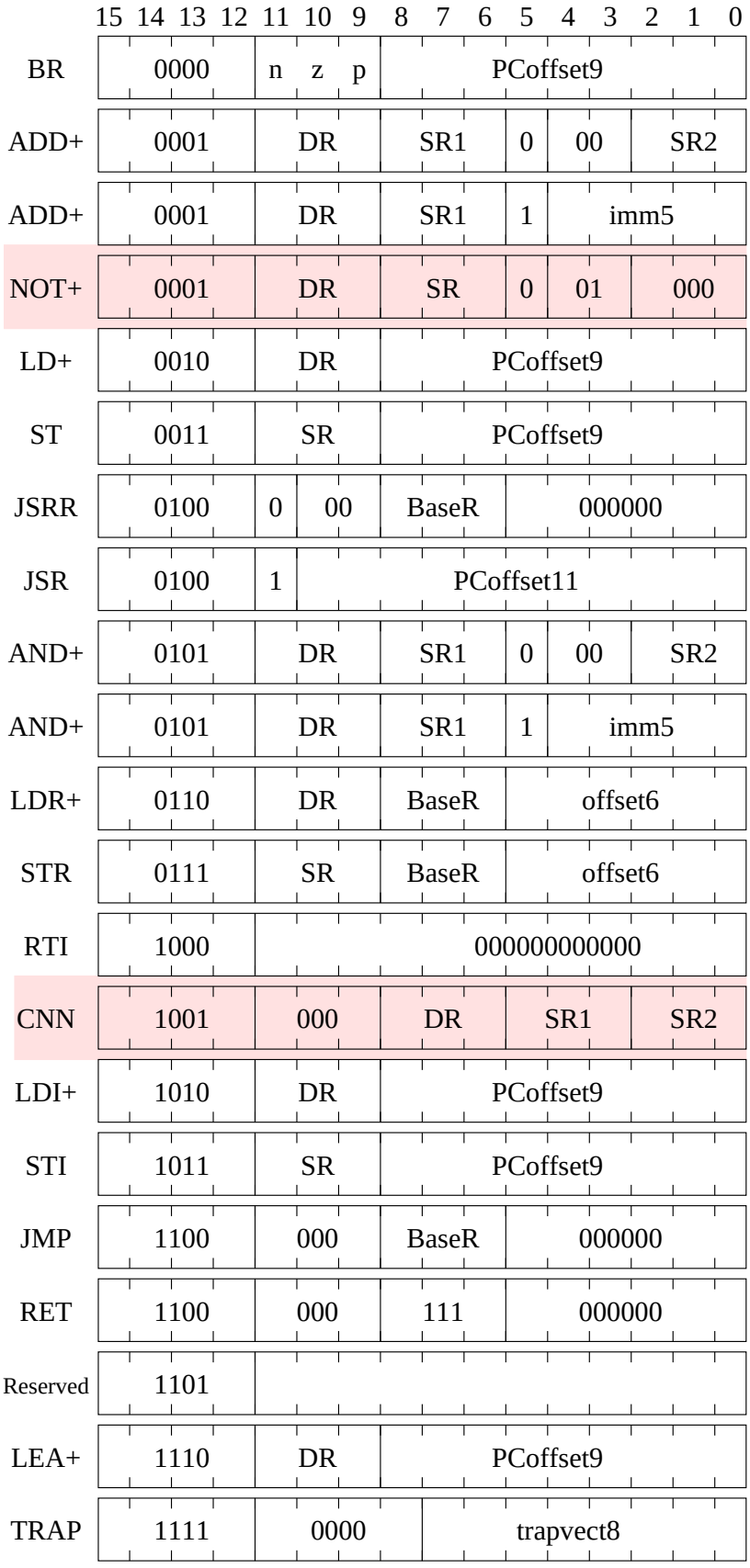


Figure 3.5: M2L2C-3 ISA

# CHAPTER IV

# RESULTS

The proposed design and modifications are implemented in register transfer level (RTL) compliant Verilog, a type of hardware descriptive language (HDL) code used to model hardware, and then synthesized in an ARM 32nm CMOS library in Global Foundries (GF) cmos32soi technology. The ARM standard-cell library utilizes multiple threshold voltage ($V_T$) values to aid in synthesis (i.e., MTCMOS), which was optimized for delay utilizing Synopsys® Design Compiler™ (DC) in topographical mode using a Process-Voltage-Temperature (PVT) variation at 25° C with typical-typical (TT) corners. Topographical synthesis, provided by Synopsys® DC™ (DC) ensures synthesis that accurately predicts timing, area and power by including information from the standard-cell layouts and underlying interconnects. Within synthesis, standard cells, which are premade to line up with other standard cells with respect to the power (VDD) and ground (GND) rails as well as the inputs and outputs of the cells, are used. The average power estimation was achieved by running the simulation on sample code test vectors. The synthesis scripts are synthesized for delay using a 1ns clock (1 GHz) and a 5× loading of a nominal flip-flop.

## 4.1 Assembly Usage of Hardware

Before running the CNN instruction, the USP and the input vectors, A and B, should be stored into memory sequentially (i.e., $A_0$, $B_0$, $A_1$, $B_1$, ... , $A_{N-1}$, $B_{N-1}$, $A_N$, $B_N$). The USP should hold the value preceding the memory address of $A_0$ (i.e. if $A_0$ is stored in memory location 0x303F, the USP should contain 0x303E before running

the instruction).

The CNN instruction should be used as follows, with the value of variable C from Equations 2.3 stored in SR1 and the value of variable N stored in SR2.

```
CNN DR SR1 SR2
```

If variable N is not stored in the proper register, the instruction will not complete the desired number of cycles with the appropriate initial offset. In addition, if variable C is not stored in the proper location, the intermediate results may be added to the data that is stored in SR1, which may be the result of a previous instruction or random data within the memory location.

An example of the usage of the CNN instruction in an assembly file can be seen below. Vectors A and B must be pre-loaded into memory according to the order discussed above. In this example, the vectors of data start at memory location 0x4000.

```
ld R6, USP
ld R3, C
ld R4, N
cnn R2, R3, R4
N       .FILL x000A
C       .FILL x0001
USP     .FILL x3FFF
```

### 4.2 Simulation and Verification

The following set of data was passed into an assembly program that calls the CNN instruction using the format shown as example code in the previous Section. Vectors A and B act as the values to be convolved and variables C and N contain the initial offset and temporary values between each cycle of the summation and the upper limit of the summation, respectively.

```
A = {1, 0, 2, 4, 1, 3, 2, 1, 0, 2}

B = {2, 7, 3, 1, 5, 0, 2, 3, 9, 1}

C = 1

N = 9

Result = 27
```

Running HDL simulation testing on the assembly program verifies that calling the CNN instruction produces the correct final and intermediate results. This can be seen in Figure 1.1, containing a portion of the Modelsim waveform produced from testing the instruction, in Appendix 5.1

Table 4.1 provides the cycles per instruction (CPI) of the microarchitecture implementation of the M2L2C-3 for the CNN, MAC, and multiply instructions. The CPI for each iteration of each instruction is large because vectors, rather than single values, are supplied to the instructions and multiple states must be executed in order to pull the data from memory. Each iteration of the CNN instruction goes through 6 states and consume 1 clock cycle per state, resulting in 6 cycles per iteration per instruction.

Table 4.1: Cycles Per Instruction (CPI)

| Cycles Per Instruction | | |
|---|---|---|
| Instruction | Iterations | Cycles |
| CNN | N | $6 \cdot N$ |
| MAC | N | $6 \cdot N$ |
| Multiply | N | $6 \cdot N$ |

### 4.3 Synthesis

In order to evaluate the proposed design, the M2L2C-3 is compared against data from Table X from the paper proposed by Ardakani et al. [14], containing 5 different machine learning architectures as well as the LC-3. Table 4.2 shows the area and

power results of running synthesis on the M2L2C-3 and LC-3 as well as the previously collected results of the other architectures. Table 4.2 also lists the methodology and technology used to determine the area, number of cells, and power for each architecture.

Compared to the base microarchitecture, the M2L2C-3 microarchitecture provides an increase in area, number of cells (see Table 1.1), and total power by 86.74%, 92.91%, and 42.67%, respectively. However, it should be noted that the LC-3 contains no form of multiplication (318 standard cells synthesized alone) or capability to compute the convolution operation and consequently has significantly less area (789 standartd cells vs. 409 standard cells). With respect to the other CNN architectures, the M2L2C-3 consumes significantly less area and total power. The M2L2C-3 consumes 99.98% less power than the machine learning architecture with the least power consumption, Envsion, and 99.95% less area than that of the machine learning architecture created by Ardakani et al., which holds the smallest area consumption.

Table 4.2: Comparison of M2L2C-3 to Pre-Existing Architectures

| Hardware | Methodology | Technology | Bits | Area [$mm^2$] | Power [mW] |
|---|---|---|---|---|---|
| LC-3 | Synthesis | 32 nm | 16 | 4.84E−4 | 3.07E−2 |
| M2L2C-3 | Synthesis | 32 nm | 16 | 9.04E−4 | 4.38E−2 |
| Chain-NN [14] | Synthesis | 28 nm | 16 | – | 567.5 |
| DNPU [14] | Silicon | 65 nm | $4-16$ | 16 | 63 |
| Envision [14] | Silicon | 28 nm | $1-16$ | 1.87 | $26-44$ |
| Eyeriss [14] | Silicon | 65 nm | 16 | 12.52 | $236-278$ |
| Ardakani et al. [14] | Synthesis | 65 nm | 16 | 1.77 | $254-260$ |

# CHAPTER V

# CONCLUSIONS

The architectural enhancements proposed in this paper allow for the use of a convolution instruction, which allows for a section of complex computation within a convolutional neural network to be accelerated through hardware on a 16-bit microarchitecture. The modifications provide a significant increase in area, number of standard cells, and power from the base microarchitecture; however, the base microarchitecture of the LC-3 contains no functionality for multiplication, which can add heavily to the area and power consumption of an architecture. This can be seen in Table 1.1, where the area associated with hardware for multiplication consumes 351.90 $\mu$m$^2$, or 72.65% of the area of the synthesized LC-3. When compared to the machine learning architectures from Table 4.2, the M2L2C-3 consumes significantly less area and power.

The motivations that affect the implementation of the architectural enhancements focus on the creation of simple, yet effective hardware modifications that produce a low overall area and power consumption. The requirements of the aforementioned motivations produced modifications that focus on the reuse of hardware, allowing for additional functionality while maintaining a small area consumption by creating the functionality for multiple operations. While the hardware added to the LC-3 for machine learning purposes focuses specifically on the convolution instruction for use in convolutional neural networks, the computation of convolution in hardware is not the only benefit of the modifications. The added hardware can also be used to provide functionality for other instructions, like those related to digital signal processing as

well as the common case instructions of addition and multiplication.

Machine learning microarchitectural enhancements such as those implemented in the M2L2C-3 could be applied to medical and IoT applications, as mentioned in Chapter 1. In addition, applications that currently benefit from machine learning could benefit from machine learning microarchitectures and the lower power consumption associated with them. One other application of the M2L2C-3 mirrors that of the base microarchitecture, the LC-3, to teach students about computer architectures, how to implement complex instructions for machine learning and digital signal processing applications, and how to test and debug the creation of microarchitectures.

## 5.1 Future Improvements

A significant disadvantage of using the LC-3 as the base for the modifications is that the 16-bit ISA limits the capabilities of the microarchitecture. Increasing the word size could greatly benefit changes for machine learning; however, the low area requirements of the 16-bit microarchitecture could benefit applications that need smaller footprints, such as IoT applications. The M2L2C-3 could also be improved with functionality that allows for multiple values to be stored and loaded (similar to the ARM store many (STM) and load many (LDM) instructions) to and from the register file. This would benefit users of the M2L2C-3 by reducing the number of instruction calls necessary to load the vectors of data before running the CNN instruction.

## REFERENCES


[1] P. Louridas and C. Ebert, "Machine learning," *IEEE Software*, vol. 33, pp. 110–115, Sept 2016.

[2] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "Dadiannao: A neural network supercomputer," *IEEE Transactions on Computers*, vol. 66, pp. 73–88, Jan 2017.

[3] C. Dubach, T. Jones, and M. O'Boyle, "Microarchitectural design space exploration using an architecture-centric approach," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 262–271, Dec 2007.

[4] S. DeGenero-Scheib and J. Stine, "Adaptive architectural enhancements for machine learning for application-specific processors with the lc-3," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, May 2019.

[5] K. H. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 1625–1637, July 2013.

[6] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei, and A. Basu, "Powering the iot through embedded machine learning and lora," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 349–354, Feb 2018.

[7] Y. N. Patt and S. J. Patel, *Introduction to Computing Systems: From Bits and Gates to C and Beyond*. New York, NY, USA: McGraw-Hill, Inc., 2 ed., 2003.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[9] A. Bhandare, M. Bhide, P. Gokhale, and R. Chandavarkar, "Applications of convolutional neural networks," *International Journal of Computer Science and Information Technologies*, vol. 7, no. 5, pp. 2206–2215, 2016.

[10] M. Schwarzinger, T. Zielke, D. Noll, M. Brauckmann, and W. von Seelen, "Vision-based car-following: detection, tracking, and identification," in *Proceedings of the Intelligent Vehicles '92 Symposium*, pp. 24–29, Jun 1992.

[11] X. Li, T. Pang, W. Liu, and T. Wang, "Fall detection for elderly person care using convolutional neural networks," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–6, Oct 2017.

[12] U. Ramadass, J. Ponnian, and V. Kumar, "A new recursive shared segmented split multiply-accumulate unit for high speed digital signal processing applications," in *2016 International Electronics Symposium (IES)*, pp. 203–208, Sept 2016.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[14] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, "An architecture to accelerate convolution in deep neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, pp. 1349–1362, April 2018.

# APPENDICES

## APPENDIX A: Additional Figures and Tables

### A.1 Topographical Synthesis

Table 1.1 displays the area and power analysis that resulted from running topographical synthesis on the M2L2C-3 and the LC-3. The same analysis is also provided for the multiplication instruction in order to quantify the increase in each area between the LC-3 and the M2L2C-3.

### A.2 Simulation

Figure 1.1 shows a small section of the waveform produced by running the CNN instruction on a set of vectors. The final result is stored in register 2, rf[2].

### A.3 Finite State Machine - Instructions

Figure 1.2 depicts the main portion of the M2L2C-3 FSM, an updated version of the LC-3 FSM adapted handle the new M2L2C-3 instructions [7]. The states that set the control signals for instructions are shown in this part of the finite state machine.

### A.4 Finite State Machine - Interrupts

Figure 1.3 depicts the second portion of the M2L2C-3 FSM, an unmodified version of the L2C-3 FSM [7]. These states affect the interrupt functionality of the processor.

Table 1.1: Topographical Synthesis Results

| Area and Power Analysis | | | | | | |
|---|---|---|---|---|---|---|
| Hardware | Bits | Area [$\mu$m$^2$] | Total Cells | Power | | |
| | | | | Static [$\mu$W] | Dynamic [$\mu$W] | Total [$\mu$W] |
| LC-3 | 16 | 484.38 | 409 | 27.60 | 3.07 | 30.7 |
| M2L2C-3 | 16 | 904.53 | 789 | 42.12 | 1.68 | 43.8 |
| multiply | 16 | 351.90 | 318 | 210.33 | 110.44 | 320.77 |

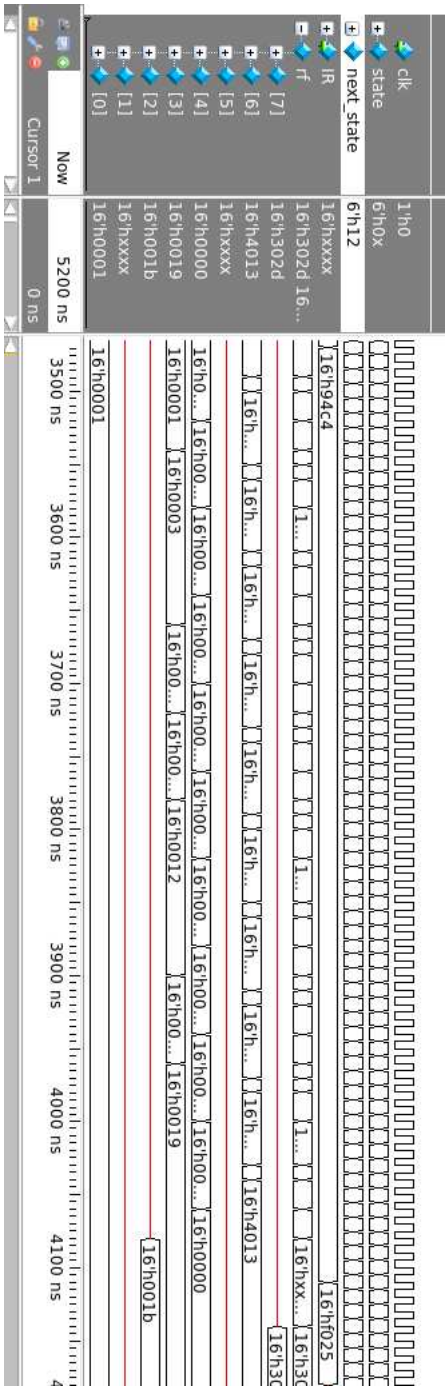


Figure 1.1: Verification of CNN instruction in ModelSim

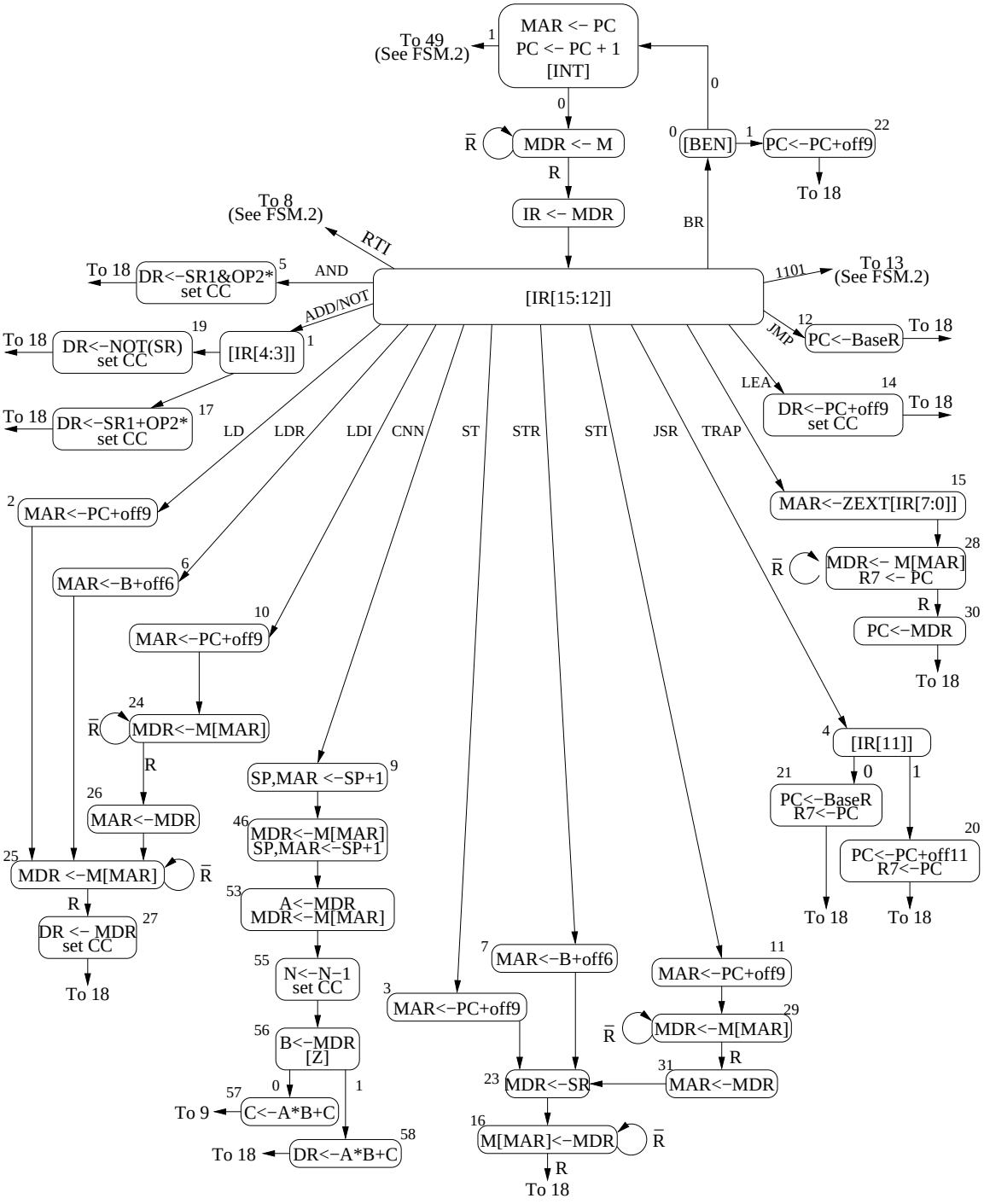


Figure 1.2: Updated LC-3 FSM adapted for use in the M2L2C-3 - Instructions [7]

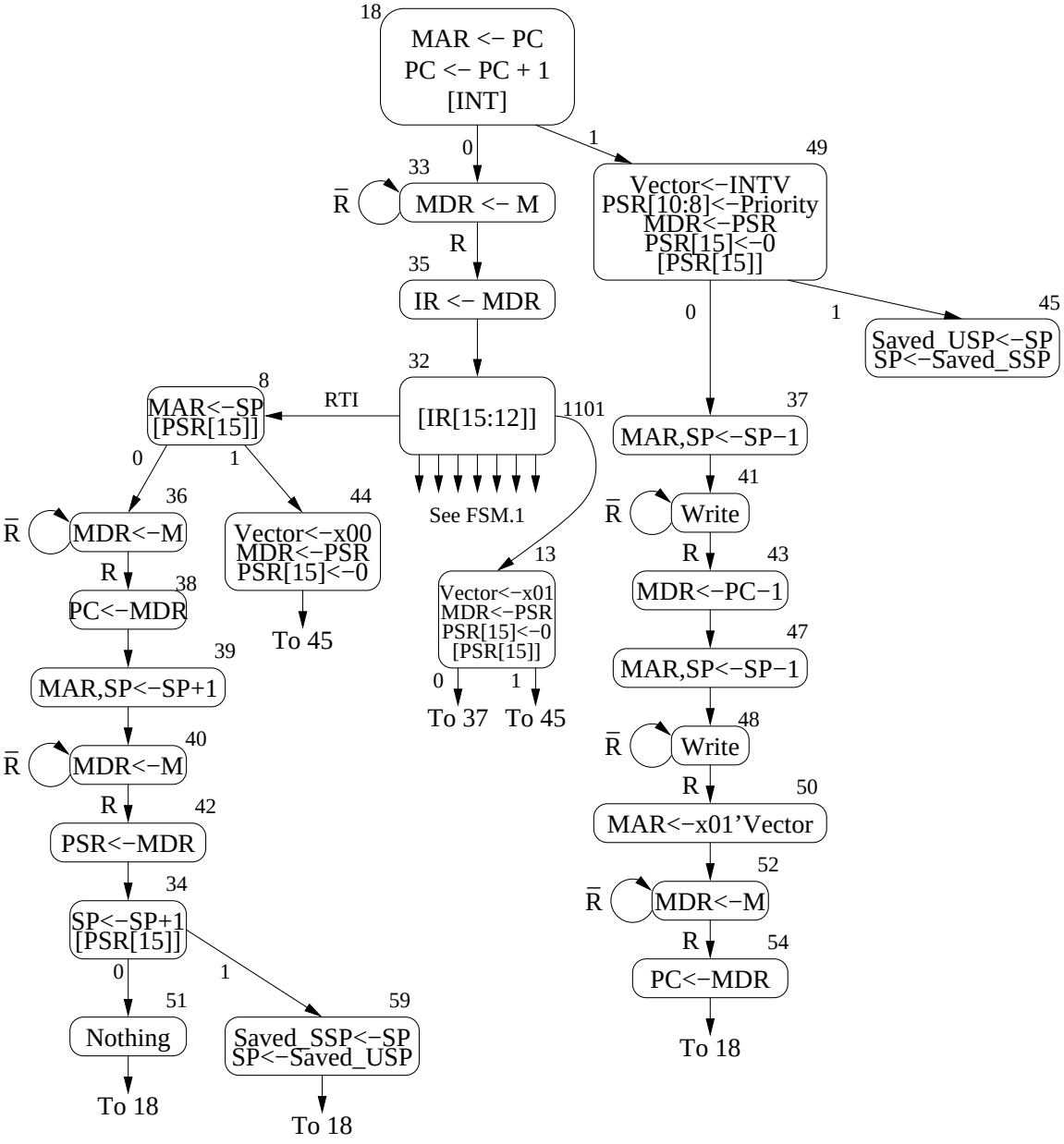


Figure 1.3: Unmodified LC-3 FSM utilized in the M2L2C-3 - Interrupts [7]

VITA

Samantha DeGenero-Scheib

Candidate for the Degree of

Master of Science

Thesis: ADAPTIVE ARCHITECTURAL ENHANCEMENTS FOR 16-BIT MACHINE LEARNING APPLICATION-SPECIFIC PROCESSORS

Major Field: Electrical Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in December, 2018.

Completed the requirements for the Bachelor of Science in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in 2017.

Experience:
Intern at Flight Safety International: Summer 2015
Intern at Sandia National Laboratories: Summer 2016
Intern at Sandia National Laboratories: Summer 2017
Intern at Sandia National Laboratories: Summer 2018
Research Assistant at Oklahoma State University: Fall 2017 - Spring 2018
Teaching Assistant at Oklahoma State University: Fall 2017 - Fall 2018

Professional Memberships:
Eta Kappa Nu