A SIMD microprocessor for image processing

Author: Zhiqiang Qiu

Student ID: 20716003

Email: CodyQiu3@gmail.com

Supervisor: Prof. Dr. Thomas Bräunl

Computational Intelligence - Information Processing Systems (CIIPS)

School of Electrical, Electronic and Computer Engineering

The University of Western Australia

THE UNIVERSITY OF WESTERN AUSTRALIA

Achieve International Excellence

1 November 2013

Abstract

The aim of this project is to design a Single Instruction Multiple Data (SIMD) microprocessor for image processing. Image processing is an important topic in computer science. There are many interesting applications based on image processing, such as stereo matching, 3D object reconstruction and edge detection. The core of image processing is matrix manipulations on the digital image. A digital image captured by a modern digital camera is made up of millions of pixels. The common challenge for most of image processing applications is the amount of data needs to be processed. It will be very slow if each pixel is processed in a sequential order. In addition, general-purpose microprocessors are highly inefficient for image processing due to their complicated internal circuit and large instruction set. One particular solution is to process all pixels simultaneously. A SIMD microprocessor with a simple instruction set can significantly increase the overall processing speed. This project focuses on the development of a SIMD image processor using software simulation. It takes a three step approach. The first step is to improve and further develop our circuit simulation software, Retro. Retro is a powerful circuit design tool with build-in real time graphical simulation. A number of improvements have been made to Retro to fulfil our design requirements. The second step is to design the actual SIMD circuit using Retro software. Tasks include designing the internal circuit of each Process Element (PE), internal circuit of the Sequencer CPU, interconnections between PEs and the instruction set of this SIMD image processor. The final step is to verify the correctness of the design by simulating the SIMD circuit in Retro using a number of image processing applications.

This has been a successful project. A number of image processing applications demonstrate the correctness of both Retro software and the SIMD circuit. This project is part of an ongoing project. It lays down a solid foundation and provides a good direction for future tasks. We should expect a simple and highly efficient image microprocessor available for embedded systems in foreseeable future.

Acknowledgement

I would like to acknowledge and express my deep gratitude to the following persons who made the completion of this project possible:

My project supervisor, Prof. Dr. Thomas Braunl, for his guidance, enthusiastic encouragement and valuable support.

My project partner, Tim Forrest, for his contribution and valuable support.

All teaching staff at UWA, for their assistance in my academic development.

The Faculty of Engineering, Computing and Mathematics for providing laboratory facilities.

Finally I would like to express my thanks and appreciation to our colleagues and industry persons for giving me such attention and time.

Content

Abbreviations

ACC	Accumulator
ALU	Arithmetic Logic Unit
AWT	Abstract Window Toolkit
CU	Control Unit
CPU	Central processing Unit
GUI	Graphical User Interface
FPGA	Field Programmable Gate Array
I/O	Input/Output
IDE	Integrated Development Environment
MSB	Most Significant Bit
VHDL	VHSIC Hardware Description Language
PC	Program Counter
PE	Process Element
Retro	Register-Transfer Object Hardware Simulation
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction Multiple Data
SoC	System on Chip

Chapter 1 Introduction

Image processing is a very important topic in computer science due to the large number of applications being used in our everyday life. The common challenge for most of the image processing applications is the large number of operations required. Microprocessors are often used in image processing. Although tody's microprocessors are much more powerful than the ones ten years ago, the amount of data acquired by image sensors has also increased rapidly. Furthermore, many microprocessors are designed for general purpose with the ability to process audio, image, video, and network packet data. However, the increased number of functionalities results in a very complicated internal circuit with a large instructions set. In terms of image processing, they are inefficient compared to a dedicated image processor. Therefore, there is a need to design a simple, highly efficient and low power image processor for embedded systems. After an intensive research, it suggested that Single Instruction Multiple Data (SIMD) architectures could significantly increase the processing speed. This will provide an on-board fast image processing solution for real time image processing applications.

This image processor will not replace any general-purpose microprocessor. It, will however, enhance the overall system performance by offloading the image processing task from the general-purpose microprocessor. This can be used in a large number of embedded systems, such as Raspberry Pi, Arduino, BeagleBone, and more. Figure 1.1 shows the setup of such system.

Figure 1.1 Image Data Flow in an Embedded System

In this setup, the raw digit image captured by the image sensor is first sent to our image processor. The image is processed by our SIMD image processor. The final result is then delivered to the on-board general purpose CPU. This can significantly reduce the work load of the on-board CPU by shifting the image processing task away from the CPU. Also, it reduces the bandwidth usage by sending results instead of raw data into the on-board CPU. The on-board CPU can spend more computational power and allocate more bandwidth to other tasks, which results in an increase in the performance of the overall system.

In many image processing applications, multiple pixels can be processed simultaneously. This project focuses on the development of an image processor using a parallel architecture called Single Instruction Multiple Data (SIMD). It takes a three step approach:

- Improve and further develop Retro
- Design SIMD circuit
- Verify the correctness by simulating a number of image processing operations

Chapter 2 provides some background information on circuit design. Chapter 3 covers the design process of Retro software. The result of the software stage is verified by using Retro software to design our SIMD circuit. The SIMD circuit design process is covered in Chapter 4. The result of both software and circuit design stages are verified in Chapter 5 by simulating the circuit using a number of different image processing examples. Chapter 6 draws a conclusion on this project and provides some recommendations on what can be done as the next step of the overall project.

Before we look into the actual design process, let us have a look at some of the background information in the following sections.

1.1 Digital Image

Images you see on your television and capture by your smartphone are digital images. Unlike a continuous picture drawn using a pencil, a digital image is a discrete image. It is made up of many small square elements termed pixel. Each pixel is filled with a colour. [2] A digital image is represented as a matrix in digital world. Each number in the matrix specifies the colour of each pixel in the image.

Figure 1.2 Pixel [2]

There are three types of digital images – binary image, greyscale image and colour image. A binary image is a 1-bit (monochrome) image. A pixel of a binary (black and white) image has only 2 possible values - 1 for black and 0 for white. [3]

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

Figure 1.3 Binary Image

In addition to binary image, a greyscale image use 8-bit of data to represent each pixel. In other words, a pixel in a greyscale image has 256 possible values. These values represent different darkness levels ranging from 0 for pure black to 255 for pure white. All other values give grey colours with different intensities. [1]

256	256	256	256	256	256	256	256	256	256
256	256	256	256	256	256	256	256	256	256
256	256	128	128	128	128	128	128	256	256
256	256	128	128	128	128	128	128	256	256
256	256	128	128	0	0	128	128	256	256
256	256	128	128	0	0	128	128	256	256
256	256	128	128	128	128	128	128	256	256
256	256	128	128	128	128	128	128	256	256
256	256	256	256	256	256	256	256	256	256
256	256	256	256	256	256	256	256	256	256

Figure 1.4 Greyscale Image

Unlike binary and greyscale images, there are a number of colour systems used to represent a colour image. The most widely used system is called RGB. A colour image is made up of three separate layers – Red, Green, and Blue. Each colour layer is a greyscale image. A pixel in a colour image carries three intensity values, and each one corresponds to each colour component. An artist can create lots of different colours by mixing them differently.

Similarly, the combination of different intensity of red, green and blue results in millions of different colours.[2] Although there are some other colour systems currently being used (such as CMYK for printing), we only focus on RGB images in this project.

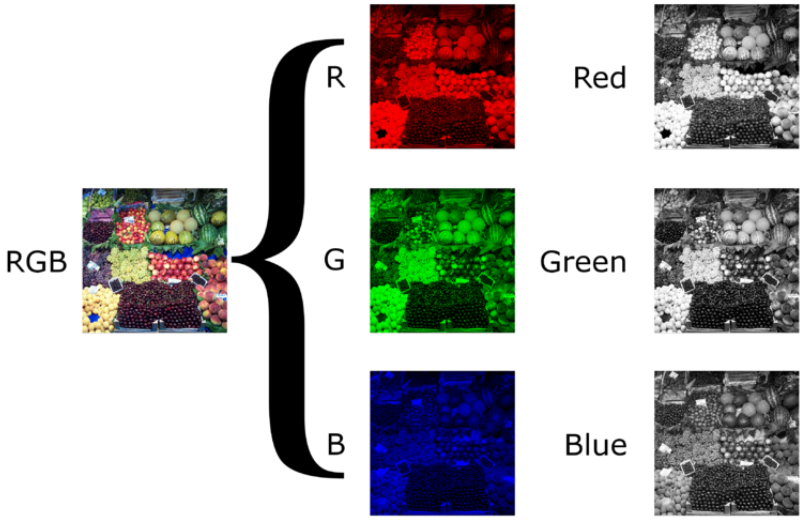


Figure 1.5 RGB Image[1]

The term colour depth is used to describe the number of possible colours a pixel can choose from. A binary image is also known as 1-bit monochrome image since each pixel has 2^1 possible values. As mention before, a greyscale image is an 8-bit image with 2^8 possible grey intensity levels. On the other hand, colour image has many different colour depth values. Some common colour depth values are: 8-bit, 15/16-bit (High Colour), 24-bit (True Colour), and 30/36/48-bit (Deep Colour). A higher colour depth gives a pixel more colours to choose from, which gives a vivid image. However, this also generates a much larger amount of data. For instance, an 8 megapixel 24-bit (8 bits per each RGB component) colour image capture by a consumer digital camera has the size of [4]:

8 * 10^6 (pixels) * 24 (bits/colour) ~ 24 Mbytes

This is a lot of data for an embedded system to process.

1.2 Image processing

There are millions of image processing applications that have been developed in the past few decades. These applications are used in every industry, such as quality control systems found in an assembly line, optical character recognition (OCR) system found in scanners, and motion detection system found in burglar alarms. They all follow the same procedure –

capturing and processing images. The capturing stage is normally done by image sensors, while the processing stage is done by microprocessors. This project only focuses on the processing stage.

Image processing can also be separated into two distinguished levels. The lower level provides hardware infrastructure support. In this level, digital images are treated as a collection of data. The actual content of the image is meaningless. On the other hand, the higher level provides methods and algorithms on how to process digital images. This level is also known as Computer Vision. Starting from a 2D image, computer vision attempts to extract information of a 3D scene [5] [6].

The core of image processing is matrix manipulation. As mentioned previously, a digital image is nothing more than a colour intensity matrix. Operations in both space domain and frequency domain produce various visual effects on the image. We can also generate lots of different images by combining a number of operations differently. Here is a list of some basic matrix operations and their corresponding effects:

- Transpose – rotation
- Amplification – changing the brightness and contrast
- Addition – Colour offset
- Finding the standard arithmetic mean of RGB component – convert into greyscale image
- Convolution/filtering – blurring, sharpening, edge detection
  Etc

Most of the image processing applications require a lot of processing power. For instance, if we blur an eight megapixel example using a 15 x 15 Gaussian Filter, the number of operations required roughly equals:

8 x 10^6 (pixels) x 15 x 15 ~ 5.4 billion operations

General purpose microprocessors used in many embedded systems are inefficient when performing such tasks. For instance, a Cortex A8 microprocessor takes roughly 3 seconds to process the entire image [7]. This can be critical for a lot of real time image processing applications.

1.3 Microprocessor

Computer vision provides the algorithms on how to manipulate a digital image, but this will not be possible without the support from its underlying hardware. Microprocessors are commonly used for image processing. In fact, because today's microprocessors are so cheap and capable of performing any sort of task, they have occupied every corner of your digital life.

Since the release of the first microprocessor, Intel 4004, in 1971, the performance of microprocessors has grown rapidly over the past four decades. New generations of microprocessors are faster and smaller by improving fabrication techniques, increasing wafer diameters, and reducing the size of transistors. [8] In 1975, the co-founder of Intel, Gorgon Moore, stated the following [9]:

"The number of transistors incorporated in a chip will approximately double every 24 months."

This is known as Moore's Law, which has be proven to be incredibly accurate and become the main driving force of the development of microprocessors.

There are two widely used computer architecture: Harvard Architecture and von Neumann Architecture. Both architectures consist of following four components [10, 11]:

- Control Unit (CU)
- Arithmetic Logic Unit (ALU)
- Memory
- Input / Output

The main difference between these two architectures is the way instructions and data are stored in the memory. In von Neumann Architecture, both instruction and data are stored in the same memory and hence the microprocessor cannot fetch both during the same cycle. This bottleneck restricts the throughput of the data bus between microprocessor and memory. In order to overcome this performance bottleneck, Harvard Architecture implies two memories, one for instructions and the other for data. Hence the microprocessor can obtain both instruction and data via two separate buses within the same clock cycle [10, 11].

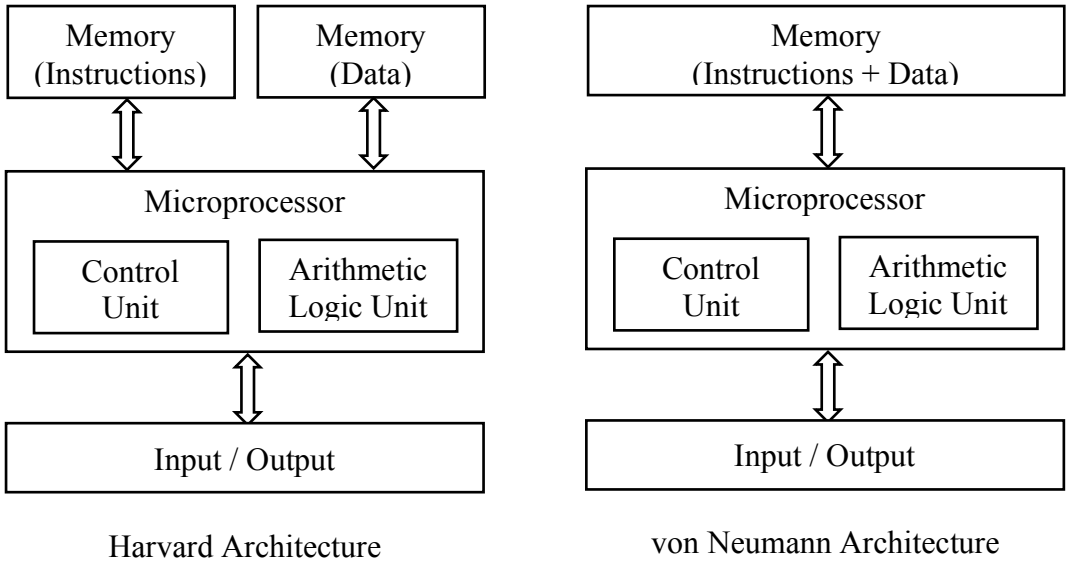


Figure 1.6 Computer Architectures

The first two components, Control Unit and Arithmetic Logic Unit, together form a microprocessor, or Central processing Unit (CPU). Control Unit is the command centre of the entire system. It directs other parts of the system according to the instructions stored in the memory. It does not perform any arithmetic operations on the data. The Arithmetic Logic Unit, on the other hand, performs either arithmetic or logic operations on data according to the instructions received from the Control Unit. Arithmetic operations are mathematical calculations, these include [12]:

- Additions
- Subtraction
- Multiplication
- Division
- Bit shifting

Logical operations are basically comparisons. These include:

- Equal to
- Less than
- Greater than
- AND
- OR
- NOT
- XOR
- NAND
- NOR

Our design is going to be based on von Neumann architecture. Chapter 4 covers the design process of the image microprocessor.

Chapter 2 Background

SIMD architectures have been around since the ILLIAC IV project in 1964[13-15]. This project perhaps the most infamous for it failure as a supercomputer project. The estimate cost increased significantly from $8 million in 1966 to $31million in 1972. There was only a quarter of the planned multiprocessors was constructed and the actual performance of 15 MFLOPS was a lot less than the initial predictions of 1000 MFLOPS[16]. Despite it was delivered to NASA Ames Research in 1972, the machine required another three years of engineering before it was functional. The following decade saw a slow development of SIMD architecture. Fortunately, Danny Hillis brought SIMD back to life with his Connection Machine.[17] However, after being resurrected in 1980s, the failure of Thinking Machines and MasPar slowed the investigation of SIMD once again. In the early 2000s, SIMD has managed to survive once again after being included in various ISA multimedia extensions, such as Intel's MMX/SSE[18, 19], Nvidia's CUDA[20], and AMD's 3D-Now[21]. The development of SIMD finally starts to pick up momentum in recent years.

As stated in Chapter 1, a digital image contains a large amount of information and it requires a lot of operations to process the entire image pixel by pixel. A lot of embedded systems running on SoC struggle in terms of image processing. Developing an image microprocessor using SIMD architecture could significantly improve the overall processing speed. Before we look at the actual design process, it is important to understand some of the basics.

2.1 Microarchitecture

According to Flynn's taxonomy, microarchitectures can be divided into four classifications [22, 23]:

- Single instruction, single data (SISD)
- Single instruction, multiple data (SIMD)
- Multiple instruction, single data (MISD)
- Multiple instruction, multiple data (MIMD)

	Instruction	
Data	SISD Single processor Computer	SIMD Vector/Array Computer
	MISD Pipeline Computer	MIMD Multiprocessor Distributed Computer System

Table 2.1 Microarchitecture Classifications

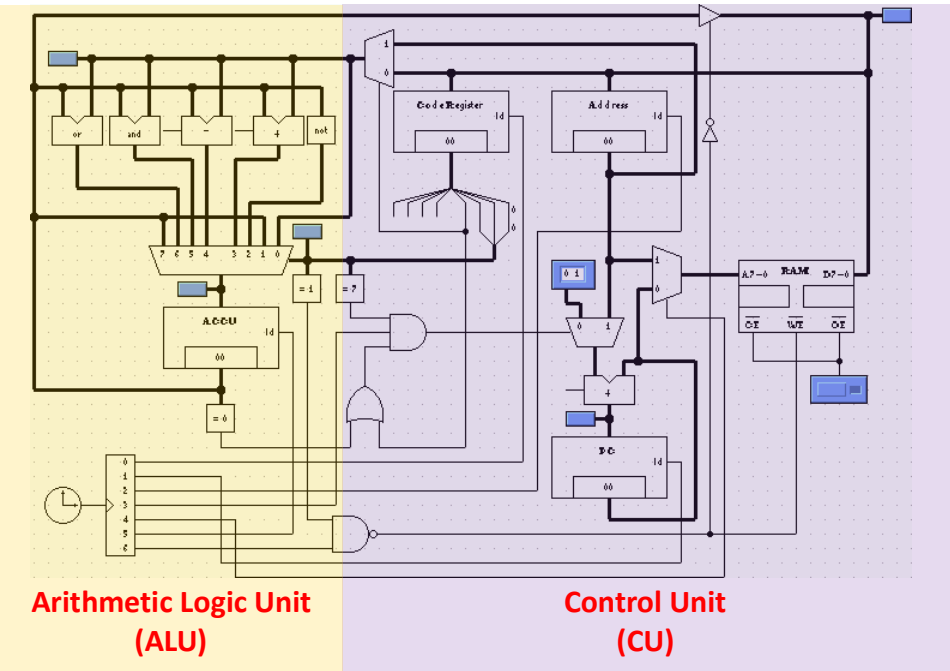


Figure 2.1 Single Instruction Single Data (SISD)

A SISD microprocessor is the most basic architecture [5]. It is made up of one control unit (CU) and one arithmetic logic unit (ALU). A single instruction is executed for every clock cycle, and since there is one ALU, it can store and process only one piece of data. This is very slow for processing a large amount of data because the data has to be queued and processed one by one. A digital image contains a large number of pixels (eg 8 megapixel

image contains 8 million pixel). It is a big challenge to process all the pixels in a split second, especially for real time image processing applications. It is clear that SISD architecture is not suitable for image processing. A parallel architecture should be considered as the next option.

Before we talk about circuits, let us have a look at an analogy. Imagine there is a one-lane street. Cars travelling down the street have to stay in the same lane, one after another. Congestion occurs as the number of cars increases. The common solution is to widen the street with additional lanes. Cars can now travel in different lanes. More cars can pass through for the same time period, or in other words, it takes less time for the same number of cars to go from one end to the other. Also, in both cases, only one traffic light is used to control the traffic flow, regardless the number of lanes.

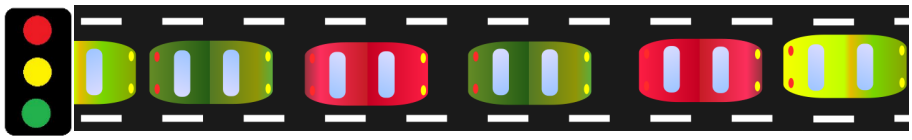


Figure 2.2 Single Lane Traffic

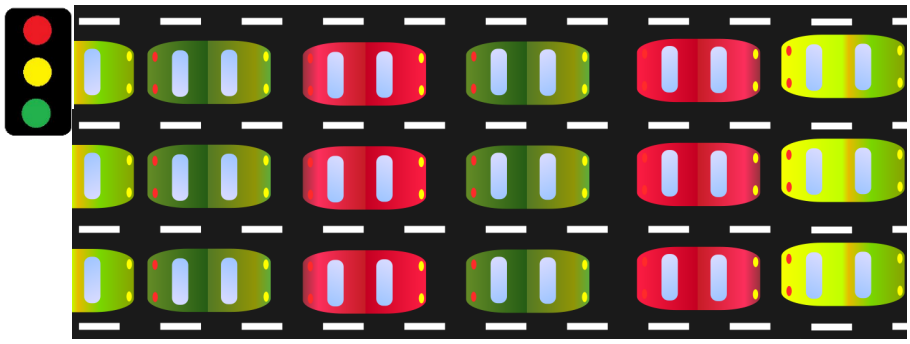


Figure 2.3 Multi Lane Traffic

Similarly, in many image processing applications, the same operations apply to all pixels, and different pixels may have different values. We therefore would like to have a system, which can send the instruction once and process multiple pixels at the same time. There is only one architecture suitable for this design idea – Single Instruction Multiple Data (SIMD) architecture. “Single Instruction” means there is only one CU in the microprocessor and “Multiple Data” means there are multiple ALUs used for multiple data.

2.2 Single Instruction Multiple Data (SIMD)

Unlike a SISD architecture, a SIMD microprocessor is made up of one CU and multiple ALUs (also known as Process Element, PE). In a microprocessor, the CU provides both data and instructions to ALU. The ALU then processes the data according to the instruction. These two parts are connected via two buses – a data bus and an instruction bus.

The following figure shows the overall layout of our SIMD design.

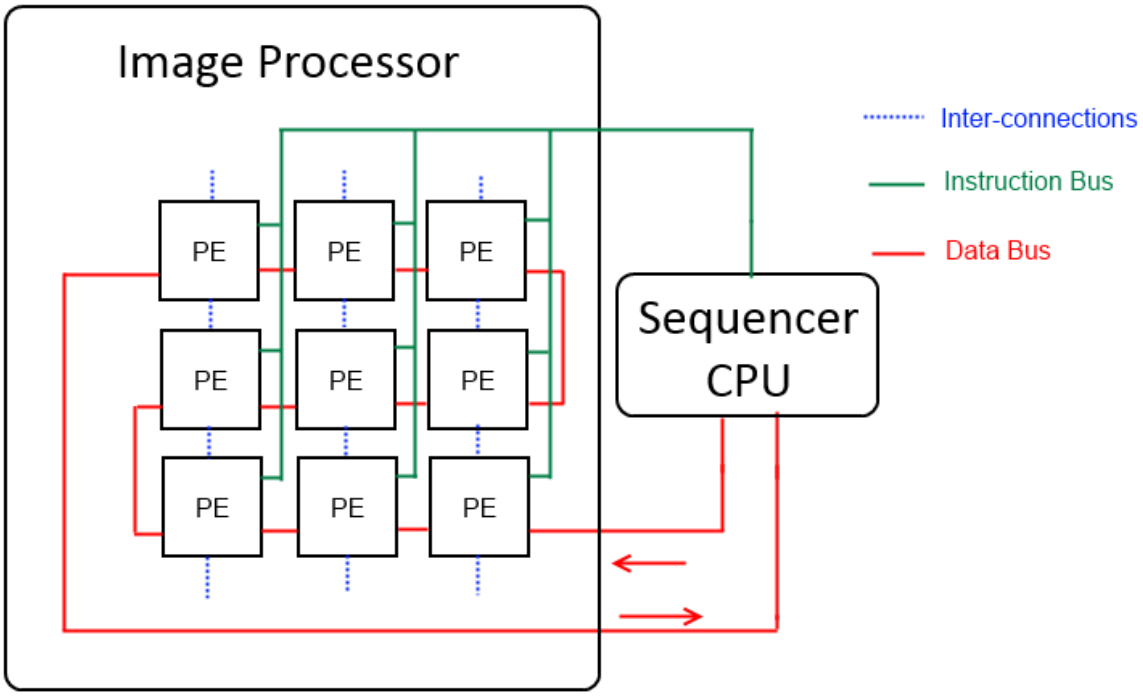


Figure 2.4 SIMD Design Concept

There are two regions in our design – Sequencer CPU and PE network. The Sequencer CPU is based on von Neumann architecture – both data and instructions are stored in the same memory. The role of the sequencer CPU is to provide input data and instructions to all PEs, and collect results from PEs. On the other hand, the role of a PE is much simpler. It is in fact an ALU (sometimes with a small local memory). It processes data according to the instructions received from the sequencer CPU.

There are two buses coming out from the sequencer CPU. All PEs are connected to the data bus in series. This data bus is a two way bus. Data coming from the sequencer CPU can be fed into each PE one by one. Data can also be sent to all the PEs in parallel using the instruction bus. This instruction bus is a one way bus and all PEs receive the same instructions and operand from the sequencer CPU at the same time. PEs are also

interconnected with their neighbours to form a network grid of PEs. These interconnects are two way buses and they allow PEs to exchange information with their neighbours without going back to the Sequencer CPU.

2.3 Instructions Set

Microprocessor architectures can also be classified by their instruction set. Some widely used architectures are X86, ARM (RISC) and SPARC. Each has its own territory: X86 microprocessors are used in every personal computer; ARM (based of RISC) microprocessors dominate the embedded world; and finally SPARC microprocessors play an important role in many supercomputers.

Over the past few decades, the size of the instruction sets of theses microprocessors has grown rapidly. This can be quite challenge for developers to use. In this project, we have decided to create our own instruction set. It is significantly smaller compared to those big names. Simple circuit design and small instruction set are the key features of this image processor.

Chapter 3 Retro

A traditional approach for circuit design is to use VHDL (VHSIC Hardware Description Language). It is a programming language largely adapted by the industries. However, designing circuits using VHDL is not straightforward and sometimes can be challenging. The lack of Graphical User Interface (GUI) means designers cannot build a circuit by simple drag and drop. It can also be hard to observe and verify the outputs easily without a real-time graphical simulation. Designing the circuit using a graphical interface with real time simulation can provide a better overview of the design and simplify the debugging process.

Software simulation has proven to be essential and powerful over the past few decades.

"A software simulation is worth a thousand wires"

– Professor John Lions [24]

There are four paradigms of science have been in human history [25]:

- Experimentation
- Theory
- Computation and Simulation
- Data Mining

The third paradigm was stated by Nobel Prize winner Ken Wilson in late 20th century. Since the introduction of first computer in late 20th century, computers and software simulations have played a key role in all scientific research. It has the abilities to produce results in a massive scale, which could not be achievable by theoretical analysis. This ability enables scientists to explore into a new territory [25].

3.1 Overview of Retro

It is extremely important to choose the right tool for the right task. Good design software can significantly increase the overall productivity. There are many circuit design products in the market, such as Cadence PSpice and CircuitLab[26, 27]. However, many of them are either used to design PCB circuits or simple memoryless circuits, in other words, these software products are powerful but not suitable for our design needs. Moreover, a lot of them are proprietary and it is impossible for to be customised.

A suitable development tool for this project should fulfil the following criteria:

- Open source
- Reliable and robust
- Meaningful Graphical User Interface (GUI) with simple drag and drop
- Powerful circuit design and simulation environment
- Easy to learn and use
- Scalable

After all, Retro stands out as the best candidate for this project. Retro stands for "Register-Transfer-Object Hardware Simulation". It was originally developed by B. "Toy" Chansavat in 1999 as part of his final year project at The University of Western Australia. Retro is an open source circuit design software with built-in real-time simulation. Anyone with basic circuit knowledge can build their own circuit with simple mouse drag and drop. This erases the requirement of knowing a circuit design language such as VHDL. By showing the circuit graphically instead of lines of VHDL source code, it allows us to have a better overview of the design and helps us more easily tackle issues during debugging.

Figure 3.1 Retro Splash Screen

Figure 3.2 shows the main user interface. On the top of the window is a menu bar and a toolbar. This toolbar consists of icons for frequently used actions. On the left of the widow is the component panel, which consists of buses and a list of all standard library components. Users can choose a wide variety of electronic components from this toolbox. The main drawing canvas sits in the centre of the Retro window. Users can create different circuits by placing different electronic components onto the canvas.

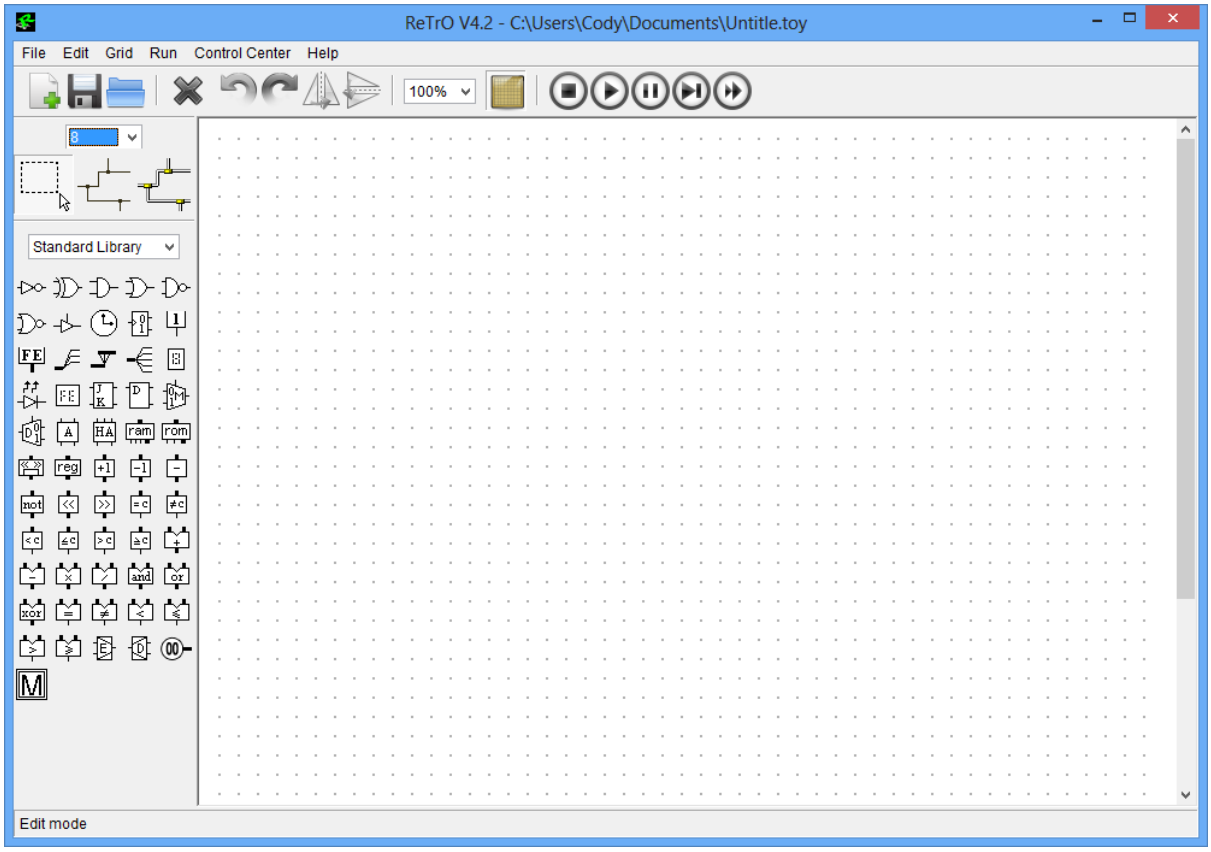


Figure 3.2 Retro Main User Interface

Retro may sound like a perfect environment for this project, but we still could not use the existing version straight out of the box due to a number of restrictions. Firstly, Retro was originally designed for simple circuits. It provides basic functionalities which makes it a good educational software for people learning circuit design, but not for production design. Additionally, there are a number of electronic components missing from its standard library, such as encoder and decoder. These components are crucial for many electronic circuit designs.

Fortunately, the open source Retro can be easily modified which allows it going from a learning tool to production software. It is the first and necessary step of this project. Section 3.5 ~ 3.7 describe some of the modifications and improvements have been made to Retro in order to fulfil our design requirements.

3.2 Retro source file

Retro is written in the Java programming language. Java is procedural, strongly typed, explicitly typed, case sensitive and object-orientated. It is developed and maintained by Sun

Microsystem and later Oracle Corporation. Unlike C programming language, Java applications cannot access lower level hardware directly; the entire runtime is governed by and performed on top of a Java Virtual Machine. This makes the development of Java applications much easier and more robust. By abstracting the hardware layer, Java applications can be compiled once only and executable on all supported platforms. This feature makes Retro very portable and distributable.

Java also comes with two GUI libraries – Abstract Window Toolkit (AWT) Framework and Java Swing Framework. AWT Framework is the first GUI framework to provide a basic abstraction over its native user interface. It was first released along with the first version of Java in 1995[28]. In the following year, Sun Microsystems, the company develop Java, introduced the second GUI framework, Java Swing [29]. Although Java Swing is built on top of AWT, it has been proven to be easier-to-use, more powerful and more robust compares to its ancestor. Due to the long history, Retro is developed using AWT framework instead of Swing framework. In this project, we primarily used AWT framework while some of the new add-ons are based on Swing framework. We have also chosen Netbeans and Eclipse as our integrated development environment (IDE) for the development of Retro software.

The object-orientated features and scalability are another two advantages of Java. This is a big step forward compared to its predecessor – C programming language. A number of Java objects are created when a Java application is executed. Each object belongs to a class, and each class can generate as many objects as you want. These object have the same attributes and actions, which specified by the class. And finally, multiple classes together form a bigger group called package.

In terms of an electronic circuit, each component is an object and belongs to a specific type. For instance, there might be many OR gate objects in a circuit, although each one is an individual, they all belong to the same OR gate class. This class specifies the attributes of all OR gates and all the actions they can perform (logical OR operation). All OR gate objects share the same attributes and actions regardless the location and the orientation of each individual OR gate. This matches perfectly with the object-orientated idea of Java mentioned above. Figure 3.3 illustrates this relationship and a real world analogy.

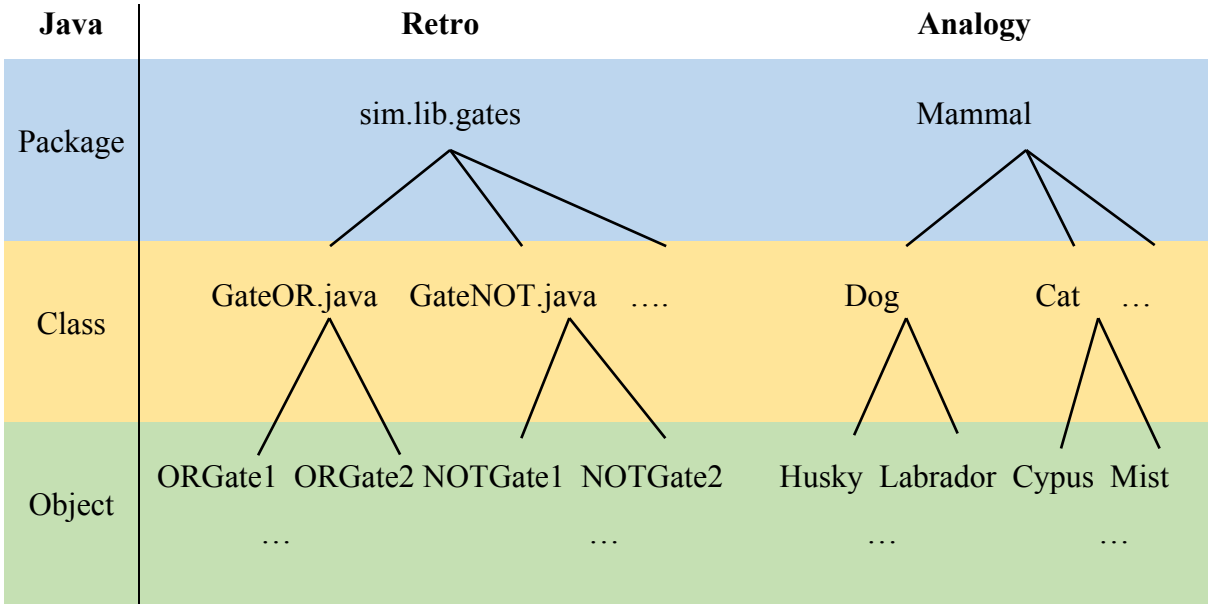


Figure 3.3 Java Hierarchy

Figure 3.4 is a screenshot of some of the source files of Retro. These file are grouped into different packets.

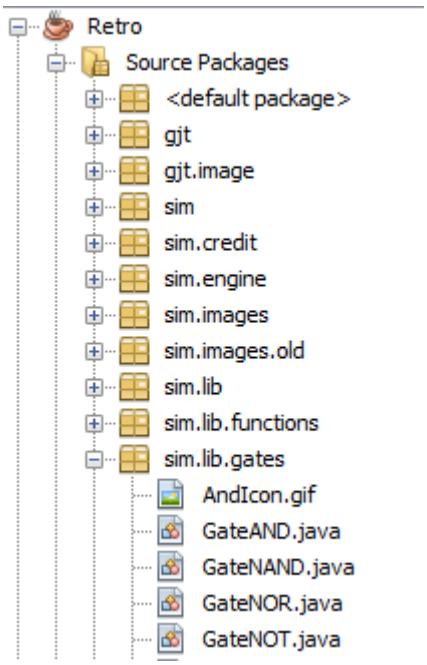


Figure 3.4 Retro Source File Hierarchy

3.3 Standard Library

Retro comes with a build-in standard library. It is a list of most commonly used electronic components. Each component is specified by its class file, which describes the attribute and behaviour of the component. The component can then be added into the standard library by adding an entry into the list. Using a standard library makes it very easy to add or remove any components.

3.4 Toy file

Similar to VHDL, components placed on the Retro canvas are described by a number of parameters. When the user hits the save button, Retro translates the entire circuit schematic into a plain text file and stores it onto user's hard drive. This circuit file is called Toy file. It is a long continuous string, which can be decomposed into multiple strings segments. One string segment specifies a component and multiple segments are separated by the symbol "#". A string segment specifies the following properties:

- the name of the component class
- parameters (the number of parameters may vary for different component)
- x, y coordinate of the component
- orientation of the component

Each field is separated by the symbol "|".

For example, an OR gate in Retro is saved as following:

sim.lib.gates.GateOR|null|6|4|7|3|0|false|1.0|2|false|00|#

Every time Retro opens up a toy file, it initialises the entire circuit using the following procedure:

1) Extract string segments from the toy file.
2) Look at the first field and locate the class of the component.
3) Generate a component from its class with the specified parameters.
4) Places the newly created component onto the canvas and displays it visually.
5) Repeat step 1 to 4 for all string segments.

This generates a complete circuit after the procedure.

3.5 Module

Module is the most crucial component we have added into Retro. It is the first step for Retro to transform from educational software into industrial designing tool. It gives Retro the ability to modularise an arbitrary circuit and reuse it in another circuit like any other electronic components.

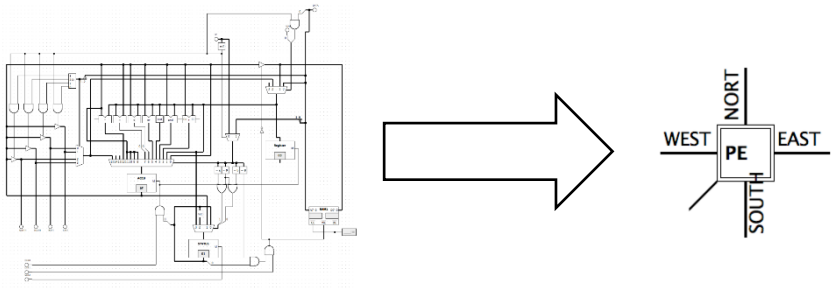
3.5.1 Why do we need modules?

The current version of Retro is only capable of simple circuit design. Each schematic is a stand-alone, completed, and self-contained circuit. It did not have the ability to create a partial circuit and bind different circuits together. It is almost impossible to have a large and complicated circuit on a single canvas. In real production circuit design, it is common to find the same part of the circuit occurring in multiple places. The designer has to place the same set of components repeatedly. This is not only time consuming, but also prone to mistakes. Being able to modularise and reuse an arbitrary circuit can significantly reduce the amount of workload and increase the productivity.

3.5.2 Solution

The design idea of a module is based on Black Box Theory. A black box is a system which can be represented by a fictional function and it produces some outputs for some given inputs [30]. A module can take an arbitrary circuit, memory or memoryless, and turn it into a standard library component. It can then be placed into another circuit as if you are placing an ordinary component. A circuit can have many modules, and each module is an individual which can have different internal circuits (of course they can also be the same internal circuit).

The module component acts as a bridge connecting different circuits. It has a number of input and output (I/O) port for the communications between two circuits. Data and instructions can be transfer in and out of a module through these I/O ports.



Circuit → Module

Figure 3.5 Modularise a Circuit

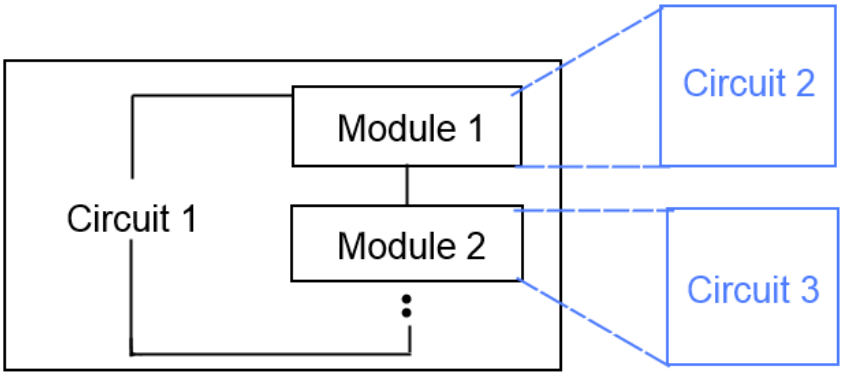


Figure 3.6 Circuit Relationship

The initialisation procedure of a module is similar to the initialisation procedure of a circuit. Every time the user places a module in the external circuit, it will open up the given toy file and generate all the components of the internal circuit. Unlike a normal circuit, Retro will keep the complete internal circuit running at the background, and will not display them onto the user screen. The user can only see the external circuit and the modules connected to it.

Same as designing other Java objects, there are two aspects – attributes and actions – we need to design for the module. A complete Java source code can be found in Appendix A.

3.5.3 Design

Figure 3.7 Module

A module has five different visual appearances depending on the number of I/O pins of the internal circuit (the details of pin will be covered latter in this chapter). The following figure shows all six appearances.

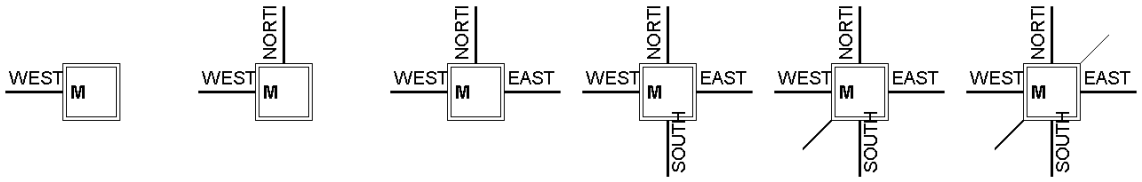


Figure 3.8 Six Appearances of a Module

Some people might be wondering why we did not group all I/O connections into one single bus. This is because it gives a clean and tide layout especially when designing a grid of modules if these I/O ports are spread on all sides of the module.

Every module component has a viewing window in the centre to display useful information to the users. In circuit design mode, this window shows the label of the module. Same as other components, this label is editable to allow users to give it a customised name. In simulation mode, we have decided to utilise this window and use it to display the real-time value of a designated register from the internal circuit (it will be empty if the internal circuit does not contain any register). This provides direct visual feedback to the user.

3.6 3.6 Pin

Pin is another important component we added to Retro. It works alongside the module. It creates access points for modules. It is essential for successful communication between two circuits. Every pin maps to an I/O port on the module. Data from external world arriving at the input port of the module will be passed onto the corresponding pin of the internal circuit. Similarly, internal data arriving at the pin will be passed onto the mounted output port of the module. A circuit without any pins is a self-contained environment and is restricted to public access. Attaching pins to a circuit allows the circuit to exchange data with external world and vice versa (require module).

Same as designing other Java objects, there are two aspects – attributes and actions – we need to design for the pin. A complete Java source code can be found in Appendix B.

3.6.1 Design

Figure 3.9 Pin

Each pin has a unique ID to distinguish between each other. This is required when mapping a pin to a port on the module. Each pin also has an editable text label. Users can give names to different pins.

3.7 Workspace

3.7.1 Why do we need a workspace?

Workspace is another very useful feature we have added into Retro. It is also a requirement for module to work properly across multiple computers. Earlier in this chapter we have discussed how a component is saved in a toy file – a string of parameters. This is absolutely fine for most of components since they use intrinsic parameters. These parameters does not depend on the operating system. However what if a component needs to read and write a file outside Retro? Take a look at the following examples:

- ROM:
  sim.lib.memory.Rom|null|10|7|63|31|C: \FYP\Design\mainrom.mem|8|256|10.0|5.0|#
- RAM:
  sim.lib.memory.Ram|null|10|7|63|31|C: \FYP\Design\maincpu.mem|8|256|12.0|12.0|#
- Module:
  sim.lib.others.Module|null|6|7|100|18|C:\FYP\Design\Draft\PE02.toy|null|PE02|#

Notice all three components need to access a file from external source. They record the file path as one of their parameters. This file path is also called an absolute path. Consider the following scenario: a user has designed a circuit and save it into a toy file as usual. The toy file contains one of those components. All of the sudden, the user decide continue his/her work using a different computer. An error message appears as soon as the user tries to open the toy file using Retro. This is because Retro is trying to access a file using the old file path (eg “C:\FYP\Design\Draft\PE02.toy”), but this file does not exist in the new computer. Moreover, different operating systems have different file system. File path representation used in Windows is different to the representation used in UNIX/Linux, which also causes a problem when a user is trying to open a toy file saved in different operating system. This file path is an extrinsic parameter for a component and it depends on the external user environment.

3.7.2 Solution

This problem can be solved by replacing the absolute path with a relative path. This is where we introduced the idea of workspace.

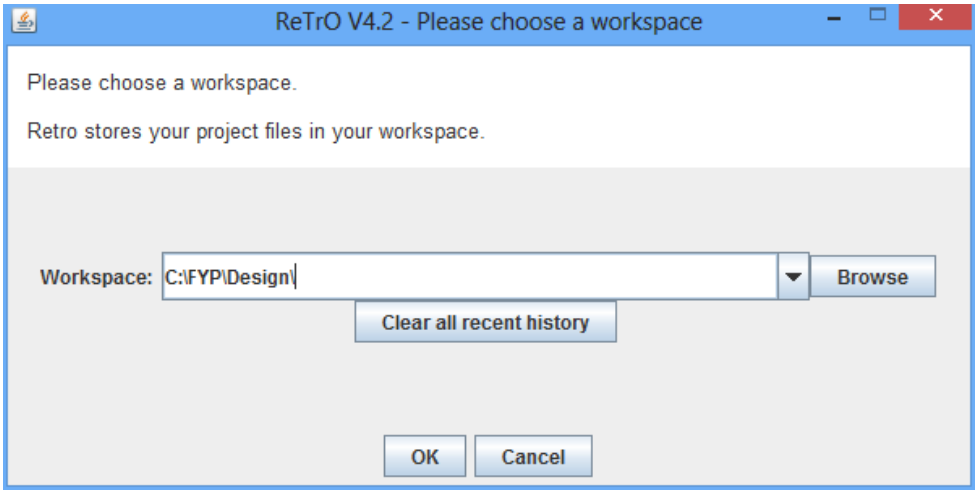


Figure 3.10 Workspace Chooser

A workspace is a directory where all project files are storing at. The path of workspace can be set by the user before the Retro main window, or by Retro every time it opens up a toy file. A special string "<WORKSPACE>" is used to replace the workspace directory when a circuit is saved. A toy file will no longer contain any absolute paths, instead, it stores the relative path of a file using one of the following format:

- <WORKSPACE>filename
- <WORKSPACE>subdirectory\...\filename

The previous example now becomes the following (workspace is set to "C:\FYP\Design\"):

- ROM:
  sim.lib.memory.Rom|null|10|7|63|31|<WORKSPACE>mainrom.mem|8|256|10.0|5.0|#
- RAM:
  sim.lib.memory.Ram|null|10|7|63|31|<WORKSPACE>maincpu.mem|8|256|12.0|12.0|#
- Module:
  sim.lib.others.Module|null|6|7|100|18|<WORKSPACE>Draft\PE02.toy|null|PE02|#

Since the workspace is dynamically set at runtime, as long as the user saves all the files in the workspace, the absolute path can be recovered correctly. This allows the user to work on the same circuit using different workstations.

There is now an extra step for both saving and loading a toy file. Before saving the circuit into the toy file, Retro scans all path and replace the prefix workspace path with the string "<WORKSPACE>". For loading, Retro opens up the given toy file and processes every

string segments. Every time it sees the keyword “<WORKSPACE>”, Retro replaces it with the current workspace path. Retro also replaces all slashes with the one used in the current operating system. After these two string substitutions, the rest follows the same procedure as stated earlier in Section 3.2 .

The complete source code is provided in Appendix C.

Chapter 4 SIMD Image Microprocessor

As described in Chapter 2, SIMD picks up a lot of popularities in recent years. This chapter covers the actual SIMD implementations. There are three parts we have designed in this stage: the instruction set used by the image microprocessor, the internal circuit of a Process Element (PE), and the internal circuit of a Sequencer CPU. Various functions of Retro have also been verified by using it to design the circuits.

4.1 Requirements

A number of factors need to be considered when designing a SIMD microprocessor[31]:

- Choice of Process Element (PE)
- Communications/network topology
- Instruction Issue

There is always a trade-off in simplicity and functionality for any circuit design. A decision need be made based on the specifications and the purpose of the design. SIMD PEs are normally interconnected to form a network grid. The network topology also needs to be defined before the circuit design process. Last but not least, we also need to develop a method by which the instructions coming from Sequencer CPU can be delivered to the PEs without errors.

Since this SIMD microprocessor is designed for image processing applications, it should have the following features:

- Simple, sufficient and highly efficient → minimum waste on transistors
- Interconnected to form a 2D network, and ideally, every PE receives one pixel value from the image sensor
- PEs are connected the same instruction bus and clock signal in order to achieve a synchronous system

Keeping these criteria in mind, we can start implementing the actual SIMD circuits step by step.

4.2 Instruction Set

The first step of designing a circuit is to construct its instruction set. As described in Chapter 2, the instruction sets used by the common architectures are often very large. This is partly because of many of them are designed for general purpose. This SIMD microprocessor is designed solely for image processing applications, and hence the instruction set can be trimmed down to a very small table. A small instruction set also makes developers' life easier.

An assembly instruction consists of two parts. The first part is called Op-Code, and the second part is called Operand. Operand can be either a numerical value, or an address location. Different semiconductor companies have different methods to identify the data type. Atmel microprocessors use separate Op-Code for address and numerical value[32]. Motorola microprocessors, on the other hand, use the same Op-Code for both data and address value. A single selection bit is used to differentiate between address and numerical value[33]. The later has a cleaner structure and smaller instruction set with less confusion. We have therefore adapted Motorola's method in our design.

There are two different circuits used in the SIMD microprocessor – one for the PE and the other one for the Sequencer CPU.

4.2.1 PE Instruction Set

Each instruction is 15 bit long. It has the following format:

MSB **LSB**

14 to 10	9 to 8	7 to 0
4bit	2bit	8bit
Op-Code	Type: 00 Constant (Seq + PE) 01 Memory (Seq + PE) 10 Neighbour (PE only) 11 Neighbour + Seq (PE only)	Data

Table 4.1 PE Instruction Format

The following table is a list of available Op-Codes for PE:

PE OPCODES

OPCODE	INSTRUCTION	INPUT	DESCRIPTION	OPERATION
0	NOP	-	-	-
1	LOAD	imm, addr or PE	Load constant from memory address into ACCU	ACCU <- RAM(imm)
2	ADD	imm, addr or PE	ADD to ACCU (without Carry in)	ACCU <- ACCU + <data>
3	AND	imm, addr or PE	Invert ACCU	ACCU <- $\overline{ACCU}$
4	NOT	-	AND with ACCU	ACCU <- ACCU · <data>
5	OR	imm, addr or PE	OR with ACCU	ACCU <- ACCU + <data>
6	ADDC	imm, addr or PE	ADD to ACCU (with Carry)	ACCU <- ACCU + <data> + 1
7	-	-	-	-
8	STORE	imm	Store ACCU at input address	RAM(imm) <- ACCU
11	LESSTHAN	imm, addr or PE	Compare <data> to ACCU, result stored in Activity register	ACTIV(0) <- ACCU < <data>
12	EQUAL	imm, addr or PE	Compare <data> to ACCU, result stored in Activity register	ACTIV(0) <- ACCU == <data>
13	ACTIV_INVERT	-	Invert the LSB of the Activity register	ACTIV(0) <- $\overline{ACTIV(0)}$

14	ACTIV_CARRY	-	If Carry set LSB of Activity register to 1	ACTIV(0) <- C
15	-	-	Currently unused	
16	ACTIV_ZERO	-	If Zero set LSB of Activity register to 1	ACTIV(0) <- Z
17	ACTIV_NEGATIVE	-	If Negative set LSB of Activity register to 1	ACTIV(0) <- N
18	STATUS_SHIFTLEFT (STATUS)	-	Bit-shift Activity register one place to the left, bringing in a 1 from the left	ACTIV(i) <- ACTIV(i-1)
19	SHIFT RIGHT (STATUS)	-	Bit-shift Activity register one place to the right, bringing in a 1 from the right	ACTIV(i) <- ACTIV(i+1)

Table 4.2 PE Opcodes

4.2.2 Sequencer CPU Instruction Set

The Sequencer CPU also has its own instruction set. Previously we have looked at the instruction set used by PE. A PE does not have CU and it has to obtain instructions from the Sequencer CPU. Therefore the sequencer CPU stores two sets of instructions – instructions for PEs and instructions for Sequencer CPU. To differentiate between these two, a number of flag bits are used in front of each Op-Code. The following table show the format of the instruction.

Op-Code								**Data**		
7	6	5	4	3	2	1	0	7	to	0

Op-Code

Bit 7 **Sequencer/PE Switch**

0 Sequencer

1 PE

Bit 6 – 5 **Input type**

00 Constant (Seq + PE)

01 Memory (Seq + PE)

10 Neighbour (PE only)

11 Neighbour + Seq (PE only)

Bit 4 – 0 **Op-Code**

Sequencer: 4 bit Op-Code (3-0), bit 4 is unused

PE: Bit 4

0 ACCU

1 STATUS/Logic

Bit 3 – 0

3 bit Op-Code

Table 4.3 Sequencer CPU Instruction Format

For the Op-Code part of the instruction:

Bit 7 indicates whether this instruction is for Sequencer CPU or PE.

Bit 6 – 5 indicates the input data type and where the data is from. This is same as Bit 9 – 8 used in PE's instruction.

Bit 4 – 0 is the "actual" Op-Code. If this instruction is for Sequencer CPU, then only Bit 3 – 0 are used. If this instruction is for PE, then all 5 bits are used with Bit 4 being a flag. This is same as Bit 14 – 10 used in PE's instruction.

The following table is the instruction set used by the Sequencer CPU.

SEQUENCER CPU OPCODES

OPCODE	INSTRUCTION	INPUT	DESCRIPTION	OPERATION
0	NOP	-	No Operation	-
1	STORE	imm	Store ACCU at input address	RAM(imm) <- ACCU
2	LOAD	imm	Load constant from memory address into ACCU	ACCU <- RAM(imm)
3	ADD	imm or addr	ADD to ACCU (without Carry in)	ACCU <- ACCU + <data>
4	NOT	-	Invert ACCU	ACCU <- $\overline{ACCU}$
5	AND	imm or addr	AND with ACCU	ACCU <- ACCU · <data>
6	OR	imm or addr	OR with ACCU	ACCU <- ACCU + <data>
7	BRA	imm	Branch Ahead	pc <- pc + K
8	-	-	-	-
9	-	-	-	-
A	-	-	-	-
B	BRR	imm	Branch on Ready	IF (rdy==1) pc <- K
C	BRC	imm	Branch on Carry	IF (C==1) pc <- K
D	-	-	-	-
E	BRZ	imm	Branch on zero	IF (Z==1) pc <- K
F	BRN	imm	Branch on negative	IF (N==1) pc <- K

Table 4.4 Sequencer CPU Opcodes

4.3 Processes Element (PE)

Each PE is an execution unit and corresponds to each pixel of the image. Multiple PEs are connected to form a PE network and they are controlled by a single sequencer CPU. The same PE internal circuit is repeated multiple times. In reality, a silicon chip has a finite number of transistors, and we would like to fit as many PEs into the chip as possible. It is therefore important to have a simple but highly efficient circuit design.

Figure 4.1 the schematic of the PE internal circuit. It is made up of five regions – ALU, Input / Output, Instruction Bus, Internal Memory, and Status Register. A PE does not have a CU. It relies on the sequencer CPU to provide both instruction and data. The following section covers each of these five regions.

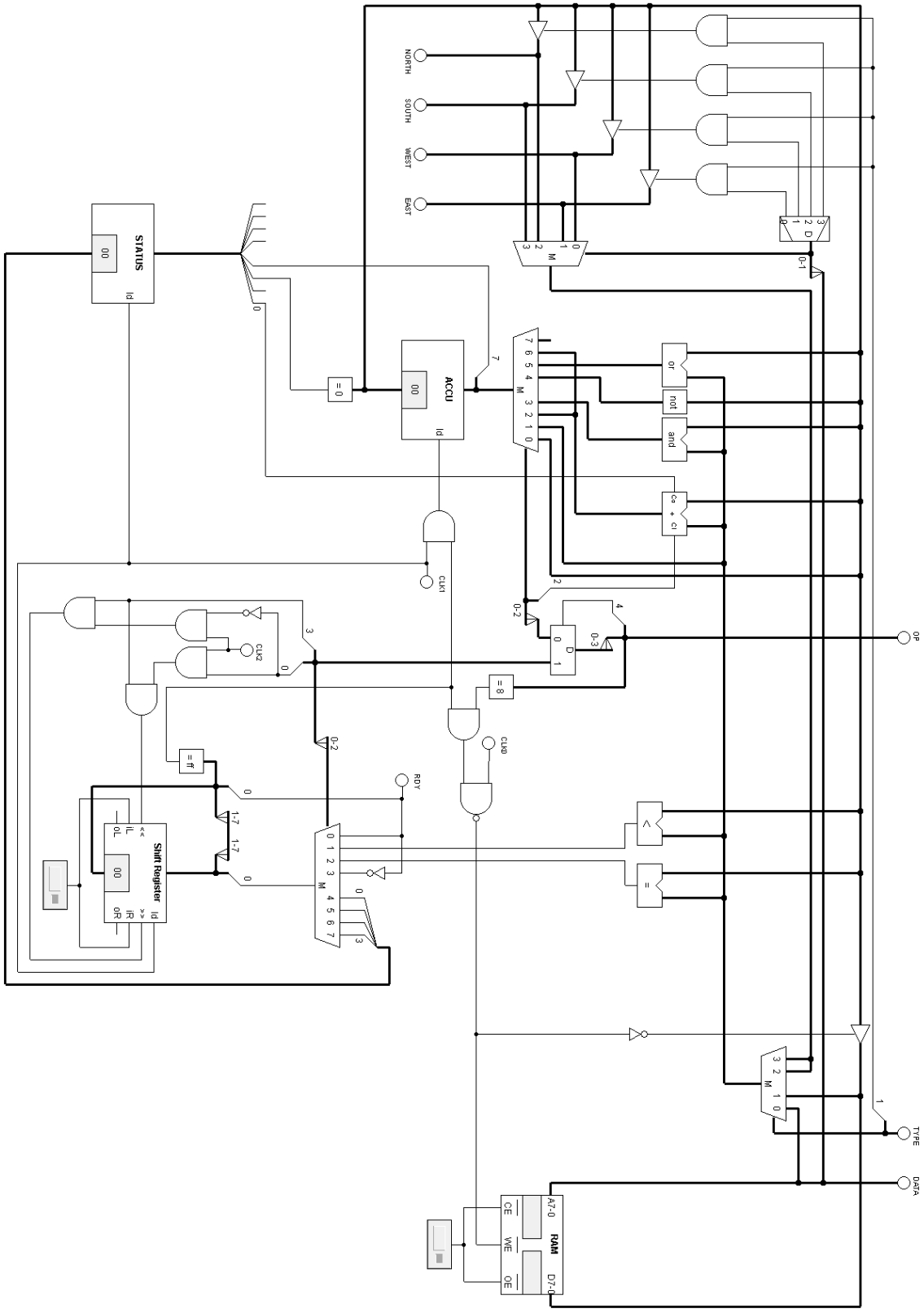


Figure 4.1 PE Internal Circuit

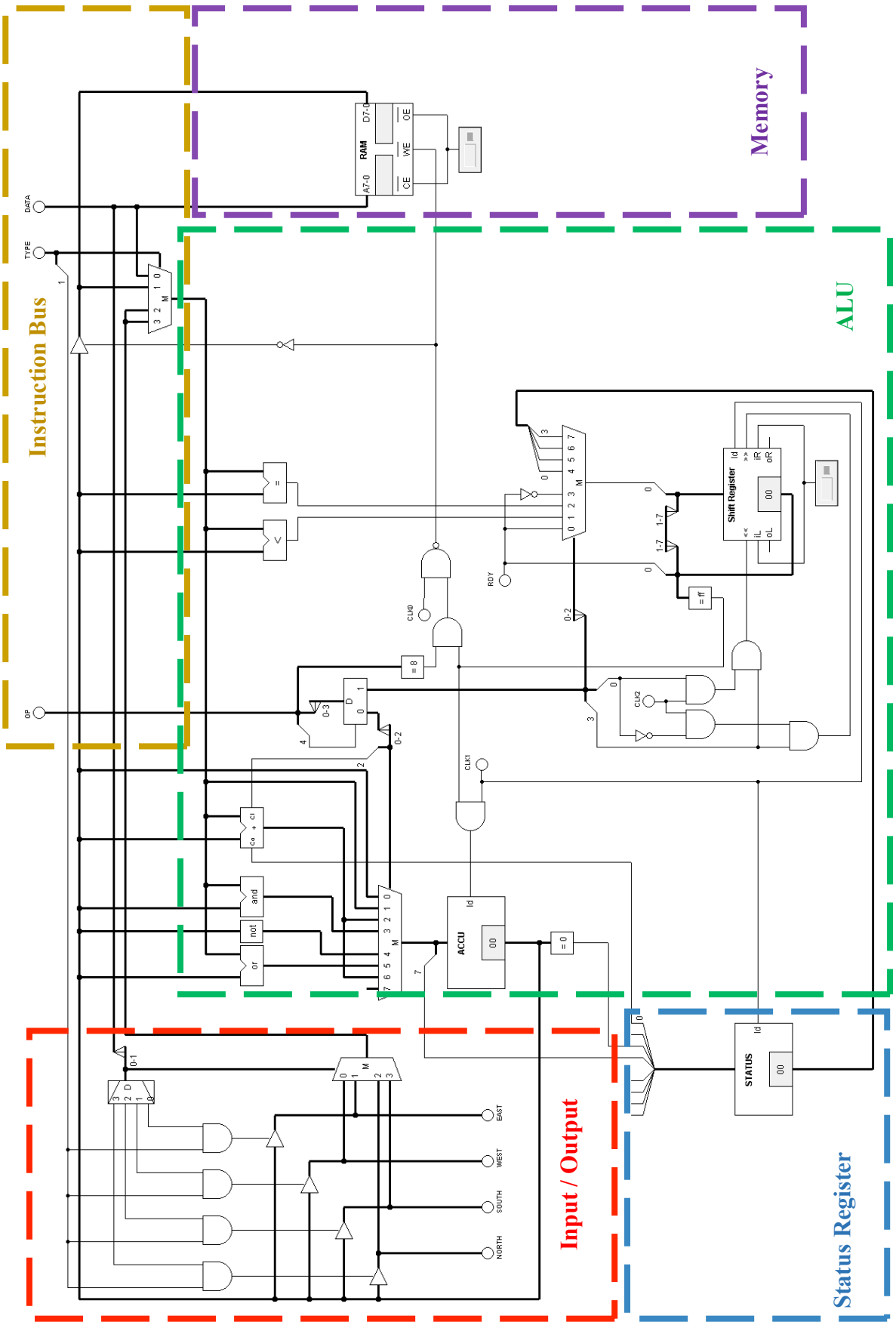


Figure 4.2 PE Internal Circuit

4.3.1 Arithmetic Logic Unit

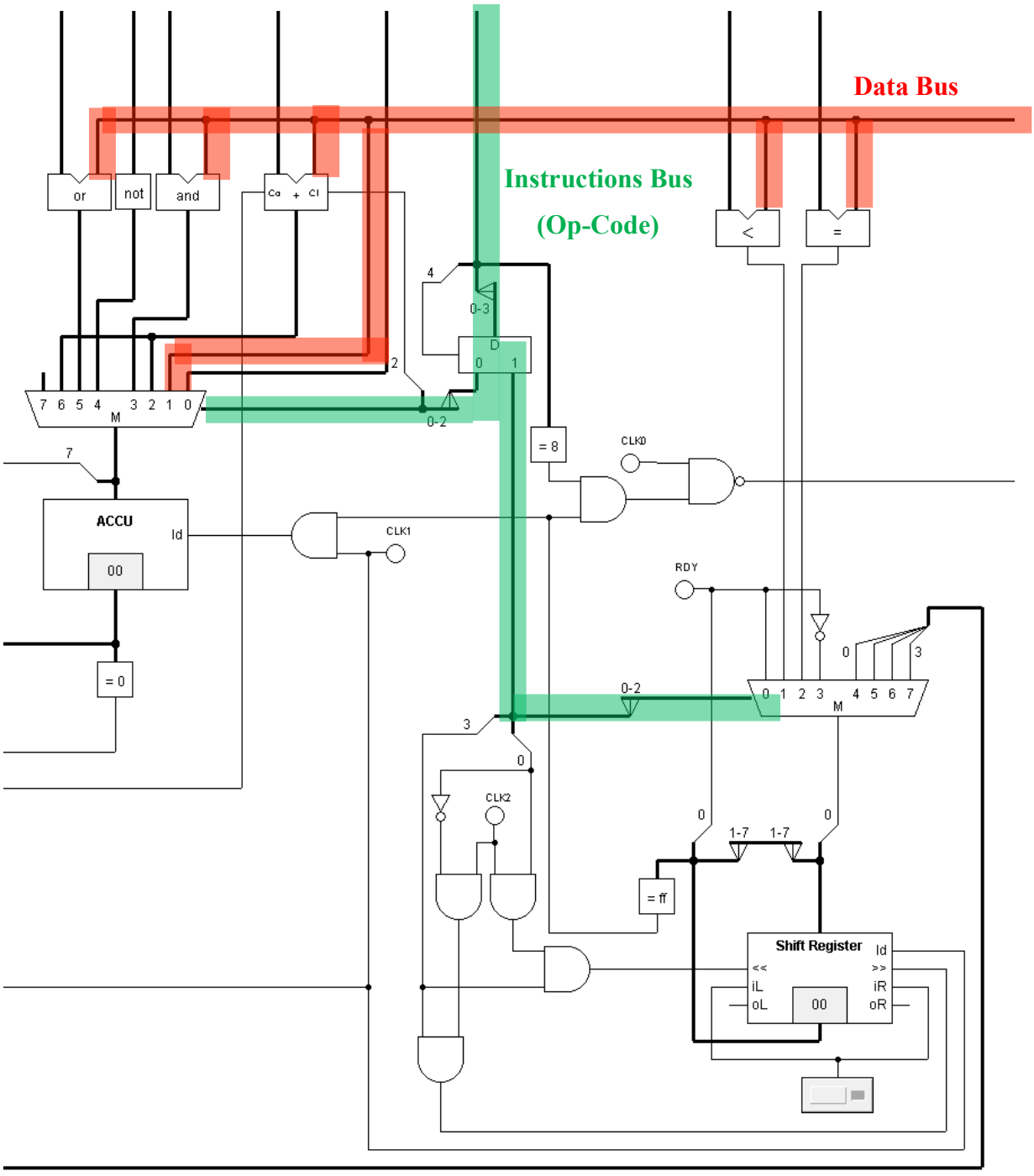


Figure 4.3 PE - ALU

The ALU is the heart of a PE and it is made up of two multiplexers, an accumulator and a shift register. The select lines of the both multiplexers are connected to the instruction bus.

Data coming from both external world and accumulator passes through different function units, which then feeds into the multiplexer. It has six most basic function units:

- Full Adder
- AND
- NOT
- OR
- Less Than
- Equal

Unlike a general-purpose microprocessor, this PE does not have some complicated function units such as multiplications. There are three major reasons behind this. First of all, these complicated function units take up a large amount of transistors, which results in less PEs fitting on a single silicon chip. Additionally, this microprocessor is built for image processing, and it does not require these complicated function units. Finally, a lot of functions can be achieved by combining these basic functions. Here are some examples.

4.3.1.1 Example 1: Negate

A negative decimal number is represented by the minus sign attached to the left of the number. However numbers are stored in binary inside a computer and there is no such thing as a minus symbol. The way to indicate a binary number being negative is by setting the most significant bit (MSB). A signed binary starts with MSB equal to 0 represents a positive number while MSB equal to 1 represents a negative number.

011	010	001	000	111	110	101	100
3	2	1	0	-1	-2	-3	-4

Two's complement is used to convert from positive to negative or vice versa. The following assembly code shows the steps to perform this using this PE. (X is the data)

```
NOT X
ADD 01
```

The value stored in the accumulator after these operations will be the negative value of X.

4.3.1.2 Example 2: Subtraction

Same as decimal number, subtracting a number can be treated as adding a negative number. The following assembly code shows the steps to perform this using this PE.

```
NOT X
ADD 01
ADD Y
```

4.3.2 Input / Output (Data Bus & Interconnections)

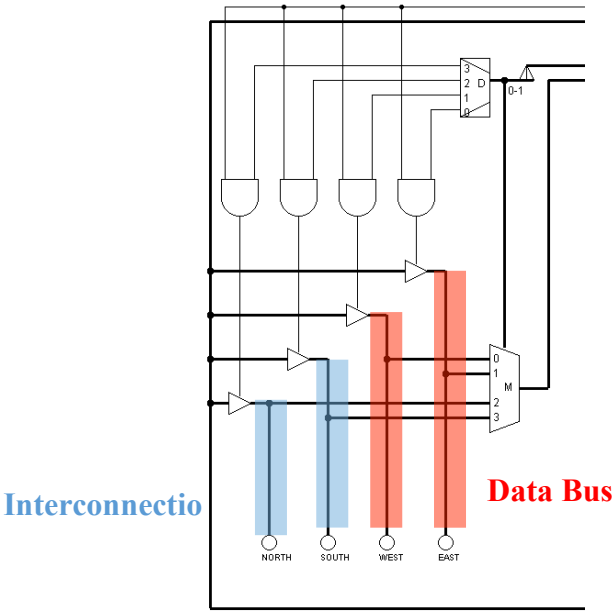


Figure 4.4 PE - Input / Output

The I/O control is important for data shifting. Every PE has two way communications with its four surrounding neighbours. In the horizontal direction, a PE exchanges data with its neighbour using the data bus. In the vertical direction, PE transfers data between its neighbours using the interconnection bus. The data flow direction of each bus is controlled by the combined action of tri-state gates, a decoder, AND gates and a multiplexer. In SIMD, a data shifting command causes all PEs to shift data into their neighbours. For example, if we want to shift data to the right (east), then all PEs enable their "east" tri-state gate. Data can flow out of PEs via "east" bus to their east neighbours. At the same time, every PE's multiplexer is switched to the "west" bus, which can now accept data coming from their west neighbours.

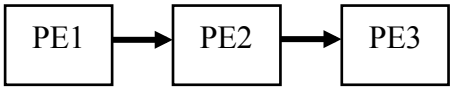


Figure 4.5 PE - Data Propagation from Left to Right

4.3.3 Instruction Bus

Figure 4.6 PE - Instruction Bus

All PEs are connected to the same instruction bus in parallel. PEs do not hold any instructions internally. They receive instructions from the external world via the instructions bus. As shown in figure 3.8, there are three pins connected to the Sequencer CPU. The “OP” pin is used for receiving Op-Code. The “TYPE” pin gets a 2-bit value, which indicates the type of the data and where the data is from – either from its neighbour or from Sequencer CPU. And finally the “DATA” pin accepts the data from the Sequencer CPU.

4.3.4 Internal Memory

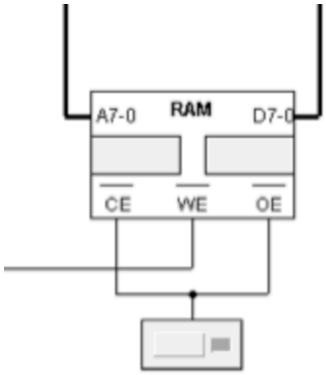


Figure 4.7 PE - Internal Memory

Each PE has a small memory space for temporary data storage. This memory can be used to store intermediate results, address pointers, as well as various data structures such as queue and stack.

4.3.5 Status Register

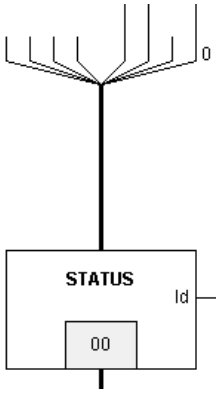


Figure 4.8 PE - Status Register

Status Register is another important component of a PE. This is an 8-bit status register, which can hold up to eight status flags. The definition of each status bit is stated in Table 4.5. Currently there are only 3 flags are used – Negative, Zero, and Carry bit.

Bit	7	6	5	4	3	2	1	0
Flag	-	-	-	-	N	Z	V	C

C	Carry bit
V	Overflow (Currently not implemented)
Z	Zero
N	Negative

Table 4.5 PE - Flags

Having a dedicated status register for each PE is very important in SIMD design. A PE can now record and determine its state by setting and checking its status flags.

4.4 Sequencer CPU

The sequencer CPU is the command centre of all PEs. It provides input data and instructions to the PEs. It also collects results from PEs at the end of the process. Figure 4.9 is the schematic of the Sequencer CPU.

Unlike the internal circuit of a PE, a Sequencer CPU is made up of five regions – CU, ALU, status register, memory, and clock. We are going to look at each region in the following sections.

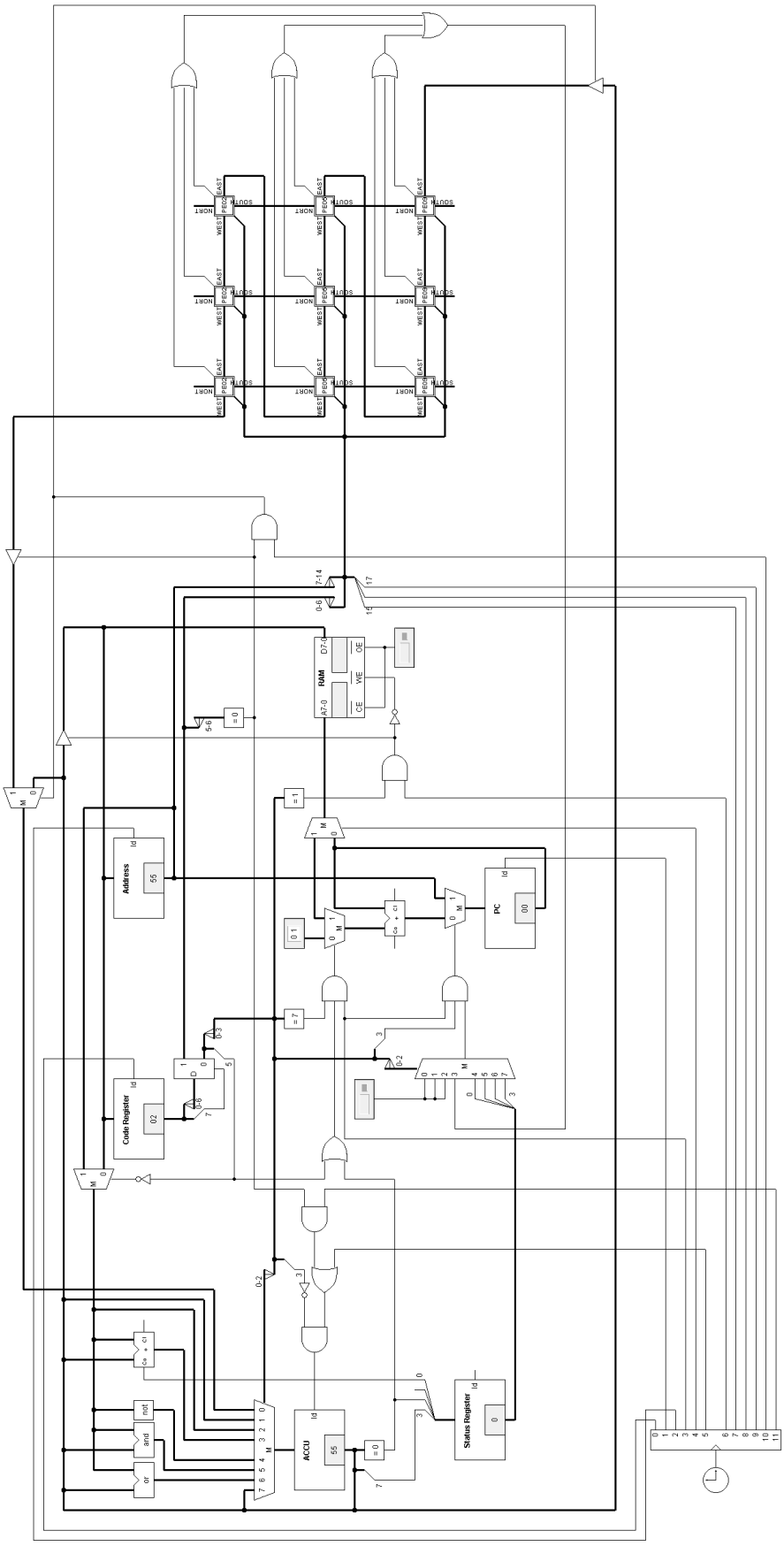


Figure 4.9 Sequencer CPU

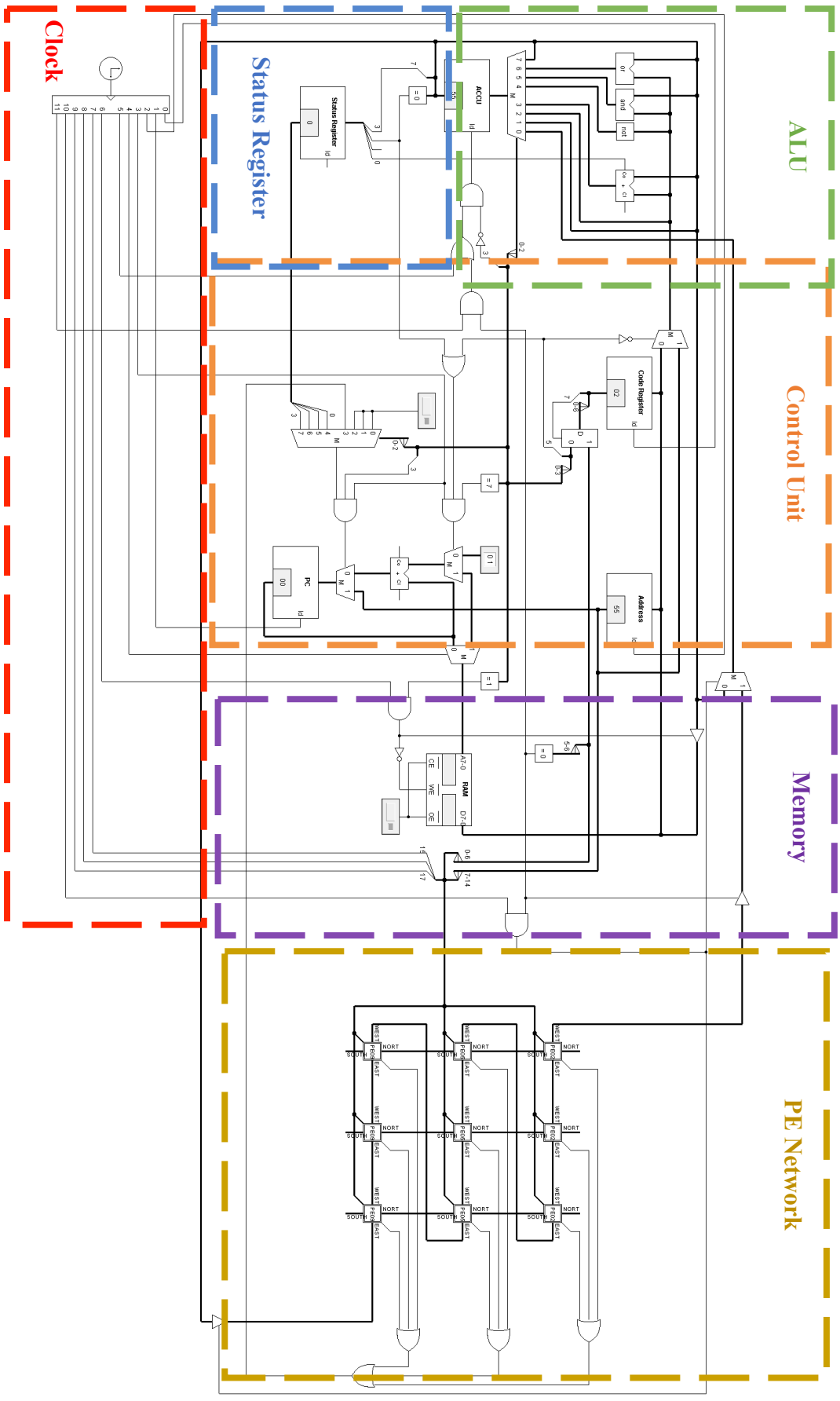


Figure 4.10 Sequencer CPU

4.4.1 Control Unit (CU)

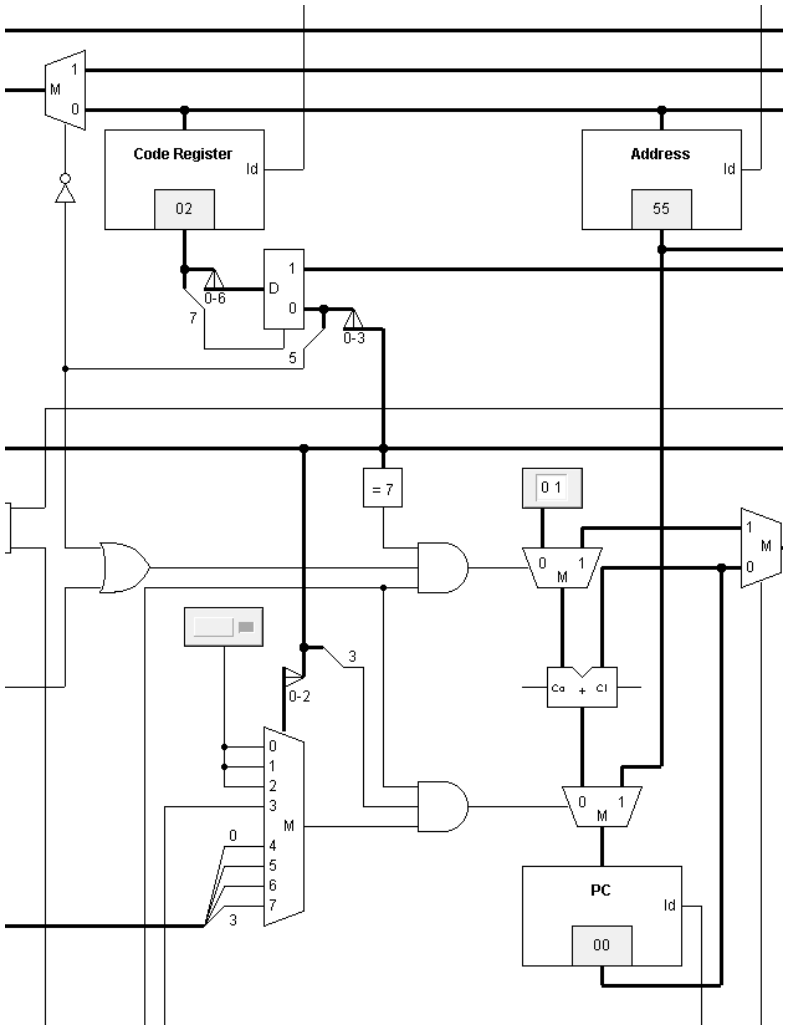


Figure 4.11 Sequencer CPU - Control Unit

The CU is the main difference between a PE and the sequencer CPU. This CU controls both the sequencer CPU and all PEs. In normal mode, for every rising clock pulse, it increments its program counter by one and reads the next instruction store in the memory. We have also added two branching operations to our design – conditional branching and unconditional branching. In conditional branching, PC moves to a given memory address only when a certain logic condition is satisfied. In unconditional branching, PC moves to a given memory address regardless of any conditions.

4.4.2 ALU

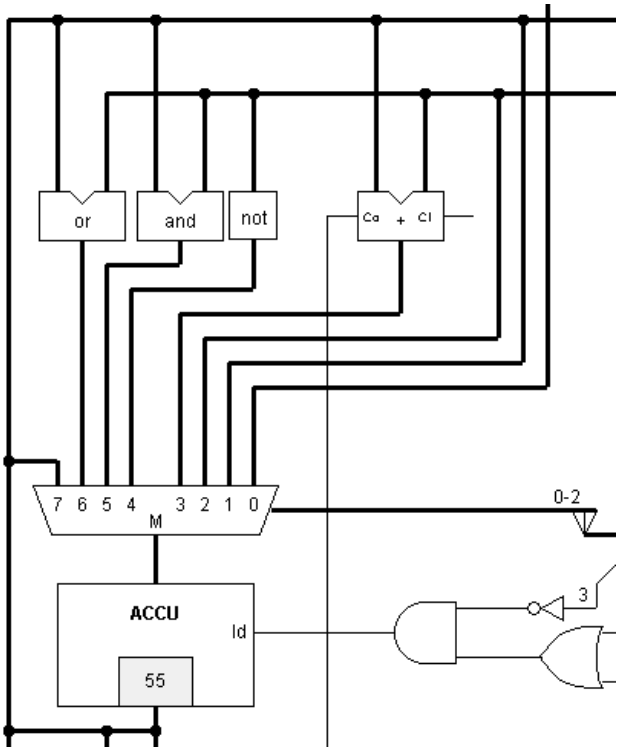


Figure 4.12 Sequencer CPU - ALU

The ALU used in sequencer CPU is much simpler than the one used in PE. This ALU is mainly used for logical operations since the sequencer CPU does not process any pixels – only the PEs are used to process an image.

4.4.3 Status Register

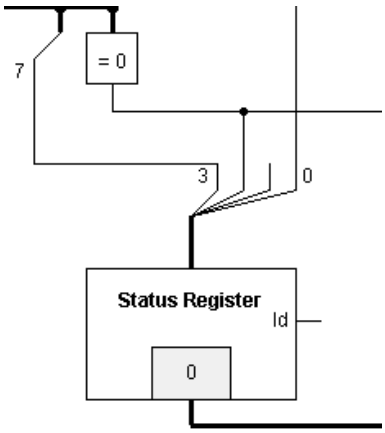


Figure 4.13 Sequencer CPU - Status Register

A 4bit status register is also used to record the current state of the sequencer CPU. Only three flags are currently used – Carry Bit, Zero, Negative. The definition of each flag is shown in the following table.

Bit	3	2	1	0
Flag	N	Z	-	C
S Negative Z Zero C Carry Bit				

Table 4.6 Sequencer CPU - Flags

4.4.4 Memory

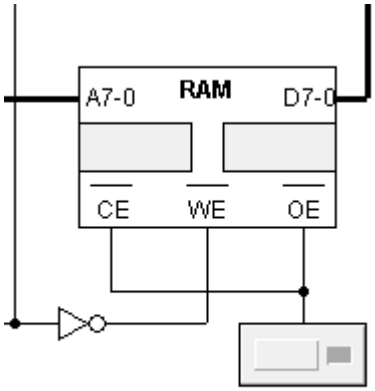


Figure 4.14 Sequencer CPU - Memory

Our image processor is based on von Neumann architecture. Only one memory unit is used in the sequencer CPU to store both Op-Code and Data/address. Inside the memory, every odd memory address is used to store Op-Code and every even address is used for data/address. The sequencer CPU needs to access the memory twice to obtain both the Op-Code and the data/address. During the first clock cycle, it reads the Op-Code from the memory and stores it into a code register. It then increases the Program Counter by one. During the second clock cycle, the sequencer CPU reads the data/address and stores it in the address register. After these two clock cycles, the sequencer CPU can now perform an action according to the instruction. Once the action is completed, the sequencer CPU visits the memory again and repeats the same process.

4.4.5 Clock

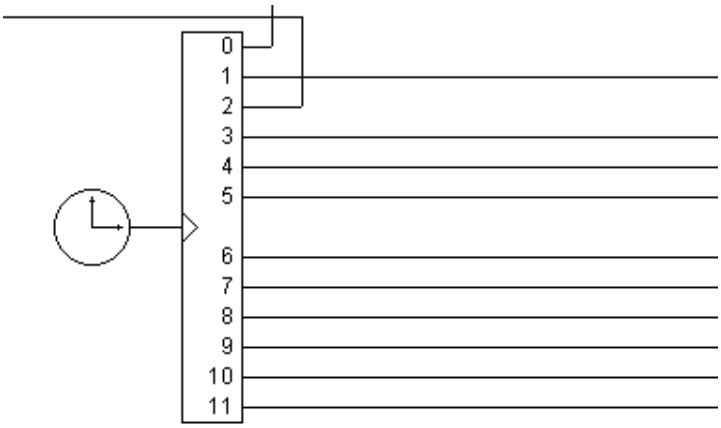


Figure 4.15 Sequencer CPU - Clock

The same clock is used for both Sequencer CPU and PEs. This avoids clock drift due to the use of multiple clocks. Clock pulse 7 to 10 are used by all PEs. Since SIMD is a synchronised architecture, it is important to have all PE are connected to the same clock signal in parallel. All other clock pulses are used by the various components in the Sequencer CPU.

4.4.6 PE Network

All PEs are interconnected to form a network. They are connected to the Sequencer CPU via two buses – data bus and instruction bus.

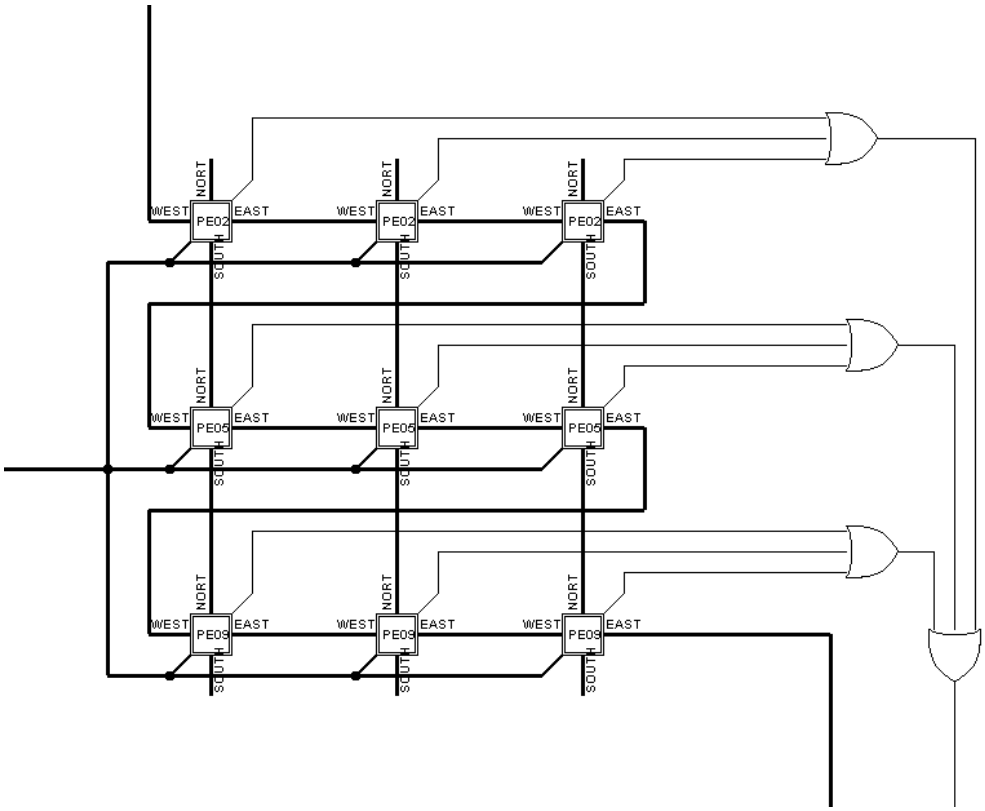


Figure 4.16 PE Network

Data bus connects all PE in series. This is drawn in red on the figure below. The red arrow shows the data flow direction along the data bus. Data coming from sequencer CPU enters the PE network at the bottom right and exits at the top left. Inside the network, data bus is also used for data transfer between neighbours in horizontal direction.

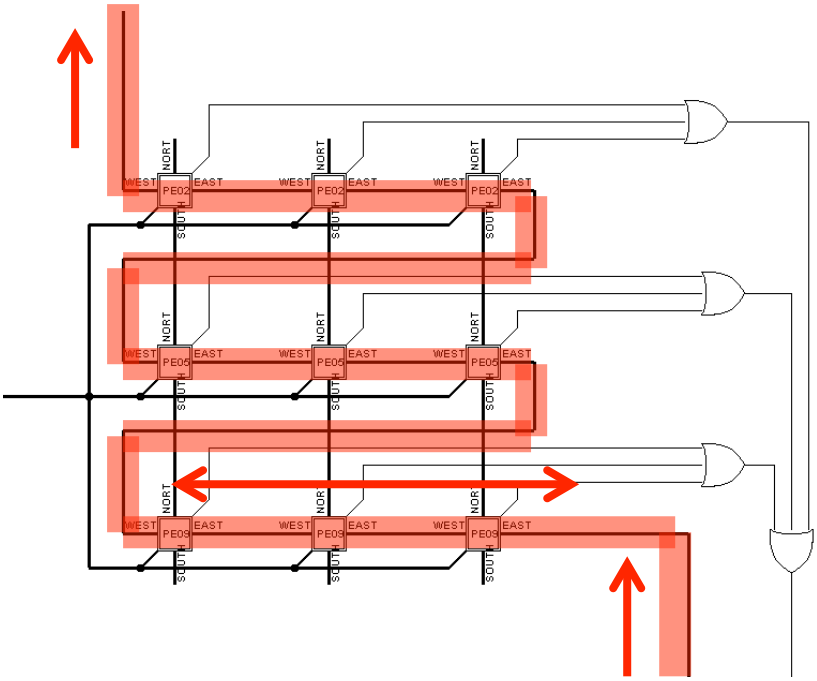


Figure 4.17 Data Bus

The instruction bus, which is drawn in green, connects all PEs in parallel. All PEs receive the same instruction simultaneously. This bus is a one-way bus and it sends the following to the PE network:

- PE Op-Code
- Data/address switching flag
- Data
- Clock signals

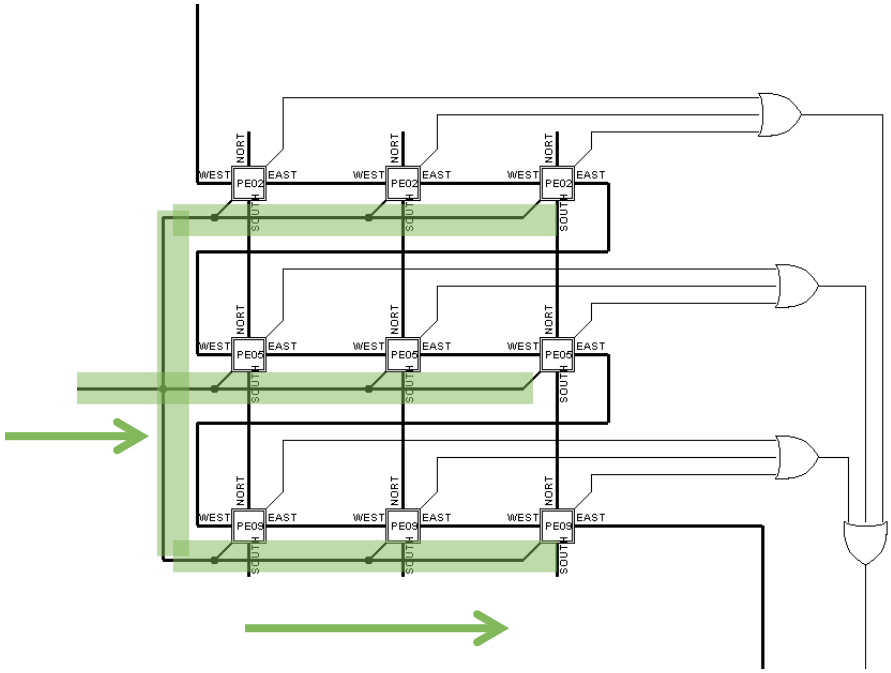


Figure 4.18 Instruction Bus

The blue lines are interconnections between PEs. These are two way buses which allows PEs to exchange data between its neighbours in the vertical direction. The use of interconnections can significantly increase the overall performance because data can move between PEs without going back to the Sequencer CPU.

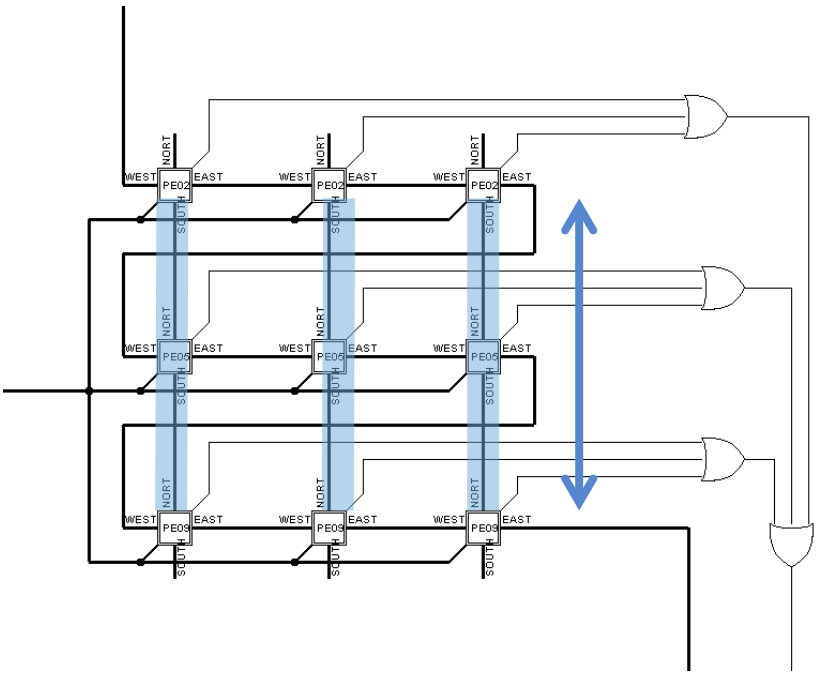


Figure 4.19 Interconnection Bus

4.4.7 Conditional Statements & Loops

It is a bit tricky to perform conditional statements and loops using SIMD architecture. In a SISD architecture, only a single piece of data is tested in the boolean statement. Based on the result, it will only execute one part of the statement. However, in a SIMD architecture, different PE holds different value, which means different PEs enter different parts of the statement and behave differently. This ends up with unpredictable and uncontrollable behaviour. There are two check bits we introduced for the conditional statements and loops.

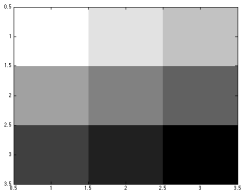
4.4.7.1 Conditional Statements

The first check bit we introduced is called "SET BIT". This bit is set to 1 if the conditional check returns true, otherwise it is set to 0.

Let us have a look at an if-else statement. If-else statement is the most basic conditional statement. Other conditional statements and loops are based on if-else statement.

Given an image having the following values:

$$\begin{bmatrix} 09 & 08 & 07 \\ 06 & 05 & 04 \\ 03 & 02 & 01 \end{bmatrix}$$



If the value is greater than 0x05, set it to 0xFF. Otherwise set it to 0x01.

Assembly

```
LESSTHAN 5
STATUS SET
LOAD FF
STATUS INVERT
LOAD 01
STATUS RESET
```

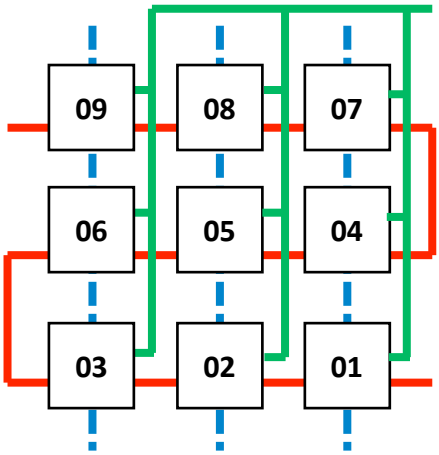
Step1

If (ACC > 0x05)

ACC = 0xFF

Else

ACC = 0x01



Step2

→ If (ACC > 0x05)

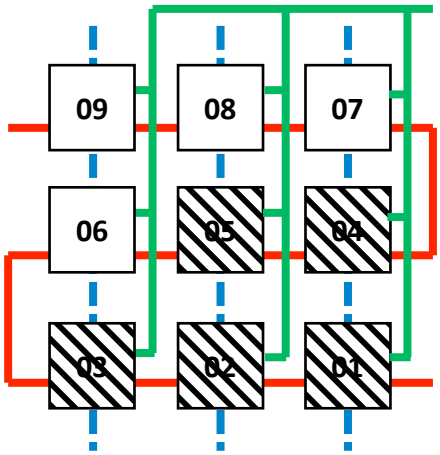
ACC = 0xFF

Else

ACC = 0x01

ON Status = 0x01

OFF Status = 0x00



Step3

If (ACC > 0x05)

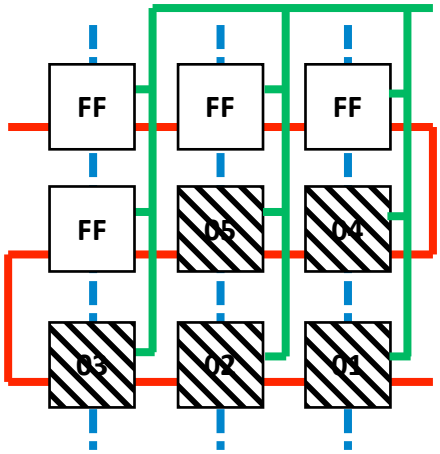
→ ACC = 0xFF

Else

ACC = 0x01

ON Status = 0x01

OFF Status = 0x00



Step4

If (ACC > 0x05)

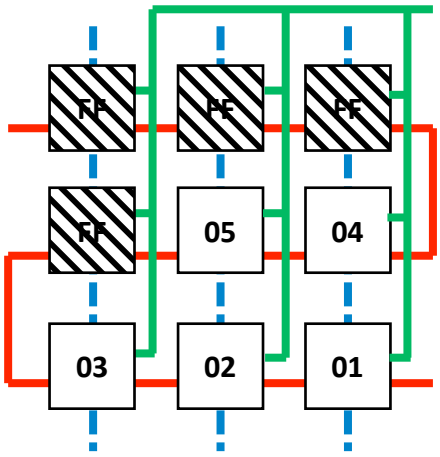
ACC = 0xFF

→ Else

ACC = 0x01

ON Status = 0x01

OFF Status = 0x00



Step5

If (ACC > 0x05)

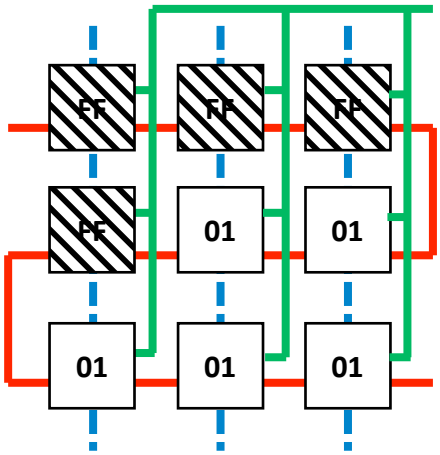
ACC = 0xFF

Else

→ ACC = 0x01

ON Status = 0x01

OFF Status = 0x00



Step6

If (ACC > 0x05)

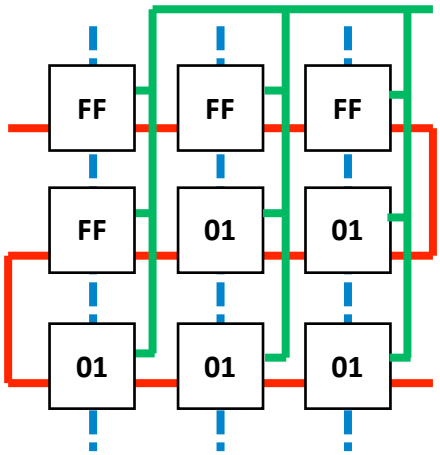
ACC = 0xFF

Else

ACC = 0x01

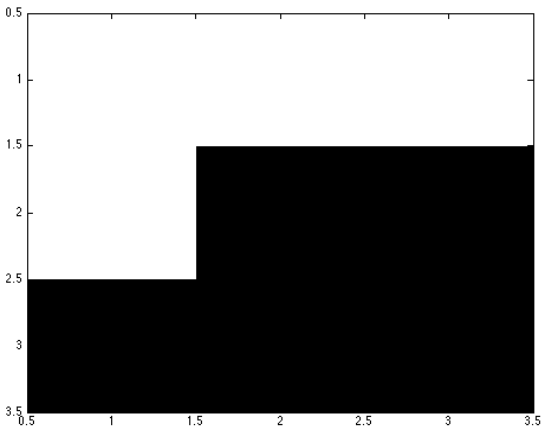
ON Status = 0x01

OFF Status = 0x00



Output

$$\begin{bmatrix} FF & FF & FF \\ FF & 01 & 01 \\ 01 & 01 & 01 \end{bmatrix}$$



4.4.7.2 Loops

There is another check bit called “READY BIT” was introduced for loops statements. This is used by the Sequencer CPU to determine if all PEs are ready for the next instruction. This is an important check to when performing loops. It takes different amount of time for different PEs to complete the same loop instructions. In order to keep the entire system synchronised, Sequencer CPU needs to wait for every PE to complete the loop before executing next instruction. The PE sets the signal wire to 0 when it completes a loop. Sequencer CPU knows all PE are ready for next instruction as soon as the OR value of all signal wires becomes 0.

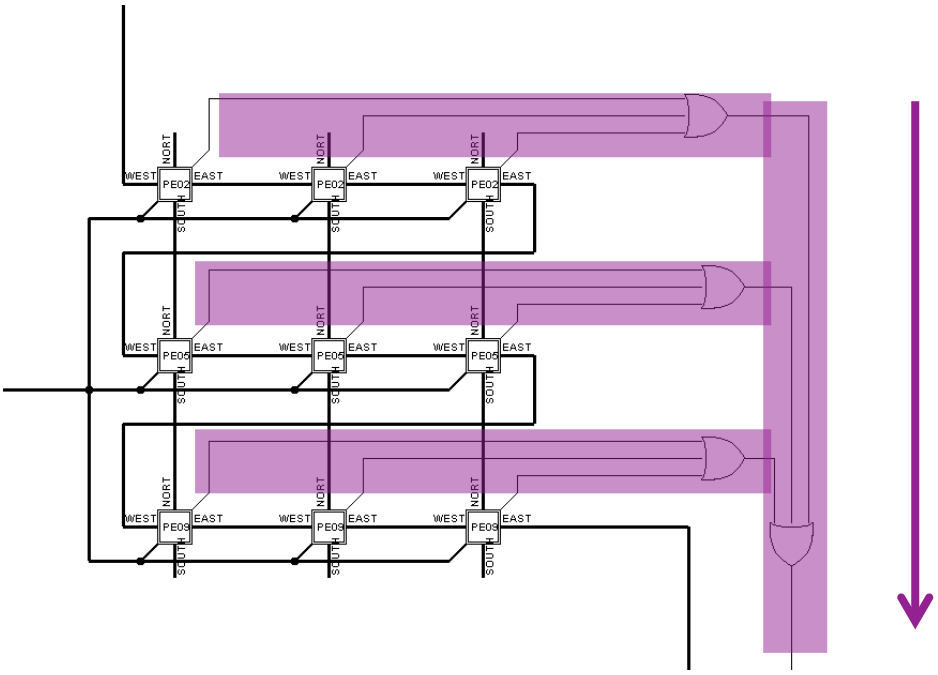


Figure 4.20 Ready Signal Bus

Chapter 5 Image Processing

Image processing is an important topic in computer science with a large number of applications. Applications such as contrast enhancement, lightening, image blurring, corner detection and edge detection are the most basic image processing applications. These applications are the building blocks of other advanced applications, such as artificial intelligence, 3D object reconstructions, and object recognitions.

The design and development of Retro and the SIMD circuit are now fully completed. It is extremely important to verify the correctness of a design in any engineering project. Test results provide evidence to support the whole design idea. This is also part of the reason why software simulation becomes more and more popular in many research areas. This chapter is going to simulate our image processor with a number of different image processing applications using Retro. The images we are going to use in the following examples are 5x5 greyscale images. Recall each pixel is processed by a PE and therefore a PE network with the size of 5x5 is needed to processes the images. Notice the size of the image can be scaled to any sizes as long the size of the PE network matches.

To provide a better visual representation, we have also modularised the Sequencer CPU. The (**HEX**) value of the current pixel data is displayed on the face of each PE. In some cases, a PE shows an undefined value "**- -**" as it reads a value from some unconnected wires. In reality, these wires are connected to ground and it should display "0" rather than "- -". In the following examples, we will treat "- -" as "0".

5.1 Summation

In a SISD microprocessor, the sum of all 25 pixel values is found by repeatedly add the current pixel value to the previous sum 25 times. However, the same operation can be achieved a lot faster using this SIMD microprocessor. It simply shifts values from right to the left 4 times and then from bottom to the top 4 times. The total number of operations reduces to 8. Imagine the given image has a size of MxN, the number of operations required would be M+N-2 when using this SIMD microprocessor instead of MxN operations needed on a SISD architecture.

Input

$$\begin{bmatrix} 01 & 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 & 01 \end{bmatrix}$$

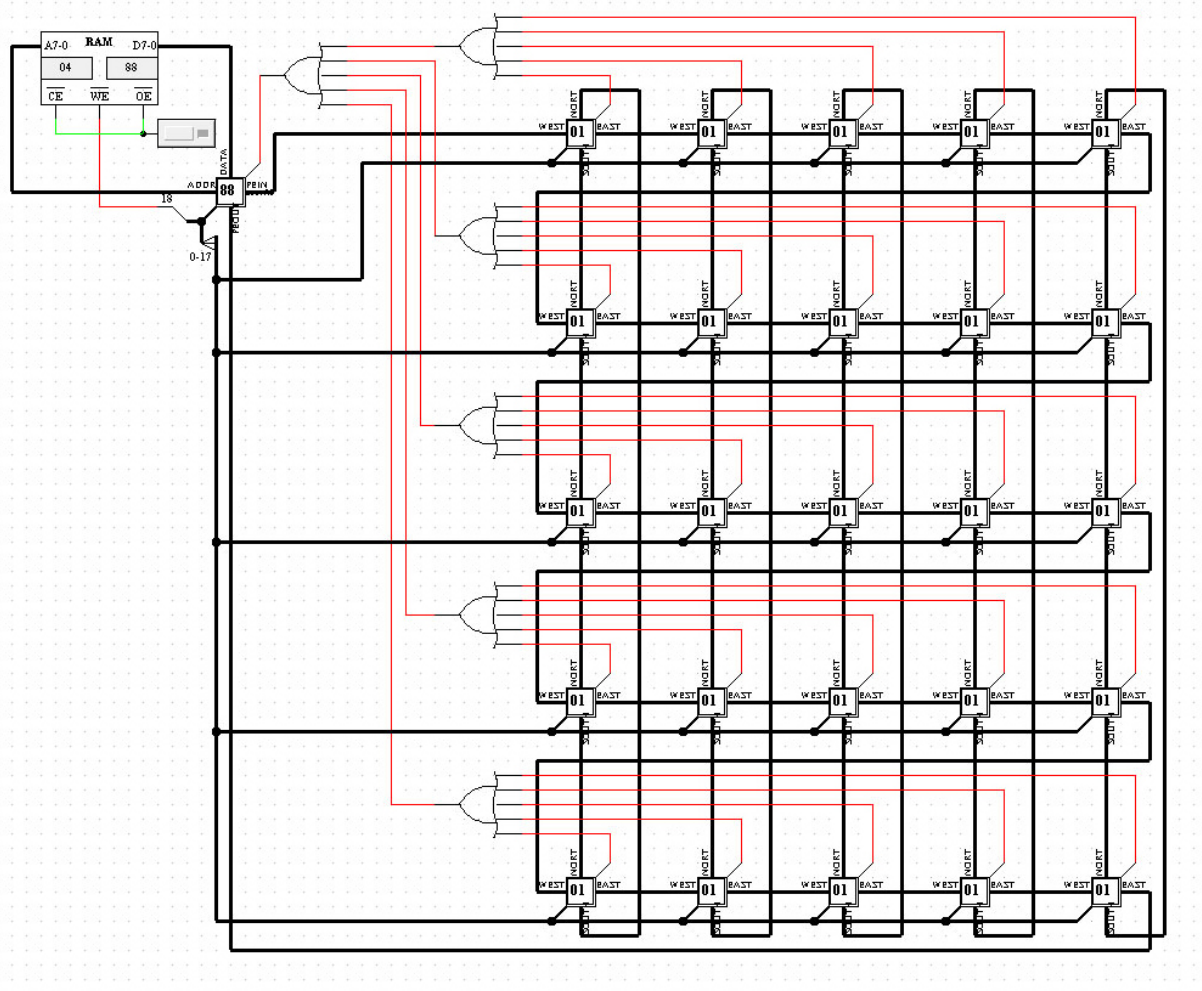


Figure 5.1 Sum - Input

Assembly Code

```
STORE F0
LOAD PE 01 (SHIFT LEFT)
ADD F0
LOAD PE 01 (SHIFT LEFT)
ADD F0
LOAD PE 01 (SHIFT LEFT)
ADD F0
LOAD PE 01 (SHIFT LEFT)
ADD F0

STORE F0
LOAD PE 03 (SHIFT UP)
```

```
ADD F0
LOAD PE 03 (SHIFT UP)
ADD F0
LOAD PE 03 (SHIFT UP)
ADD F0
LOAD PE 03 (SHIFT UP)
ADD F0
```

Expected Output

(01)hex = (01)oct

(01)oct * (25)oct = (25)oct = (19)hex

$$\begin{bmatrix} 19 & 00 & 00 & 00 & 00 \\ 19 & 00 & 00 & 00 & 00 \\ 19 & 00 & 00 & 00 & 00 \\ 19 & 00 & 00 & 00 & 00 \\ 19 & 00 & 00 & 00 & 00 \end{bmatrix}$$

Actual Output

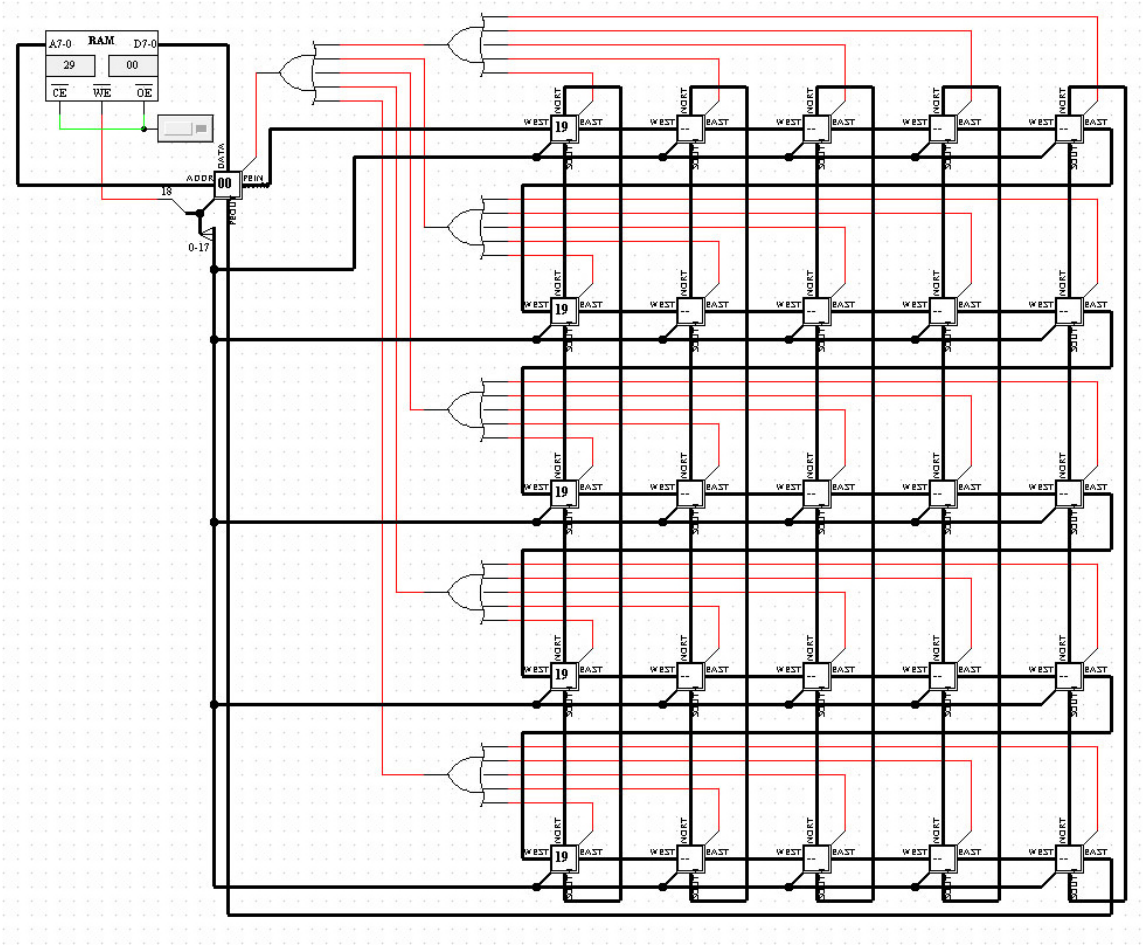


Figure 5.2 Sum - Output

5.2 Thresholding

Greyscale images are often needed to convert into a binary image. The process is a simple if-else statement. A threshold value needs to be defined first (in this example we set it to 0x80). Then all values lower than the threshold are set to 0x00 and the others are set to 0xFF.

Input

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 \\ 40 & 40 & 40 & 40 & 40 \\ 80 & 80 & 80 & 80 & 80 \\ C0 & C0 & C0 & C0 & C0 \\ FF & FF & FF & FF & FF \end{bmatrix}$$

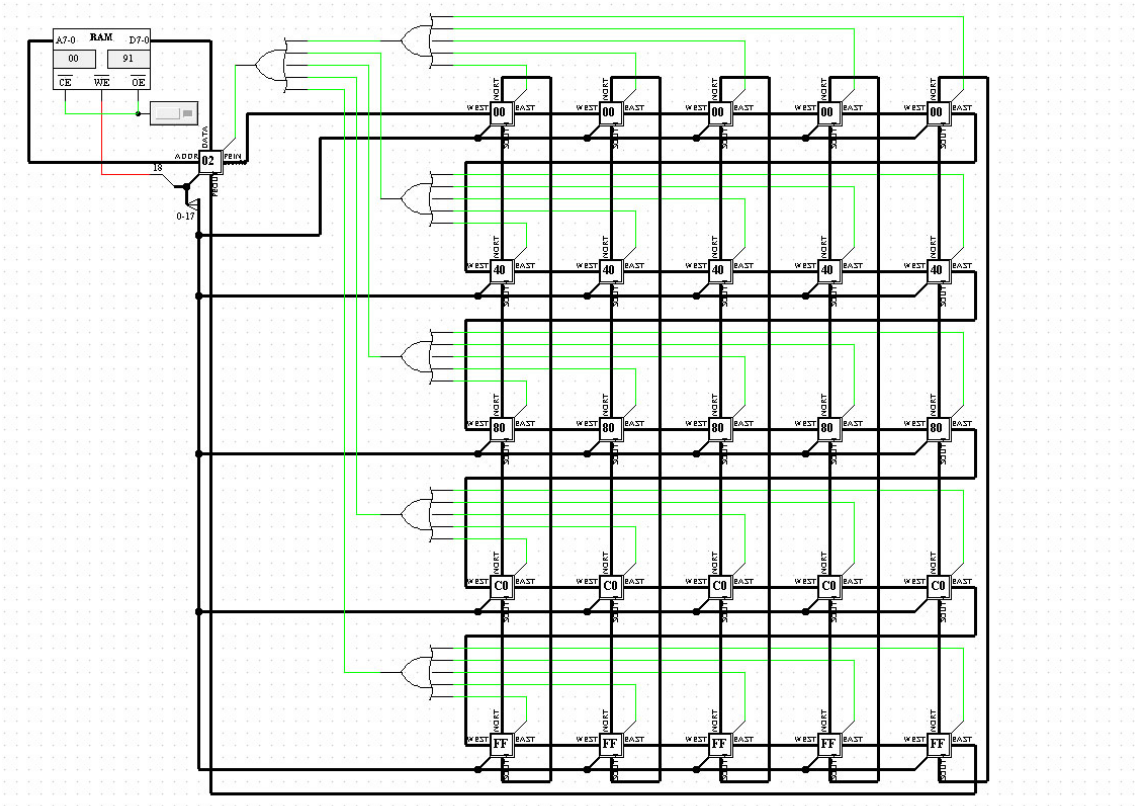


Figure 5.3 Thresholding - Input

Assembly

```
LESSTHAN 80
LOAD 00
STATUS_INVERT
LOAD FF
```

Expected Output

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 \\ FF & FF & FF & FF & FF \\ FF & FF & FF & FF & FF \\ FF & FF & FF & FF & FF \end{bmatrix}$$

Actual Output

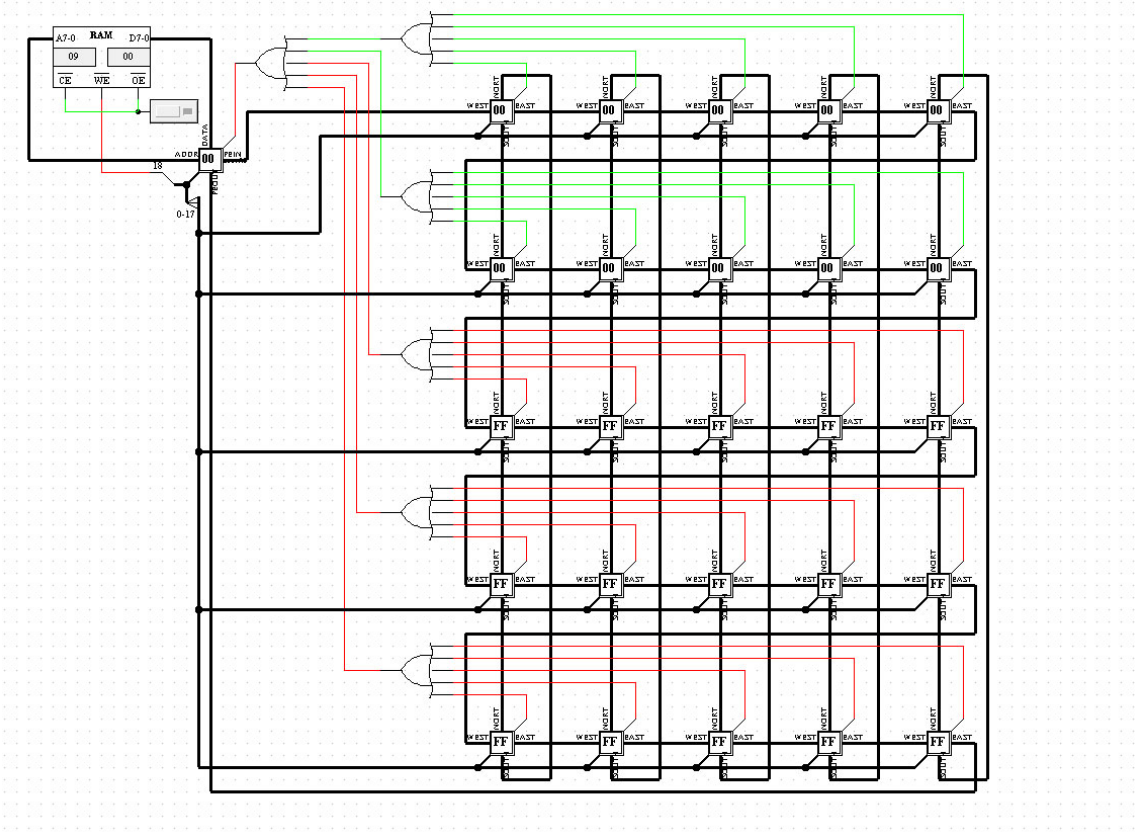


Figure 5.4 Thresholding - Output

5.3 Nested-If

This SIMD microprocessor is also capable to execute nested-if statements. Here is the example:

```
If (ACC < 0x07)
    If (ACC < 0x05)
            ACC = 0x01
    Else
            ACC = 0x05
Else
    ACC = 0x03
```

Input

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 \\ 02 & 02 & 02 & 02 & 02 \\ 05 & 05 & 05 & 05 & 05 \\ 09 & 09 & 09 & 09 & 09 \\ 10 & 10 & 10 & 10 & 10 \end{bmatrix}$$

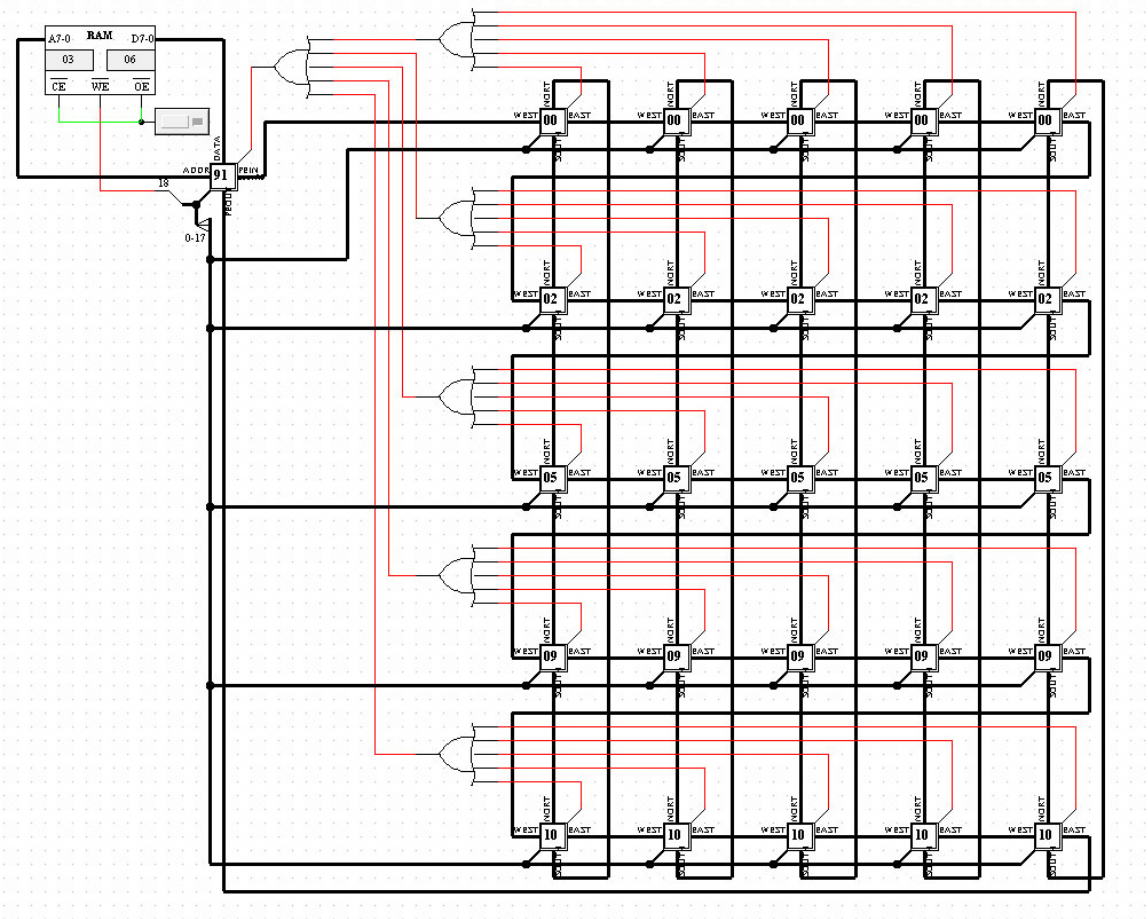


Figure 5.5 NestedIf - Input

Assembly

```
LESSTHAN 07
STATUS_SHIFTLEFT
LESSTHAN 05
LOAD 01 (CONST)
STATUS_INVERT
LOAD 02 (CONST)
STATUS_SHIFTRIGHT
STATUS_INVERT
LOAD 03 (CONST)
```

Expected Output

$$\begin{bmatrix} 01 & 01 & 01 & 01 & 01 \\ 01 & 01 & 01 & 01 & 01 \\ 02 & 02 & 02 & 02 & 02 \\ 03 & 03 & 03 & 03 & 03 \\ 03 & 03 & 03 & 03 & 03 \end{bmatrix}$$

Actual Output

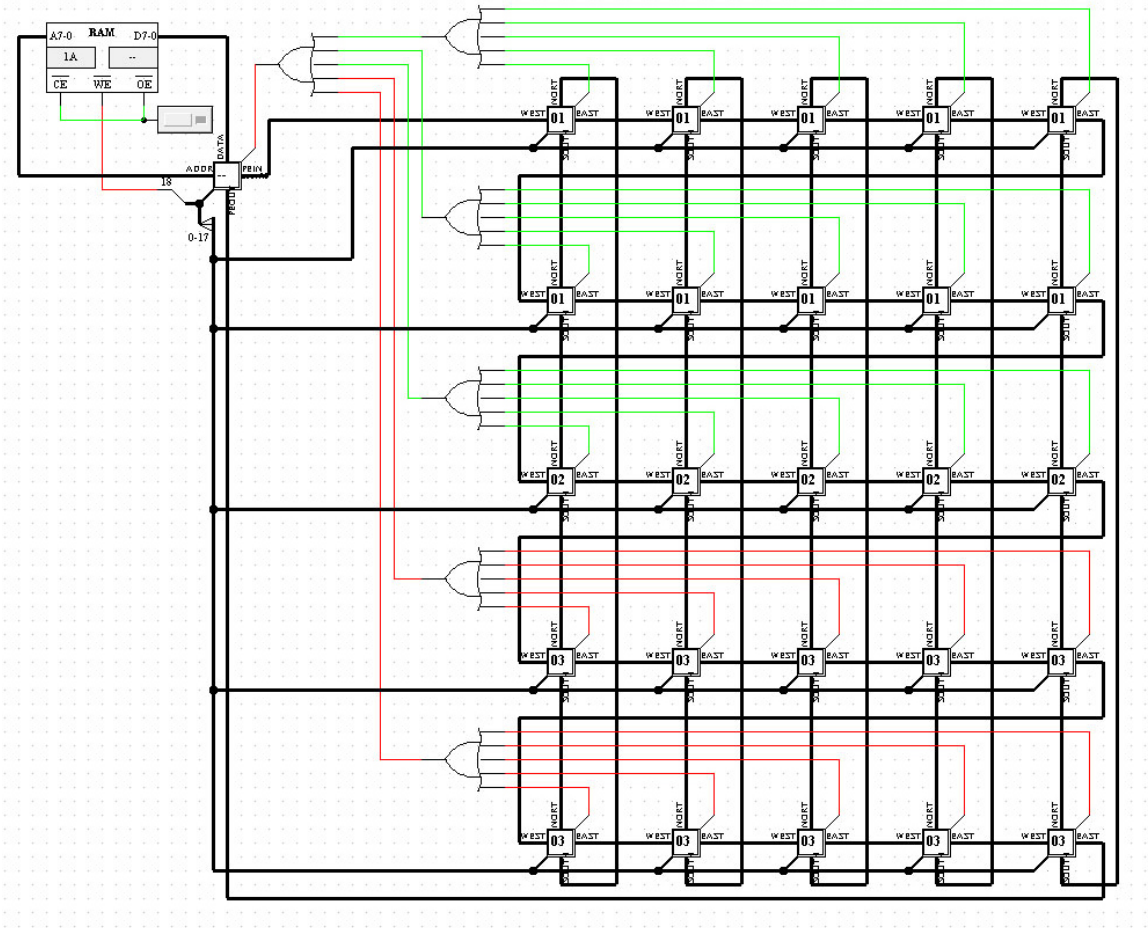
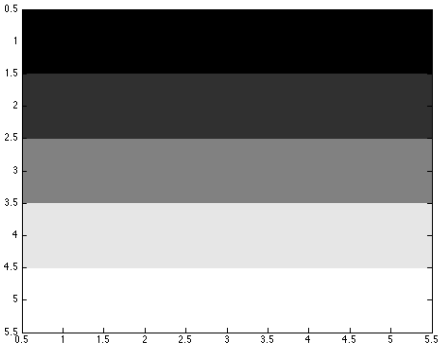
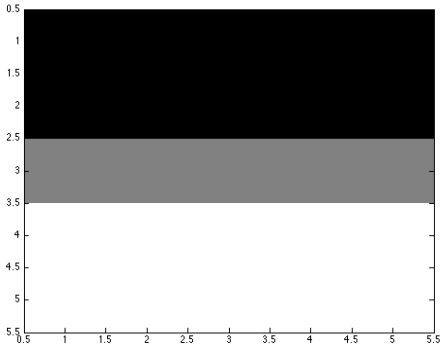


Figure 5.6 NestedIf - Output

Visual Comparison



Input



Output

5.4 While Loop

In this example, all pixels increase their value by one for every cycle until hitting 0x0F.

Input

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 \\ 02 & 02 & 02 & 02 & 02 \\ 05 & 05 & 05 & 05 & 05 \\ 09 & 09 & 09 & 09 & 09 \\ 10 & 10 & 10 & 10 & 10 \end{bmatrix}$$

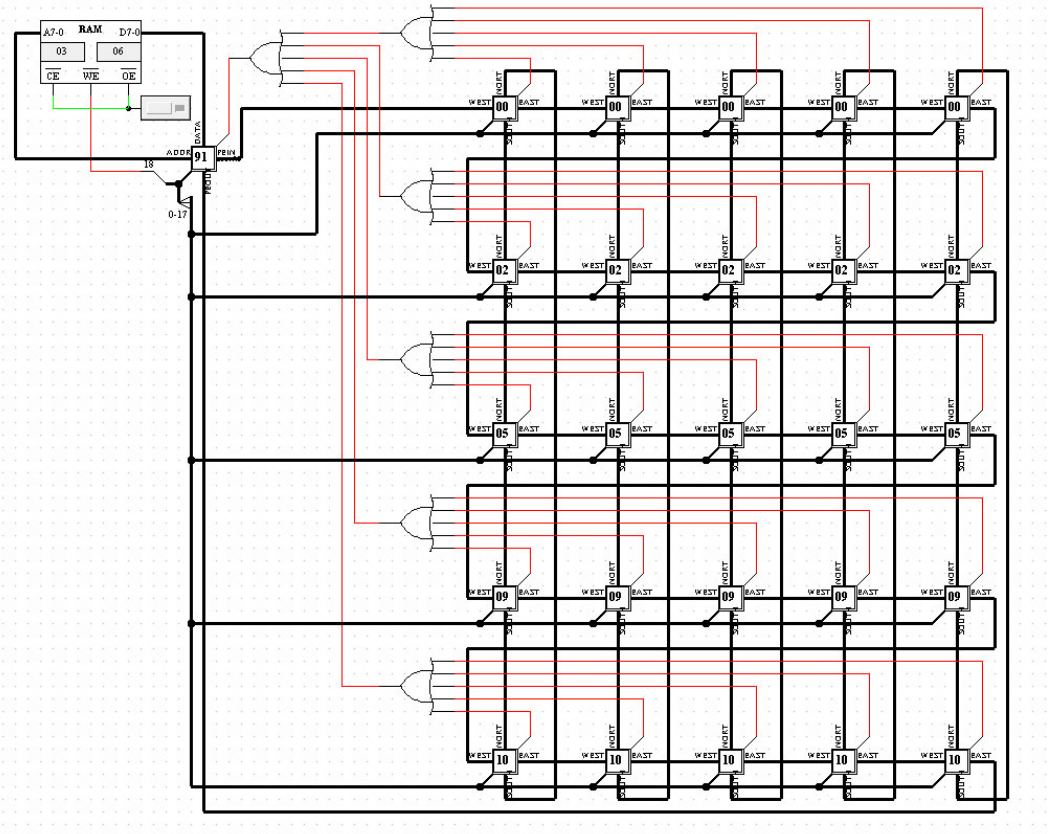


Figure 5.7 While - Input

Assembly

```
LESSTHAN 0f
ADD 01
BRR 00
```

Expected Output

$$\begin{bmatrix} 0F & 0F & 0F & 0F & 0F \\ 0F & 0F & 0F & 0F & 0F \\ 0F & 0F & 0F & 0F & 0F \\ 0F & 0F & 0F & 0F & 0F \\ 10 & 10 & 10 & 10 & 10 \end{bmatrix}$$

Actual Output

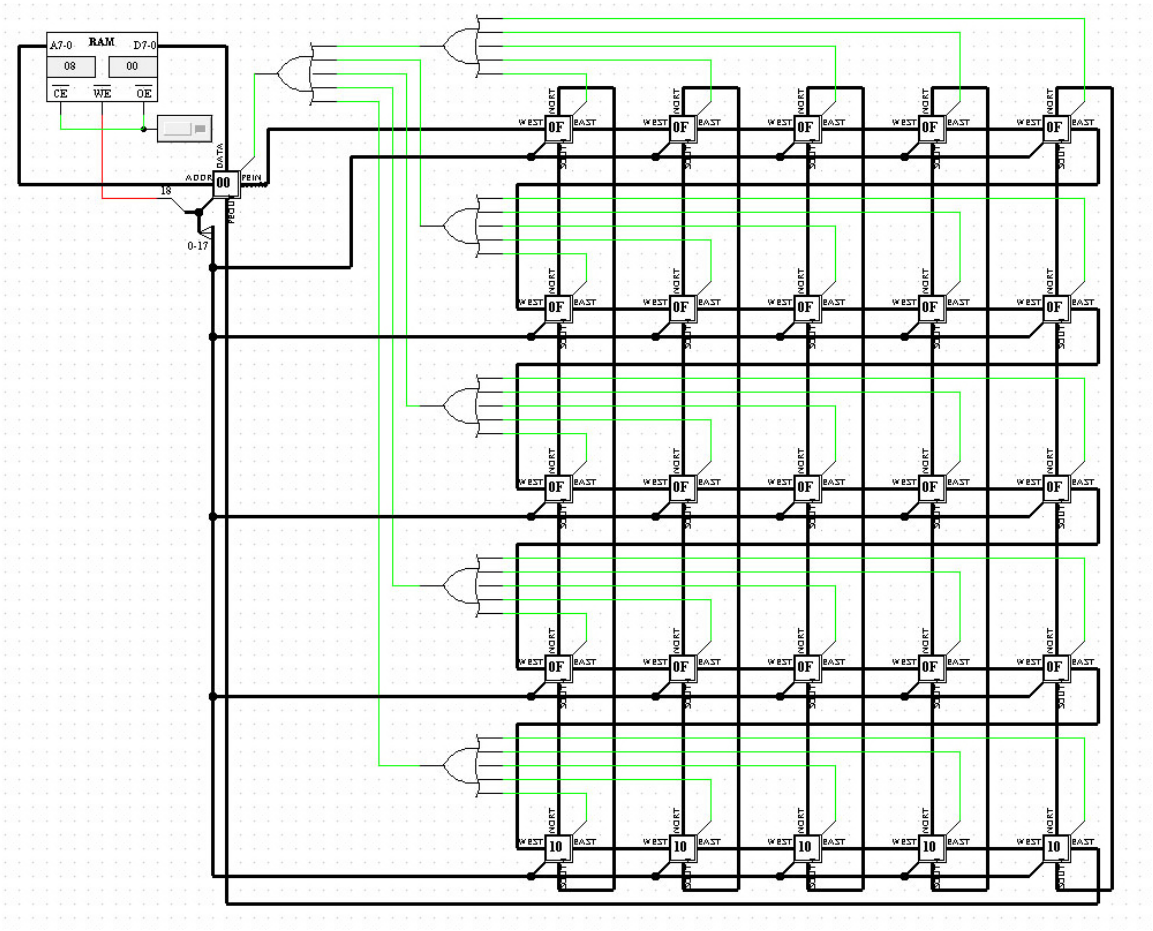
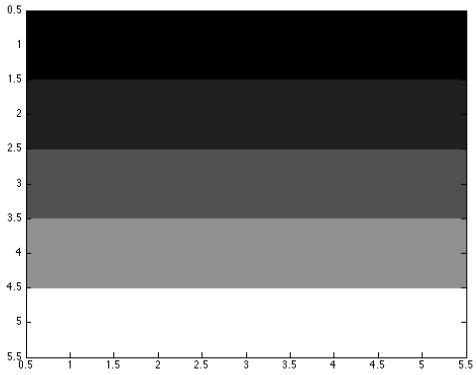
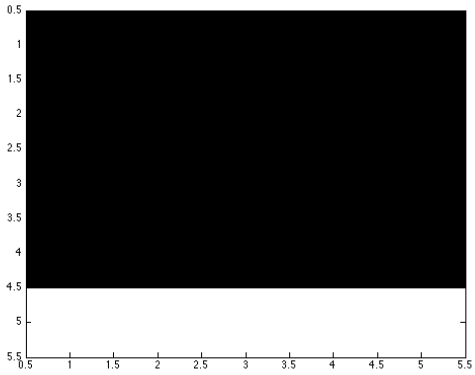


Figure 5.8 While - Output

Visual Comparison



Input



Output

5.5 Modifying Image Brightness

This example increases the brightness of the image. In this example, we are going to double every pixel value. Notice some of the results may be larger then the maximum value, 255 (FF), of a grayscale pixel. In this case, our image processor can detect this and set it the maximum.

Input

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 \\ 20 & 20 & 20 & 20 & 20 \\ 50 & 50 & 50 & 50 & 50 \\ 90 & 90 & 90 & 90 & 90 \\ F0 & F0 & F0 & F0 & F0 \end{bmatrix}$$

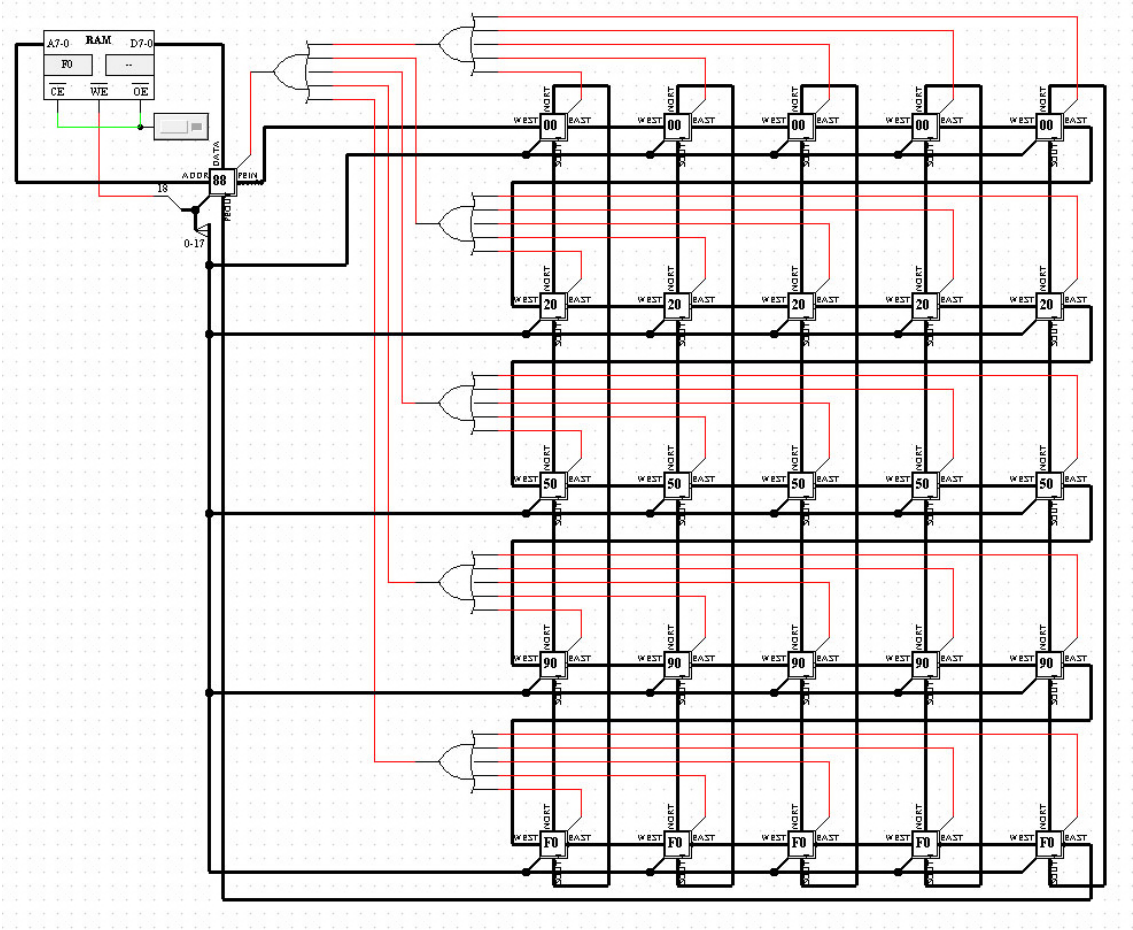


Figure 5.9 Brightness - Input

Assembly Code

```
STORE F0
ADD F0 (ADDR)
STATUS_CARRY
LOAD FF (CONST)
```

Expected Output

$$\begin{bmatrix} 00 & 00 & 00 & 00 & 00 \\ 40 & 40 & 40 & 40 & 40 \\ A0 & A0 & A0 & A0 & A0 \\ FF & FF & FF & FF & FF \\ FF & FF & FF & FF & FF \end{bmatrix}$$

Actual Output

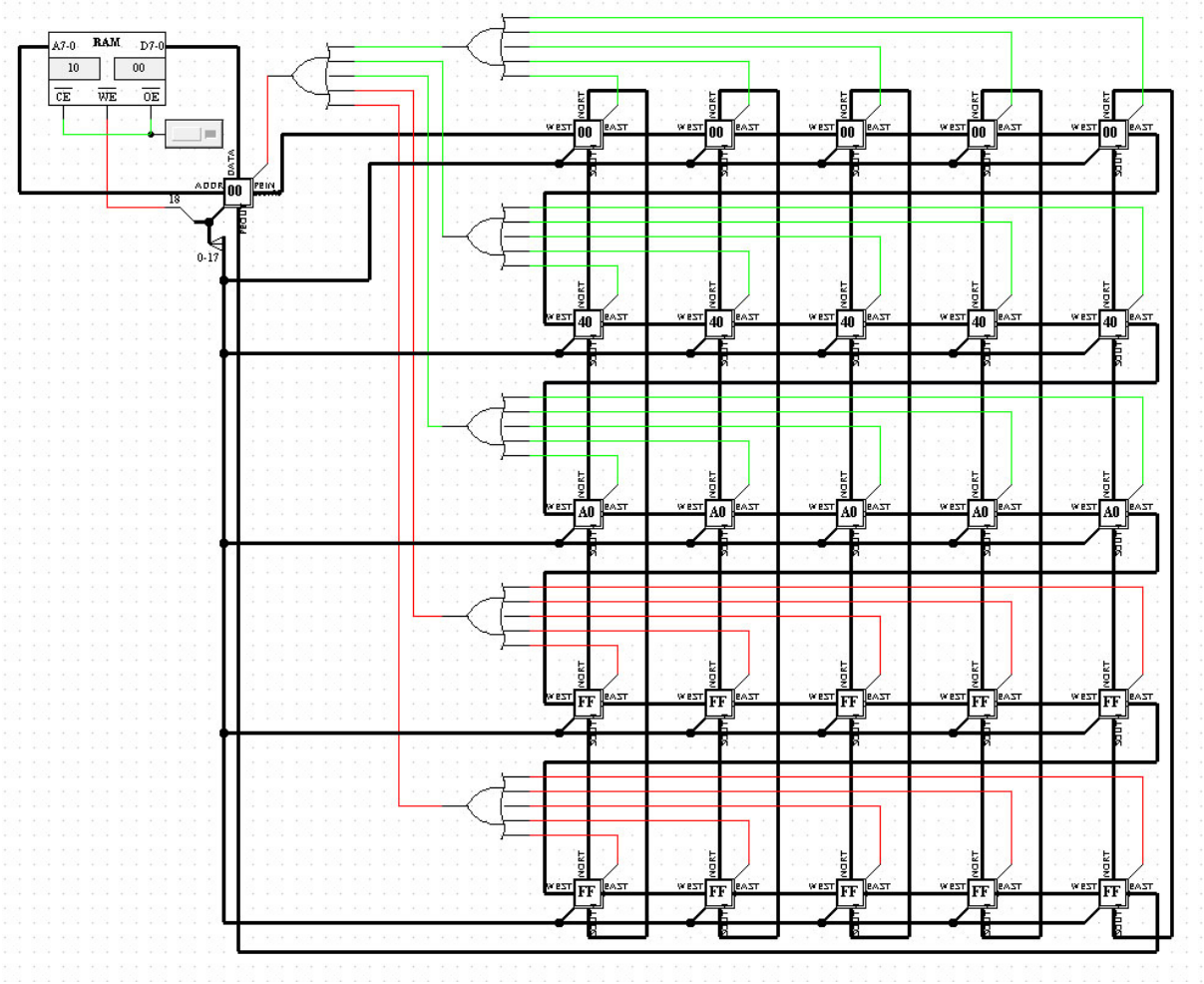
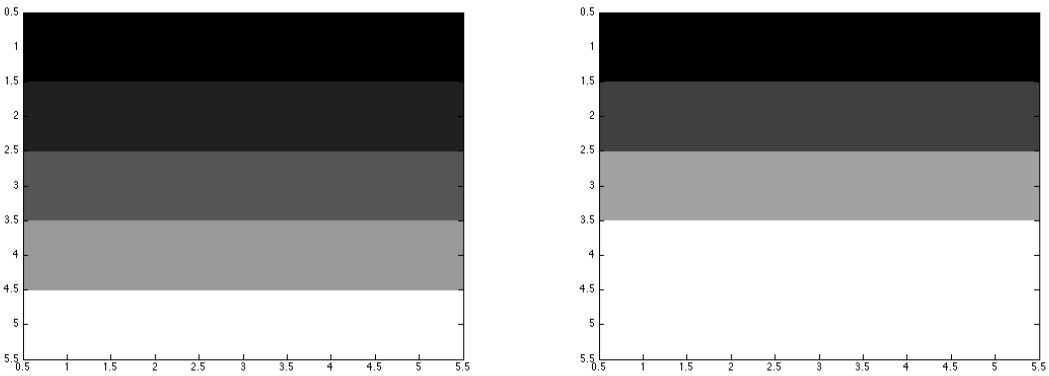


Figure 5.10 Brightness - Output

Visual Comparison



Input Output

5.6 Sobel Edge Detection

An edge in an image contains a lot of information. Finding an edge in an image is considered as one of the most basic image processing operations in computer vision. An edge can be found by passing an image through a filter, and the edges enhanced in the resultant image. Sobel operator is one of the most famous and commonly used filters for edge detection. A Sobel operator has the following form:

-1	0	1
-2	0	2
-1	0	1

For vertical edges

1	2	1
0	0	0
-1	-2	-1

For horizontal edges

Input

$$\begin{bmatrix} 00 & 00 & 10 & 00 & 00 \\ 00 & 00 & 10 & 00 & 00 \\ 10 & 10 & 10 & 10 & 10 \\ 00 & 00 & 10 & 00 & 00 \\ 00 & 00 & 10 & 00 & 00 \end{bmatrix}$$

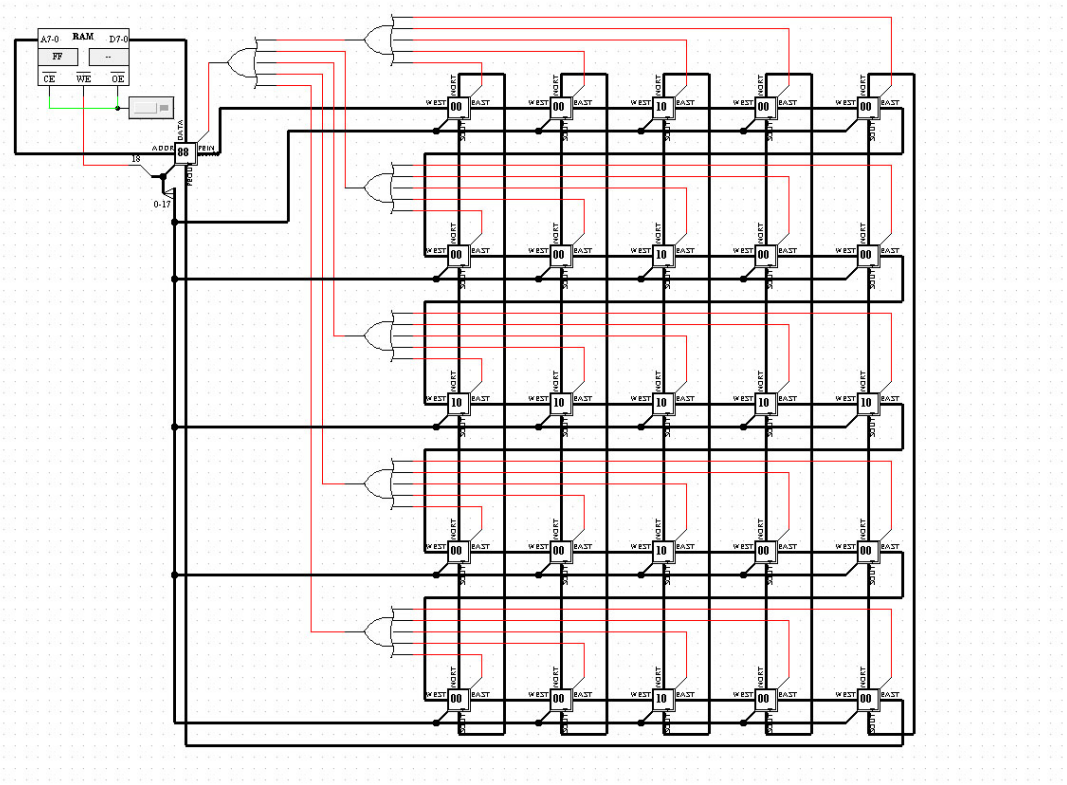


Figure 5.11 Sobel - Input

Assembly Code

```
STORE ff
SHIFT UP
STORE c1
SHIFT LEFT
STORE c2
LOAD ff
SHIFT UP
SHIFT RIGHT
STORE c0
LOAD ff
SHIFT DOWN
STORE a1
SHIFT LEFT
STORE a2
LOAD ff
SHIFT DOWN
SHIFT RIGHT
STORE a0
LOAD ff
SHIFT LEFT
STORE b2
LOAD ff
SHIFT RIGHT
STORE b0

LOAD 00 (CNST)
ADD c0
ADD c1
ADD c1
ADD c2
STORE 10
LOAD 00 (CNST)
ADD a0
ADD a1
ADD a1
ADD a2
INVERT
ADD_WITHCARRY ADDR 10
STATUS_NEGATIVE
INVERT
ADD 01 (CNST)
STATUS_SHIFTRIGHT
STORE F0

LOAD 00 (CNST)
ADD a0
ADD b0
ADD b0
ADD c0
STORE 20
LOAD 00 (CNST)
ADD a2
ADD b2
ADD b2
ADD c2
INVERT
ADD_WITHCARRY ADDR 10
STATUS_NEGATIVE
INVERT
ADD 01 (CNST)
STATUS_SHIFTRIGHT
STORE f1

ADD f0 (ADDR)
```

Expected Output

$$\begin{bmatrix} 00 & 40 & 00 & 40 & 20 \\ 40 & 60 & 20 & 60 & 40 \\ 20 & 20 & 00 & 20 & 20 \\ 40 & 60 & 20 & 60 & 40 \\ 20 & 40 & 00 & 40 & 00 \end{bmatrix}$$

Actual Output

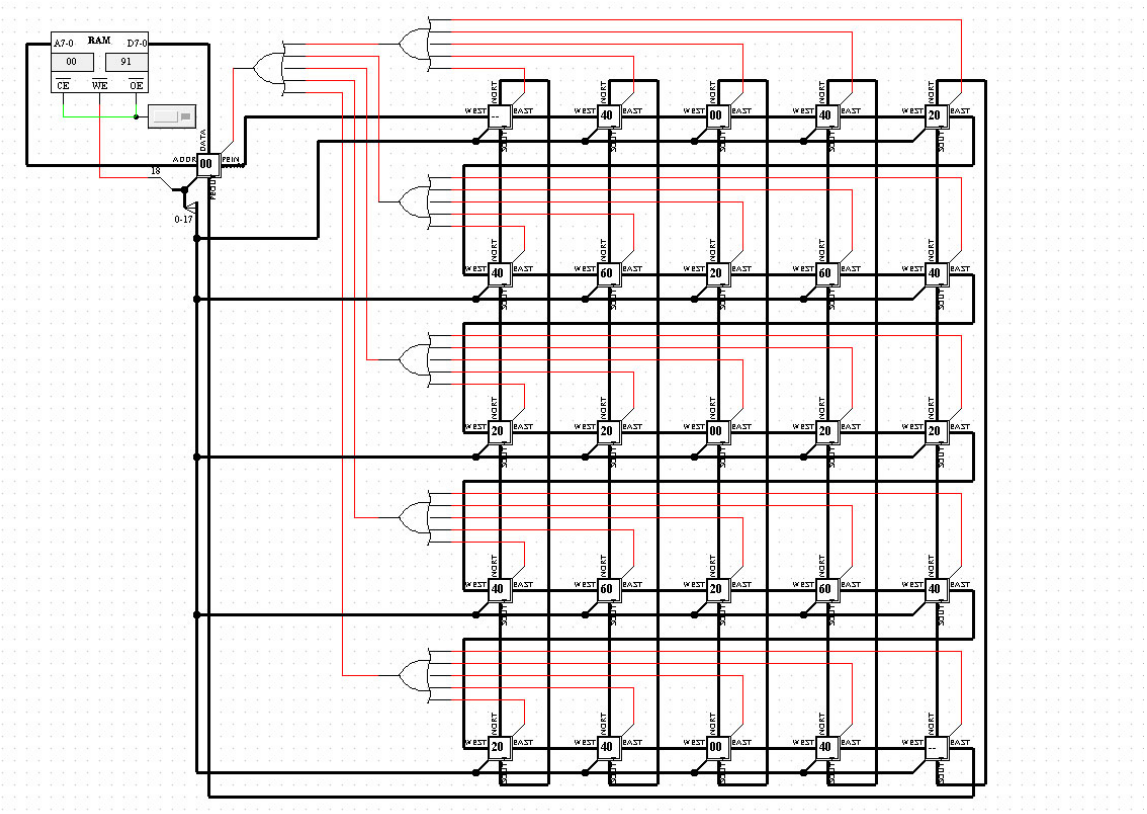
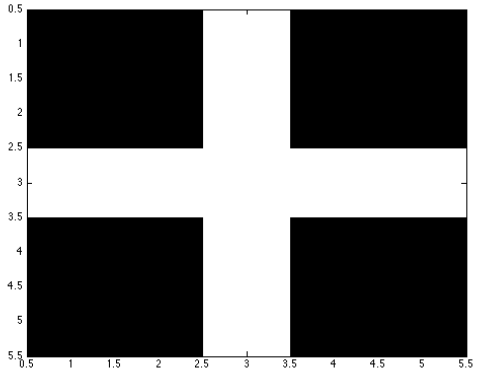
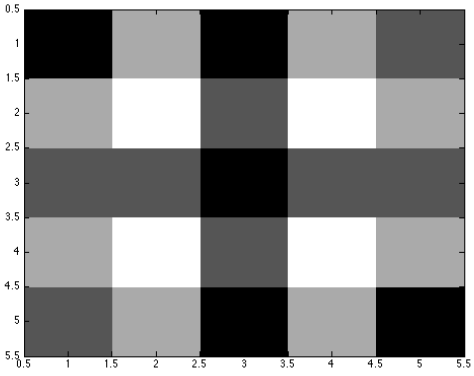


Figure 5.12 Sobel - Output

Visual Comparison



Input



Output

Chapter 6 Conclusion

The aim of this project was to design a SIMD microprocessor for image processing. Using Retro software, a complete circuit has been developed and simulated using a number of image processing applications.

In the first stage of the project, we have improved and added a number of features into the Retro software. The entire design process is closely matched to the project requirement. A number of issues were discovered during the design process and the solutions were developed. Being able to use it to design and simulate our SIMD circuit confirms the working state of Retro. The newly added features give Retro the ability to handle a much more complicated circuit. Retro software is more powerful than ever before, and it is now one step closer to become an industrial circuit design tool.

Using Retro software, we have successfully developed a SIMD microprocessor for image processing. This included a simple and easy-to-understand instruction set, a simple and highly efficient SIMD circuits. A list of criteria has been considered and decisions were made at the planning stage. During the design process, we have overcome a number of challenges, such as conditional statements and loops.

Finally, we simulated our circuit using various image processing applications in Retro software. The outputs from the simulation are consistent with our expected outcome. This verifies the correctness of our design concept. This also suggests our SIMD circuit can be turn into a real product in a foreseeable future.

Future Work

This project is part of an ongoing project. This project lays down a solid foundation and provides guidance to all future work. There are three tasks need to be done in the future.

First of all, more features need to be added into Retro. The most important feature is the ability to convert a circuit schematic into VHDL source code. This functionality makes Retro even one step closer to become an industrial design tool.

Secondly, implementing the circuit in VHDL is the next step. VHDL is still the on of the most popular hardware programming language. Having the circuit in VHDL form enable us to test the circuit in more platforms.

Finally, we also need a hardware implementation of the circuit design. This can be done using a FPGA development kit such as Xilinx Vertex 5. A number f image processing applications can be run on the actual hardware.

Reference

[1] (2013, 22/04/2013). *Grayscale*. Available: http://en.wikipedia.org/wiki/Grayscale

[2] B. Atkins. (2003, 25 Spetember 2013). *Digital Cameras - A beginner's guide*. Available: http://photo.net/equipment/digital/basics/

[3] R. Owens. (1997, 22/04/2013). *Binary Images*. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT2/lect2.html

[4] W. Fulton. (2010, 06/05/2013). *Color Bit-Depth, & Memory Cost of Images*. Available: http://www.scantips.com/basics1d.html

[5] T. B. W. R. S. F. M. Reinhardt, *Parallel Image Processing*. Germany: Springer, 2000.

[6] D. Huynh, "CITS4402 Computer Vision," The University of Western Australia, 2013.

[7] A. L. Shimpi. (2011, 21 Spetember 2013). *ARM's Cortex A7: Bringing Cheaper Dual-Core & More Power Efficient High-End Devices*. Available: http://www.anandtech.com/show/4991/arms-cortex-a7-bringing-cheaper-dualcore-more-power-efficient-highend-devices

[8] E. Harding. (16 Apr 2013). *The History of Microprocessors*. Available: http://jupiter.plymouth.edu/~harding/historymicro.html

[9] I. Corporation. (2013, 02 Apr 2013). *Moore's Law and Intel Innovation*. Available: http://www.intel.com/content/www/us/en/history/museum-gordon-moore-law.html

[10] R. Traylor, "A brief view of computer architecture," presented at the ECE 112 - Introduction to Electrical and Computer Engineering Oregon State University, 2002.

[11] A. S. O'Fallon, "Week 2 Unit 3: Computer Architecture Overview," presented at the Microprocessor Systems, Washington State University, 2013.

[12] V. Fay-Wolfe. (2005, 17 May 2013). *How Computers Work: The CPU and Memory*. Available: http://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading04.htm

[13] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proceedings of the IEEE,* vol. 60, pp. 369-388, 1972.

[14] G. Wilson. (1993, 12 March 2013). *History of Supercomputing (1993-08-20)*. Available: http://wotug.ukc.ac.uk/parallel/documents/misc/timeline/timeline.txt

[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*: Morgan Kaufmann Publishers Inc., 2011.

[16] R. M. Hord, *The Illiac IV, the First Supercomputer*: Computer Science Press, 1982.

[17] W. D. Hillis, *The Connection Machine*: Cambridge, 1989.

[18] S. Siewert. (14 Mar 2013). *Using Intel® Streaming SIMD Extensions and Intel® Integrated Performance Primitives to Accelerate Algorithms*. Available: http://software.intel.com/en-us/articles/using-intel-streaming-simd-extensions-and-intel-integrated-performance-primitives-to-accelerate-algorithms

[19] "PENTIUM® PROCESSOR WITH MMX™ TECHNOLOGY," I. Corporation, Ed., ed. Mt. Prospec: Intel Corporation, 1997.

[20] N. Corporation. (2013, 14 Mar 2013). *Parallel Programming and Computing Platform | CUDA | NVIDIA | NVIDIA*. Available: http://www.nvidia.com/object/cuda_home_new.html

[21] I. Advanced Micro Devices. (2013, 14 Mar 2013). *3DNow!™ Technology*. Available: http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/3DNOW/Pages/3dnow.aspx

[22] R. Duncan, "A survey of parallel computer architectures," *Computer,* vol. 23, pp. 5-16, 1990.

[23] M. Flynn, "Some Computer Organizations and Their Effectiveness," *Computers, IEEE Transactions on,* vol. C-21, pp. 948-960, 1972.

[24] C. McDonald, "CITS3230 Computer Network," presented at the Lecture 4, The University of Western Australia, 2013.

[25] T. Hey. (2010, 9/5/2013). *The Big Idea: The Next Scientific Revolution*. Available: http://hbr.org/2010/11/the-big-idea-the-next-scientific-revolution/ar/1

[26] I. Cadence Design Systems. (2013, 18 May 2013). *PSpice A/D and Advanced Analysis*. Available: http://www.cadence.com/products/orcad/pspice_simulation/pages/default.aspx

[27] I. CircuitLab. (2013, 18 May 2013). *CircuitLab - online schematic editor and simulator*. Available: https://http://www.circuitlab.com/

[28] O. Corporation. (2011, 19 Jul 2013). *Abstract Window Toolkit (AWT)*. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/awt/

[29] O. Corporation. (2013, 18 May 2013). *JDK 6 Swing (Java Foundation Classes (JFC))-related APIs and Developer Guides*. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/swing/index.html

[30] M. Bunge, "A General Black Box Theory," *Philosophy of Science,* vol. 30, pp. 346-358, 1963.

[31] M. Sung, "SIMD Parallel Processing," Massachusetts Institute of Technology, Cambridge2000.

[32] "8-bit AVR Instruction Set," ed. San Jose: Atmel Corporation, 2010, p. 160.

[33] "M68000 FAMILY PROGRAMMER'S REFERENCE MANUAL," ed. Denver: Motorola Inc, 1992, p. 646.

Appendix A Module

Source Code (Module.java)

```
package sim.lib.others;

import java.awt.*;
import java.awt.geom.AffineTransform;
import java.io.*;
//import java.nio.file.*;
import java.util.Arrays;
import javax.swing.*;

import sim.*;
import sim.lib.*;
import sim.engine.*;
import sim.lib.outputs.BusLed;
import sim.lib.wires.Junction;
import sim.lib.wires.SplitterModule;
import sim.lib.wires.Wire;

public class Module extends WrapperPainted implements EngineModule {
	/*
==========================================================================
	 Creation Part

========================================================================== */

	private static Image ICON =
GuiFileLink.getImage("sim/lib/others/Module.gif");

	public Image getIcon() {
		return Module.ICON;
	}

	public Wrapper createWrapper() {
		return this.getCopy();
	}

	public Wrapper createWrapper(Point gridPosition) {
		Module result = this.getCopy();
		result.setGridLocation(gridPosition);
		return result;
	}

	public String getBubbleHelp() {
		return "Module";
	}
```

```
	/*
==========================================================================
	 GUI part

========================================================================== */
	private String value = null;
	private int busSize;
	private int valueLenght;
	private String modpath = null;
	private int numPin;
	private boolean pin99 = false;
	private String[] labels = new String[100];
	private String modLabel = null;
	public static String lastPath;
	//private static String errorPath;
	private static boolean loadError;
	private int[] busSizes = new int[100];
	private static loadedModule[] loaded = new loadedModule[50];
	private String regName;
	private sim.lib.memory.Register regOut = null;
	private String[] regList = new String[50];

	public Module() {
		super();

		this.setBusSize(8);

		//this.setPath(lastPath);
	}

	public Module getCopy() {
		Module result = new Module();
		result.setPath(lastPath);
		result.setRName(this.regName);
		result.defaultLabel();

		//result.setBusSize(this.busSize);

		return result;
	}

	public void defaultLabel() {
		if (this.modpath != null) {
			String parts[] = this.modpath.split("\\\\");
			String name = parts[parts.length - 1];
			name = name.substring(0, name.length() - 4);
			this.modLabel = name;
		}
	}
```

```
	public String getLabel() {
		return this.modLabel;
	}

	public void setLabel(String label) {
		this.modLabel = label;
	}

	public void setRName(String reg) {
		this.regName = reg;
	}

	public void setRegOut(sim.lib.memory.Register r) {
		this.regOut = r;
	}

	public String getPath() {
		return this.modpath;
	}

	public void setPath(String file) {
		this.modpath = file;

		//LOAD numPin = number of pin components from file
		//get names for pins into string array
		this.numPin = 0;
		Grid g = new Grid();
		String[] basics, specifics;
		String className, componentName, readIn;
		BufferedReader inStream;
		Wrapper created;
		int z = 0;

		try {
			inStream = new BufferedReader(new FileReader(file));
			readIn =
SaveLoadShortcut.GUI_FILE_LINK.extractParameter(inStream);
			SaveLoadShortcut.GUI_FILE_LINK.readBlank(inStream);

	SaveLoadShortcut.GUI_FILE_LINK.extractParameters(g.getNumberOf
Parameters(), inStream);
			SaveLoadShortcut.GUI_FILE_LINK.readBlank(inStream);
			int size =
Integer.valueOf(SaveLoadShortcut.GUI_FILE_LINK.extractParameter(inSt
ream)).intValue();
			for (int index = 0; index < size; index++) {

	SaveLoadShortcut.GUI_FILE_LINK.readBlank(inStream);
```

```
                        className =
SaveLoadShortcut.GUI_FILE_LINK.extractParameter(inStream);
                        componentName =
SaveLoadShortcut.GUI_FILE_LINK.extractParameter(inStream);
                        created =
SaveLoadShortcut.GUI_FILE_LINK.getWrapper(className);

                        basics =
SaveLoadShortcut.GUI_FILE_LINK.extractParameters(created.getNumberOf
BasicParameters(), inStream);
                        specifics =
SaveLoadShortcut.GUI_FILE_LINK.extractParameters(created.getNumberOf
SpecificParameters(), inStream);
                        if (className.equals("sim.lib.outputs.Pin")) {
                              if(Integer.valueOf(specifics[1]) == 99)
                                    this.pin99 = true;
                              else
                                    this.numPin++;

      this.busSizes[Integer.valueOf(specifics[1]) - 1] =
Integer.valueOf(specifics[0]); //get bus size for each pin
                              this.labels[Integer.valueOf(specifics[1])
- 1] = specifics[2]; //get label string for each pin

                        } else if
(className.equals("sim.lib.memory.Register")) {
                              this.regList[z] = specifics[2];
                              z++;
                        }
                  }
                  lastPath = file;

            } catch (Exception e) {
                  if (!loadError) {
                        JOptionPane.showMessageDialog(null, "Error
loading module file, please correct file paths in Control Center",
"ERROR", JOptionPane.ERROR_MESSAGE);
                        loadError = true;
                  }
            }


      }

      public String[] getRegList() {
            return this.regList;
      }

      public void initializeGridSize() {
```

```
            this.setGridSize(6, 7);
      }

      public void setBusSize(int size) {
            this.busSize = size;

            String max = Integer.toHexString((int) Math.pow(2,
this.busSize) - 1).toUpperCase();
            FontMetrics fm = this.getFontMetrics(new
Font(Wrapper.FONT_NAME, Font.PLAIN, 3 * Grid.SIZE / 4));

            //this.setGridSize(fm.stringWidth(max) / Grid.SIZE + 4,
4);
            //this.setGridSize(6,6);
            this.valueLenght = max.length();
      }

      public int getBusSize() {
            return this.busSize;
      }

      public void paintValue(Graphics g) {
            if (this.isVisible()) {
                  int gridGap =
CentralPanel.ACTIVE_GRID.getCurrentGridGap();
                  int increment = gridGap / 4;
                  String val = "";
                  if (this.regOut != null) {
                        val = this.regOut.getValue();
                        g.setFont(new Font(Wrapper.FONT_NAME,
Font.BOLD, gridGap));
                  }
                  g.drawString(val, 2 * gridGap + increment + 1,
gridGap * 2 + 5 * increment);
            }
      }

      public void paint(Graphics g) {

            // draw if visible
            if (this.isVisible()) {
                  int gridGap =
CentralPanel.ACTIVE_GRID.getCurrentGridGap();
                  int increment = gridGap / 4;

                  //g.setColor(this.brush);

                  int offset = 2 * gridGap;
                  g.setColor(Color.WHITE);
```

```
                g.fillRect(2 * gridGap + increment / 2, 2 * gridGap
+ increment / 2, offset - increment, offset - increment);
                g.setColor(this.brush);

               g.drawRect(2 * gridGap, 2 * gridGap, offset, offset);
                g.drawRect(2 * gridGap + increment / 2, 2 * gridGap
+ increment / 2, offset - increment, offset - increment);

                //g.drawRect(gridGap + gridGap / 5, 2 * increment +
gridGap / 5, offset - 2*gridGap/5, offset -2*gridGap/5);

                int start = 3 * gridGap;
                int end = 5 * gridGap;
                offset = 2 * gridGap;
                g.setFont(new Font(Wrapper.FONT_NAME, Font.BOLD,
gridGap * 2 / 3));
                Font f = new Font(Wrapper.FONT_NAME, Font.PLAIN, 3 *
increment);
                String name;
                if (modLabel == null) {
                      String parts[] = this.modpath.split("\\\\");
                      name = parts[parts.length - 1];
                      name = name.substring(0, name.length() - 4);
                      this.modLabel = name;
                } else {
                      name = modLabel;
                }
                if (this.regOut != null) {
                      name = this.regOut.getValue();
                      g.setFont(new Font(Wrapper.FONT_NAME,
Font.BOLD, gridGap));
                }
                g.drawString(name, 2 * gridGap + increment + 1,
gridGap * 2 + 5 * increment);
                //g.drawString(this.modpath, 0,0);

                offset = offset + gridGap;

                //**********Draw ports based on number of pins in
input file**********
                g.setFont(new Font(Wrapper.FONT_NAME, Font.PLAIN,
gridGap * 2 / 3));
                if(this.pin99){
                      g.drawLine(4 *gridGap, 2 * gridGap, 5 *
gridGap, 1 * gridGap); //portF (DIAGONAL)
                }
                if (this.numPin > 0) {
                      g.drawString(this.labels[0].substring(0,
Math.min(5, this.labels[0].length())), gridGap / 8, gridGap * 5 / 2
```

```
+ increment); //GET PIN NAMES FROM INPUT FILE
                        g.fillRect(0, 3 * gridGap - 1, 2 * gridGap, 3);
//portA (LEFT)
                  }
                  if (this.numPin > 1) {
                        AffineTransform tmp = f.getTransform();
                        AffineTransform at = new AffineTransform();
                        at.rotate(-90 * java.lang.Math.PI / 180);
                        Font tf = f.deriveFont(at);
                        g.setFont(tf.deriveFont((float) (gridGap * 2 /
3)));
                        g.drawString(this.labels[1].substring(0,
Math.min(5, this.labels[1].length())), gridGap * 11 / 4, gridGap * 2
- gridGap / 8);

                        g.fillRect(3 * gridGap - 1, 0, 3, 2 * gridGap);
//portB (TOP)
                        /*for(int i = 0; i < this.labels[1].length();
i++){
                         if (i > 10){
                         break;
                         }
                         g.drawString(this.labels[1].charAt(i)+"",
gridGap*3/2, gridGap*2/3+i*gridGap/2);
                         }*/
                  }
                  if (this.numPin > 2) {
                        g.setFont(new Font(Wrapper.FONT_NAME,
Font.PLAIN, gridGap * 2 / 3));
                        g.drawString(this.labels[2].substring(0,
Math.min(5, this.labels[2].length())), gridGap * 4 + gridGap / 8,
gridGap * 5 / 2 + increment);
                        g.fillRect(4 * gridGap, 3 * gridGap - 1, 2 *
gridGap, 3); //portC (RIGHT)
                  }
                  if (this.numPin > 3) {
                        AffineTransform tmp = f.getTransform();
                        AffineTransform at = new AffineTransform();
                        at.rotate(-90 * java.lang.Math.PI / 180);
                        Font tf = f.deriveFont(at);
                        g.setFont(tf.deriveFont((float) (gridGap * 2 /
3)));
                        g.drawString(this.labels[3].substring(0,
Math.min(5, this.labels[3].length())), gridGap * 7 / 2 + gridGap / 8,
gridGap * 6 - gridGap / 8);

                        g.fillRect(3 * gridGap - 1, 4 * gridGap, 3, 2
* gridGap); //portD (BOTTOM)
                  }
```

```
                if (this.numPin > 4) {
                      g.drawLine(gridGap, 5 * gridGap, 2 * gridGap,
4 * gridGap); //portE (DIAGONAL)
                      g.drawLine(gridGap, 5 * gridGap - 1, 2 *
gridGap, 4 * gridGap - 1); //portE (DIAGONAL)
                      g.drawLine(gridGap, 5 * gridGap + 1, 2 *
gridGap, 4 * gridGap + 1); //portE (DIAGONAL)
                }

           }
      }
      /*
================================================================
       Maintanance Part

================================================================ */
      private Junction portA = null; //IN //LEFT
      private Junction portB = null; //TOP
      private Junction portC = null; //OUT //RIGHT
      private Junction portD = null; //BOTTOM
      private Junction portE = null; //DIAGONAL LEFTBOTTOM OUT (Any
other bits combined into single bus - except pin 99 (special))
      private Junction portF = null; //DIAGONAL TOPRIGHT OUT single
special bit
      //private Junction portO = null; //LCD OUT

      public boolean canDrop() {
           boolean result = false;
           if (this.numPin > 0) {
                result = Wrapper.canDropJuncion(this.gridLocation.x,
this.gridLocation.y + 3, this.busSizes[0]); //PORTA
           }
           if (this.numPin > 1) {
                result = result &&
Wrapper.canDropJuncion(this.gridLocation.x + 3, this.gridLocation.y,
this.busSizes[1]); //PORTB
           }
           if (this.numPin > 2) {
                result = result &&
Wrapper.canDropJuncion(this.gridLocation.x + 6, this.gridLocation.y
+ 3, this.busSizes[2]); //PORTC
           }
           if (this.numPin > 3) {
                result = result &&
Wrapper.canDropJuncion(this.gridLocation.x + 3, this.gridLocation.y
+ 6, this.busSizes[3]); //PORTD
           }
           if (this.numPin > 4) {
```

```
                int totalbus = 0;
                for (int i = 4; i < this.busSizes.length - 10; i++)
{
                     totalbus = totalbus + busSizes[i];
                }
                result = result &&
Wrapper.canDropJuncion(this.gridLocation.x + 1, this.gridLocation.y
+ 5, totalbus); //PORTE
           }
           if(this.pin99){
                result = result &&
Wrapper.canDropJuncion(this.gridLocation.x + 5, this.gridLocation.y
+ 1, 1); //PORTF
           }
           return result;
     }

     public void droped() {
           if (this.numPin > 0) {
                this.portA = Wrapper.setPinAt(this.gridLocation.x,
this.gridLocation.y + 3, this.busSizes[0]);
           }
           if (this.numPin > 1) {
                this.portB = Wrapper.setPinAt(this.gridLocation.x +
3, this.gridLocation.y, this.busSizes[1]);
           }
           if (this.numPin > 2) {
                this.portC = Wrapper.setPinAt(this.gridLocation.x +
6, this.gridLocation.y + 3, this.busSizes[2]);
           }
           if (this.numPin > 3) {
                this.portD = Wrapper.setPinAt(this.gridLocation.x +
3, this.gridLocation.y + 6, this.busSizes[3]);
           }
           if (this.numPin > 4) {
                int totalbus = 0;
                for (int i = 4; i < this.busSizes.length - 5; i++) {
                     totalbus = totalbus + busSizes[i];
                }
                this.portE = Wrapper.setPinAt(this.gridLocation.x +
1, this.gridLocation.y + 5, totalbus);
           }
           if(this.pin99){
                this.portF = Wrapper.setPinAt(this.gridLocation.x +
5, this.gridLocation.y + 1, 1); //PORTF
           }

           this.changeColor(Color.black);
     }
```

```
	public void selected() {
		if (this.numPin > 0) {
			this.portA.removePin();
		}
		if (this.numPin > 1) {
			this.portB.removePin();
		}
		if (this.numPin > 2) {
			this.portC.removePin();
		}
		if (this.numPin > 3) {
			this.portD.removePin();
		}
		if (this.numPin > 4) {
			this.portE.removePin();
		}
		if (this.pin99) {
			this.portF.removePin();
		}

		this.changeColor(Color.green);
	}

	public void checkAfterSelected() {
		if (this.numPin > 0) {
			Wrapper.checkPin(this.portA);
		}
		if (this.numPin > 1) {
			Wrapper.checkPin(this.portB);
		}
		if (this.numPin > 2) {
			Wrapper.checkPin(this.portD);
		}
		if (this.numPin > 3) {
			Wrapper.checkPin(this.portC);
		}
		if (this.numPin > 4) {
			Wrapper.checkPin(this.portE);
		}
		if (this.pin99) {
			Wrapper.checkPin(this.portF);
		}
	}
	/*
================================================================
	 Simulation part

================================================================ */
```

```
	public void evaluateOutput(double currentTime, Data[]
currentInputs, EnginePeer peer) {
		CentralPanel.ACTIVE_GRID.paintComponent(this);
	}

	public void createEnginePeer(EnginePeerList epl) {
	}

	private Object[] loadModule(String fname) {
		String inFile = fname;
		Reader inStream;

		MainWindow.CENTRAL_PANEL.createGrid("");
		CentralPanel.ACTIVE_GRID = new Grid();

		try {
			MainWindow.CENTRAL_PANEL.createGrid("MODULE");
			MainWindow.CENTRAL_PANEL.setVisible(false);
			inStream = new BufferedReader(new
FileReader(inFile));
			SaveLoadShortcut.GUI_FILE_LINK.loadMod(inStream,
CentralPanel.ACTIVE_GRID);
			//loaded[next] = new
loadedModule(fname,g,SaveLoadShortcut.GUI_FILE_LINK.getComp(inStream,
CentralPanel.ACTIVE_GRID));
			inStream.close();
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (SimException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		//Get EnginePeers from file
		//insert into input EnginePeerList
		//connect relevant nodes

		Grid g = CentralPanel.ACTIVE_GRID;

		//*******LOAD COMPONENTS FROM FILE******
		int loop;

		int wires = g.getNumberOfWires();
		int junctions = g.getNumberOfJunctions();
		int splitters = g.getNumberOfSplitters();
		int components = g.getNumberOfComponents();
```

```
            int total;

            NodeList tempnl = new NodeList();
            EnginePeerList tempepl = new EnginePeerList();

            for (loop = 0; loop < junctions; loop++) {
                  ((NodeModule)
g.getComponent(loop)).createNode(tempnl);
            }

            total = junctions + wires + splitters;
            for (loop = junctions + wires; loop < total; loop++) {
                  ((SplitterModule)
g.getComponent(loop)).mergeNodes(tempnl);
            }

            total = total + components;
            for (loop = junctions + wires + splitters; loop < total;
loop++) {
                  ((EngineModule)
g.getComponent(loop)).createEnginePeer(tempepl);
            }

            Object[] out = new Object[2];

            out[0] = tempepl;
            out[1] = tempnl;

            return out;
      }

      public void createEnginePeer(EnginePeerList epl, NodeList nl)
{
            //ENGINE PEER FOR MODULE
            this.busSize = 8;

            Object[] tmp = loadModule(this.modpath); //load EPL and
NL from file
            EnginePeerList tempepl = (EnginePeerList) tmp[0];
            NodeList tempnl = (NodeList) tmp[1];
            int[] pinpos = new int[this.busSizes.length]; //array to
hold current bit position of each pin

            for (Object o : tempepl) {
                  EnginePeer ept = (EnginePeer) o;
                  if
(!ept.getParent().getClass().toString().equals("class
sim.lib.outputs.Pin")) {
                        for (int k = 0; k <
```

```
ept.getInputPins().getSize(); k++) {
                              for (int i = 0; i <
ept.getInputPins().getItemAt(k).getConnection().getSize(); i++) {
                                    EnginePeer eptt =
ept.getInputPins().getItemAt(k).getConnection().getItemAt(i);
                                    if
(eptt.getParent().getClass().toString().equals("class
sim.lib.outputs.Pin")) { //component connected to a pin
                                          int n = ((sim.lib.outputs.Pin)
eptt.getParent()).getNumber();
                                          if(pinpos[n-1] >=
eptt.getInputPins().size()){
                                                pinpos[n-1] = 0;
                                          }
                                          if
(!ept.getInputPins().getItemAt(k).equals(eptt.getInputPins().getItem
At(pinpos[n - 1]))) { //find correct bit position of pin
                                                pinpos[n - 1] = 0;
                                                while
(!ept.getInputPins().getItemAt(k).equals(eptt.getInputPins().getItem
At(pinpos[n - 1]))) {
                                                      pinpos[n - 1]++;
                                                }
                                          }

                                          switch (n) //connect pins from
loaded file to ports on module component
                                          {
                                                case 1:
                                                      ept.setInputPin(k,
this.portA.getNodes().getItemAt(pinpos[n - 1]));
                                                      pinpos[n - 1]++;
                                                      break;
                                                case 2:
                                                      ept.setInputPin(k,
this.portB.getNodes().getItemAt(pinpos[n - 1]));
                                                      pinpos[n - 1]++;
                                                      break;
                                                case 3:
                                                      ept.setInputPin(k,
this.portC.getNodes().getItemAt(pinpos[n - 1]));
                                                      pinpos[n - 1]++;
                                                      break;
                                                case 4:
                                                      ept.setInputPin(k,
this.portD.getNodes().getItemAt(pinpos[n - 1]));
                                                      pinpos[n - 1]++;
                                                      break;
                                                case 99:
```

```
                                                ept.setInputPin(k,
this.portF.getNodes().getItemAt(pinpos[n - 1]));
                                                pinpos[n - 1]++;
                                                break;
                                          default:

                                                int base = 0;
                                                //n = n - 1;
                                                for (int j = 4; j
<= (n - 2); j++) { //get correct starting bit position for portE
                                                      base = base +
this.busSizes[j];
                                                }
                                                int a = base +
pinpos[n - 1];

      System.out.println(k + "," + a);
                                                ept.setInputPin(k,
this.portE.getNodes().getItemAt(base + pinpos[n - 1]));
                                                pinpos[n - 1]++;
                                          }

                                          break;
                                    }
                              }
                        }
                        Arrays.fill(pinpos, 0); //reset array to hold
current bit position of each pin
                        if (ept.getOutputPins() != null) {
                              for (int k = 0; k <
ept.getOutputPins().getSize(); k++) {
                                    for (int i = 0; i <
ept.getOutputPins().getItemAt(k).getConnection().getSize(); i++) {
                                          EnginePeer eptt =
ept.getOutputPins().getItemAt(k).getConnection().getItemAt(i);
                                          if
(eptt.getParent().getClass().toString().equals("class
sim.lib.outputs.Pin")) {
                                                int n =
((sim.lib.outputs.Pin) eptt.getParent()).getNumber();
                                                if(pinpos[n-1] >=
eptt.getInputPins().getSize()){
                                                      pinpos[n-1] = 0;
                                                }
                                                if
(!ept.getOutputPins().getItemAt(k).equals(eptt.getInputPins().getIte
mAt(pinpos[n - 1]))) { //find correct bit position of pin
                                                pinpos[n - 1] = 0;
                                                while
```

```
(!ept.getOutputPins().getItemAt(k).equals(eptt.getInputPins().getIte
mAt(pinpos[n - 1]))) {
                                                            pinpos[n -
1]++;
                                                      }
                                                }
                                                switch (n) {
                                                      case 1:

      ept.setOutputPin(k, this.portA.getNodes().getItemAt(pinpos[n -
1]));
                                                            pinpos[n -
1]++;
                                                            break;
                                                      case 2:

      ept.setOutputPin(k, this.portB.getNodes().getItemAt(pinpos[n -
1]));
                                                            pinpos[n -
1]++;
                                                            break;
                                                      case 3:

      ept.setOutputPin(k, this.portC.getNodes().getItemAt(pinpos[n -
1]));
                                                            pinpos[n -
1]++;
                                                            break;
                                                      case 4:

      ept.setOutputPin(k, this.portD.getNodes().getItemAt(pinpos[n -
1]));
                                                            pinpos[n -
1]++;
                                                            break;
                                                      case 99:

      ept.setOutputPin(k, this.portF.getNodes().getItemAt(pinpos[n -
1]));
                                                            pinpos[n -
1]++;
                                                            break;
                                                      default:
                                                            int base = 0;
                                                            for (int j =
4; j <= (n - 2); j++) { //get correct starting bit position for
portE
                                                                  base =
base + this.busSizes[j];
```

```
                                                                }

      ept.setOutputPin(k, this.portE.getNodes().getItemAt(base +
pinpos[n - 1]));
                                                                pinpos[n -
1]++;
                                                    }

                                                    break;
                                               }
                                          }
                                     }
                                }

                                epl.insertItem(ept);

                                if (ept.getParent() instanceof
sim.lib.memory.Register) {
                                     if (((sim.lib.memory.Register)
ept.getParent()).getRegName().equals(this.regName)) {
                                          //this.regout =
((sim.lib.memory.Register) ept.getParent());
                                          this.regOut =
((sim.lib.memory.Register) ept.getParent());
                                          //this.regList[z] =
((sim.lib.memory.Register) ept.getParent());
                                          //z++;
                                     }
                                }
                           }

                     }

                     EnginePeer ep = new EnginePeer(0, 0, this);
                     epl.insertItem(ep);

                     for (Object n : tempnl) {
                          for (Object j : ((Node) n).getJunctiions()) {
                               //((Junction)j).setLocation(-100,-100);
                               //((Junction)j).setEnabled(false);
                          }
                          for (Object w : ((Node) n).getWires()) {
                               ((Wire) w).setVisible(false);
                          }
                          EnginePeerList epltemp = ((Node) n).getConnection();
                          for (Object c : epltemp) {
                               ((EnginePeer)
c).getParent().getParentWrapper().setVisible(false);
                          }
```

```
                nl.insertItem((Node) n);
            }


            //this.regOut = this.regList[regIndex];

      }

      public void reset() {
      }

      public Wrapper getParentWrapper() {
            return this;
      }

      /*
=====================================================================
       Storage Part

===================================================================== */
      public String getSpecificParameters() {
            return (this.modpath + Wrapper.SEPARATOR + this.regName +
Wrapper.SEPARATOR + this.modLabel + Wrapper.SEPARATOR);
      }

      public void loadWrapper(String[] specificParameters) throws
SimException {
            if (specificParameters.length ==
this.getNumberOfSpecificParameters()) {
                  try {

      //this.setBusSize(Integer.valueOf(specificParameters[0]).intVa
lue());
                        this.setPath(specificParameters[0]);
                        this.regName = specificParameters[1];
                        this.modLabel = specificParameters[2];
                  } catch (NumberFormatException e) {
                        throw (new SimException("incorrect parameter
type"));
                  }
            } else {
                  throw (new SimException("incorrect number of
parameters"));
            }
      }

      public int getNumberOfSpecificParameters() {
            return 3;
      }
```

```
	/*
========================================================================
	 Rotation abd Flipping Part

======================================================================== */
	protected void adjustToChanges() {
	}
	/*
========================================================================
	 Popup Part

======================================================================== */
	private int oldBusSize = 0;

	public boolean hasProperties() {
		return true;
	}

	public Component getPropertyWindow() {
		return (new ModProperties(new String[5], this.regList,
this.regName, this.modLabel));
	}

	public void respondToChanges(Component property) {
		CentralPanel.ACTIVE_GRID.eraseComponent(this, false);
		ModProperties mp = (ModProperties) property;
		this.regName = mp.getRegName();
		this.modLabel = mp.getLabel();
		CentralPanel.ACTIVE_GRID.paintComponent(this);
	}

	public void restoreOriginalProperties() {
		if (this.oldBusSize != 0) {
			this.setBusSize(this.oldBusSize);
			this.oldBusSize = 0;
		}
	}
}
```

Source Code (ModProperties.java)

```
package sim.lib.others;

import java.awt.*;
import java.awt.event.*;

import sim.util.SimSeparator;

public class ModProperties extends Container implements ItemListener,
ActionListener, FocusListener {

	//private TextField editRDI = new TextField(5);
	//private TextField editSelectSize = new TextField(5);
	private TextField editLabel = new TextField(10);
	private Choice displayReg = new Choice();
	//private double oldDelay;
	//private int oldRDI;
	//private int oldSize;
	//private int oldRegIndex;
	private String oldRegName;
	private String oldLabel;
	private Label pins = new Label("Pins");
	//private Label simulation = new Label("Simulation");

	public ModProperties(String[] pins, String[] regList, String
regName, String modLabel) {
		super();
		this.setLayout(new BorderLayout(0, 15));
		int z = 0;
		while (regList[z] != null && z < 50) {
			this.displayReg.add(regList[z]);
			z++;
		}

		this.displayReg.select(regName);
		this.displayReg.addItemListener(this);

		this.oldLabel = modLabel;
		this.editLabel.addActionListener(this);
		this.editLabel.addFocusListener(this);
		this.editLabel.setText(modLabel);

		//this.oldRegName = regName;


		/*this.oldDelay = delay;
		 this.editDelay.addActionListener(this);
		 this.editDelay.addFocusListener(this);
```

```
 this.editDelay.setText(Double.toString(delay));

 this.oldSize = size;
 this.editSelectSize.addActionListener(this);
 this.editSelectSize.addFocusListener(this);
 this.editSelectSize.setText(Integer.toString(size));*/

// pins
Panel big = new Panel(new BorderLayout(0, 15));
Panel p = new Panel(new BorderLayout());
GridBagConstraints c = new GridBagConstraints();

p.add(this.pins, BorderLayout.WEST);
p.add(new SimSeparator(), BorderLayout.CENTER);
big.add(p, BorderLayout.NORTH);
p = new Panel(new GridBagLayout());

c.fill = GridBagConstraints.NONE;
c.insets = new Insets(0, 0, 0, 0);

c.gridy = 0;
c.gridx = 0;
p.add(new Label("Display Register"), c);
c.gridx = 1;
p.add(this.displayReg, c);

c.gridy = 1;
c.gridx = 0;
p.add(new Label("Module Label"), c);
c.gridx = 1;
p.add(this.editLabel, c);

//c.anchor = GridBagConstraints.EAST;
//c.gridx = 1;

big.add(p, BorderLayout.CENTER);

c.gridx = 0;
c.gridy = 0;
c.insets = new Insets(0, 0, 15, 0);
c.fill = GridBagConstraints.HORIZONTAL;
this.add(big, BorderLayout.CENTER);


// simulation
/*big = new Panel(new BorderLayout(0, 15));
 p = new Panel(new BorderLayout());
 p.add(this.simulation, BorderLayout.WEST);
 p.add(new SimSeparator(), BorderLayout.CENTER);
```

```
            big.add(p, BorderLayout.NORTH);

            p = new Panel(new FlowLayout(FlowLayout.LEFT, 0, 0));
            p.add(new Label("Propagation Delay"));
            p.add(this.editDelay);
            big.add(p, BorderLayout.CENTER);

            this.add(big, BorderLayout.CENTER);*/
      }

      public void addNotify() {
            super.addNotify();
            this.setSize(290, this.getPreferredSize().height * 2 +
this.pins.getPreferredSize().height * 2 + 45);
      }

      public void itemStateChanged(ItemEvent e) {
            //Choice source = (Choice) e.getSource();

            //if (source == this.displayReg) {
            this.oldRegName = this.displayReg.getSelectedItem();
            System.out.println(this.oldRegName);
            //}
      }

      public void actionPerformed(ActionEvent e) {

            if ((TextField) e.getSource() == this.editLabel) {
                  this.getLabel();
            }
            //if(source == this.editDelay)
            //    this.getDelay();
            //else if(source == this.editBus)
            //this.getBusSize();
      }

      public void focusGained(FocusEvent e) {
      }

      public void focusLost(FocusEvent e) {


            //if(source == this.editDelay)
            //    this.getDelay();
            //else if(source == this.editBus)
            //    this.getBusSize();
            if ((TextField) e.getSource() == this.editLabel) {
                  this.getLabel();
            }
```

```
	}

	/*public int getBusSize()
	 {
	 int newBus;

	 try
	 {
	 newBus = Integer.valueOf(this.editBus.getText()).intValue();

	 if(newBus > 0)
	 this.oldBus = newBus;
	 else
	 this.editBus.setText(Integer.toString(this.oldBus));
	 }
	 catch(NumberFormatException nfe)
	 {
	 this.editBus.setText(Integer.toString(this.oldBus));
	 }

	 return this.oldBus;
	 }*/
	public void setRegName() {
		String newReg = this.displayReg.getSelectedItem();
		this.oldRegName = newReg;
		this.displayReg.select(this.oldRegName);
	}

	public String getRegName() {
		//this.oldRegName = this.displayReg.getSelectedItem();
		/*String newReg = this.displayReg.getSelectedItem();
		 this.oldRegName = newReg;
		 this.displayReg.select(this.oldRegName);*/
		System.out.println(this.oldRegName);
		return this.oldRegName;
	}

	public String getLabel() {
		String newLabel = this.editLabel.getText();
		this.oldLabel = newLabel;
		return this.oldLabel;
	}

	/*public int getInputSize() {
	 int newSize;

	 try {
	 newSize =
Integer.valueOf(this.editSelectSize.getText()).intValue();
```

```
        if ((newSize > 0) && (newSize < 7)) {
        this.oldSize = newSize;
        } else {
        this.editSelectSize.setText(Integer.toString(this.oldSize));
        }
        } catch (NumberFormatException nfe) {
        this.editSelectSize.setText(Integer.toString(this.oldSize));
        }

        return this.oldSize;
        }*/

       /*public double getDelay()
        {
        double newDelay;

        try
        {
        newDelay =
Double.valueOf(this.editDelay.getText()).doubleValue();

        if(newDelay >= 0)
        this.oldDelay = newDelay;
        else
        this.editDelay.setText(Double.toString(this.oldBus));
        }
        catch(NumberFormatException nfe)
        {
        this.editDelay.setText(Double.toString(this.oldBus));
        }

        return this.oldDelay;
        }*/
       public Dimension getPreferredSize() {
            return this.getSize();
       }

       public Dimension getMinimumSize() {
            return this.getSize();
       }

       public Dimension getMaximumSize() {
            return this.getSize();
       }
}
```

Module Action

A number of action occurs in different phases of the lifetime of a module.

a) Phase 1 – loading
Phase 1 starts soon after the user click the module icon in the standard library. During this phase, module reads in a toy file provided by the user and initialises itself. It also generates a mouse tooltip.

Methods called:

```
public Module();
public void setPath(String File);
public void initializeGridSize();
public void setBusSize(int size);
public void paint(Grphaics g);
```

b) Phase 2 – drawing
Phase 2 starts when the user places down a module. During this phase, module checks to see if the given location is valid and the size of each bus is matched. If all requirements are valid, it will generate a module object and connect its buses.

Methods called:

```
public void paint(Grphaics g);
public boolean canDrop();
public void droped();
public void createEnginePeer(EnginePeerList epl, NodeList nl);
private Object[] loadModule(String fname);
```

c) Phase 3 – simulation
Phase 3 starts as soon as the user hit the simulation button. During this phase, module acts like a bridge, which transfers data between different circuits.

Methods called:

```
public void evaluateOutput(double currentTime, Data[] currentInputs,
    EnginePeer peer);
```

d) Phase 4 – clean up
Phase 4 starts when the user deletes a module. It clears itself from the canvas and deletes all related objects.

Methods called:

```
public void eraseComponent(Component comp, boolean update);
//in BufferedContainer.java
```

Appendix B Pin

Source Code (Pin.java)

```
package sim.lib.outputs;

import java.awt.*;
import java.io.*;

import sim.*;
import sim.engine.*;
import sim.lib.EditBusSize;
import sim.lib.wires.Junction;

public class Pin extends RotatableFlippableWrapperPainted implements
EngineModule
{
/*
=========================================================================
      Creation Part
      =========================================================================
=== */
      private static Image ICON =
GuiFileLink.getImage("sim/lib/outputs/pin.gif");

      public Image getIcon()
      {
            return Pin.ICON;
      }

      public Wrapper createWrapper()
      {
            return this.getCopy();
      }

      public Wrapper createWrapper(Point gridPosition)
      {
            Pin result = this.getCopy();
            result.setGridLocation(gridPosition);
            return result;
      }

      public String getBubbleHelp()
      {
            return "Pin";
      }

/*
```

```
=================================================================
      GUI part
      =================================================================
=== */
      private String value = null;
      private int busSize = 8;
      private int valueLenght;
      private int pinNumber;
      private String label;
      public static int numPins = 1;
      public static int nextNum = 1;
      public static boolean deleted = false;

      public Pin()
      {
            super();
            this.setBusSize(8);
            //this.setLabel("PIN"+nextNum);
            //this.setLabel("");
            this.pinNumber = nextNum;

      }

      public Pin getCopy()
      {
            Pin result = new Pin();
            result.setBusSize(this.busSize);
            result.setLabel("PIN"+nextNum);
            result.setNumber(nextNum);

            if (deleted){
                  deleted = false;
            }else{
                  nextNum = numPins;
            }

            return result;
      }

      public void setLabel(String lbl){
            this.label = lbl;
      }

      public void initializeGridSize()
      {
            this.setGridSize(4, 4);
      }

      public String getLabel(){
```

```
            return this.label;
      }

      public int getNumber(){
            return this.pinNumber;
      }

      public void setNumber(int n){
            this.pinNumber = n;
      }

      public void setBusSize(int size)
      {
            this.busSize = size;

            String max = Integer.toHexString((int)Math.pow(2,
this.busSize) - 1).toUpperCase();
            FontMetrics fm = this.getFontMetrics(new
Font(Wrapper.FONT_NAME, Font.PLAIN, 3 * Grid.SIZE / 4));

            //this.setGridSize(fm.stringWidth(max) / Grid.SIZE + 4,
4);
            this.valueLenght = max.length();
      }

      public int getBusSize()
      {
            return this.busSize;
      }

/*
=================================================================
      Maintanance Part
      =================================================================
=== */
      private Junction input = null;

      public void selected()
      {
            this.input.removePin();
            nextNum = this.pinNumber;
            deleted = true;
            numPins--;
            this.changeColor(Color.green);
      }

      public void checkAfterSelected()
      {
            Wrapper.checkPin(this.input);
```

```
	}

/*
=================================================================
	Simulation part
	=================================================================
=== */
	public void evaluateOutput(double currentTime, Data[]
currentInputs, EnginePeer peer)
	{
		boolean foundUndefined = false;
		int hex = 0;
		int base = 1;
		int loop, index;

		for(loop = 0; loop < (this.busSize / 4 + 1); loop++)
		{
			for(index = loop * 4; (index < 4 * (loop + 1)) &&
(index < this.busSize) &&(!foundUndefined); index++)
			{
				if(currentInputs[index].isUndefined())
					foundUndefined = true;
				else if(currentInputs[index].getValue())
					hex = hex + base;

				base = 2 * base;
			}

			if(loop == 0)
			{
				if(foundUndefined)
					this.value = "-";
				else
					this.value = Integer.toHexString(hex);
			}
			else if(foundUndefined)
				this.value = "-" + this.value;
			else if(this.value.length() < this.valueLenght)
				this.value = Integer.toHexString(hex) +
this.value;

			hex = 0;
			base = 1;
			foundUndefined = false;
		}

		this.value = this.value.toUpperCase();
		CentralPanel.ACTIVE_GRID.paintComponent(this);
	}
```

```
	public void createEnginePeer(EnginePeerList epl)
	{
		EnginePeer ep = new EnginePeer(this.busSize, this);

		for(int index = 0; index < this.busSize; index++)
			ep.setInputPin(index,
this.input.getNodes().getItemAt(index));

		epl.insertItem(ep);
	}

	public NodeList getInputPins(){
		return this.input.getNodes();
	}

	public void reset()
	{
		this.value = null;
		CentralPanel.ACTIVE_GRID.paintComponent(this);
	}

	public Wrapper getParentWrapper()
	{
		return this;
	}

/*
=================================================================
	Storage Part
	=================================================================
=== */
	public String getSpecificParameters()
	{
		return (Integer.toString(this.busSize) +
Wrapper.SEPARATOR + this.getNumber() + Wrapper.SEPARATOR +
this.getLabel() + Wrapper.SEPARATOR);
	}

	public void loadWrapper(String[] specificParameters) throws
SimException
	{
		if(specificParameters.length ==
this.getNumberOfSpecificParameters())
		{
			try
			{

	this.setBusSize(Integer.valueOf(specificParameters[0]).intValu
```

```
e());

      this.setNumber(Integer.valueOf(specificParameters[1]).intValue
());
                        this.setLabel(specificParameters[2]);
                        //this.flow =
Boolean.valueOf(specificParameters[0]).booleanValue();
                  }
                  catch(NumberFormatException e)
                  {
                        throw (new SimException("incorrect parameter
type"));
                  }
            }
            else
                  throw (new SimException("incorrect number of
parameters"));
      }

      public int getNumberOfSpecificParameters()
      {
            return 3;
      }

/*
=================================================================
      Rotation abd Flipping Part
      =================================================================
=== */

      protected void paintNormal_0(Graphics g)
      {
            int gridGap =
CentralPanel.ACTIVE_GRID.getCurrentGridGap();
            int width = this.gridSize.width * gridGap;
            int offsetX, offsetY;

            int buswidth = 3;
            if (this.busSize < 2){
                  buswidth = 1;
            }

            g.setColor(WrapperPainted.BACKGROUND);

            offsetY = 3 * gridGap / 2;
            offsetX = 2 * gridGap;

            //g.setColor(Color.white);
           //g.fillArc(gridGap, offsetY, 5*gridGap/4, 5*gridGap/4, 0,
```

```
360);
            g.setColor(this.brush);
            g.drawArc(gridGap, offsetY, gridGap, gridGap, 0, 360);
            g.fillRect(gridGap*2, offsetX - 1, gridGap, buswidth);

            g.setFont(new Font(Wrapper.FONT_NAME,Font.PLAIN,3 *
gridGap / 5));
            g.drawString(this.label,gridGap,gridGap);

      }

      protected void paintNormal_90(Graphics g)
      {
            int gridGap =
CentralPanel.ACTIVE_GRID.getCurrentGridGap();
            //this.gridSize.width = this.gridSize.width - gridGap;
            int width = this.gridSize.width * gridGap;
            int offsetX, offsetY;

            int buswidth = 3;
            if (this.busSize < 2){
                  buswidth = 1;
            }

            g.setColor(WrapperPainted.BACKGROUND);

            offsetY = 3 * gridGap / 2;
            offsetX = 2 * gridGap;

            g.setColor(this.brush);
            g.drawArc(3*gridGap/2, gridGap*2, gridGap, gridGap, 0,
360);
            g.fillRect(gridGap*2-1, gridGap, buswidth, gridGap);

            g.setFont(new Font(Wrapper.FONT_NAME,Font.PLAIN,3 *
gridGap / 5));
            g.drawString(this.label,6*gridGap/4,15*gridGap/4);


      }

      protected void paintNormal_180(Graphics g)
      {
            int gridGap =
CentralPanel.ACTIVE_GRID.getCurrentGridGap();
            int width = this.gridSize.width * gridGap;
            int offsetX, offsetY;

            int buswidth = 3;
```

```
            if (this.busSize < 2){
                  buswidth = 1;
            }

            g.setColor(WrapperPainted.BACKGROUND);

            offsetY = 3 * gridGap / 2;
            offsetX = 2 * gridGap;

            g.setColor(this.brush);
            g.drawArc(gridGap*2, offsetY, gridGap, gridGap, 0, 360);
            g.fillRect(gridGap, offsetX - 1, gridGap, buswidth);

            g.setFont(new Font(Wrapper.FONT_NAME,Font.PLAIN,3 *
gridGap / 5));
            g.drawString(this.label,2*gridGap,gridGap);
      }

      protected void paintNormal_270(Graphics g)
      {
            int gridGap =
CentralPanel.ACTIVE_GRID.getCurrentGridGap();
            int width = this.gridSize.width * gridGap;
            int offsetX, offsetY;

            int buswidth = 3;
            if (this.busSize < 2){
                  buswidth = 1;
            }

            g.setColor(WrapperPainted.BACKGROUND);

            offsetY = 3 * gridGap / 2;
            offsetX = 2 * gridGap;

            g.setColor(this.brush);
           g.drawArc(3*gridGap/2, gridGap, gridGap, gridGap, 0, 360);
            g.fillRect(gridGap*2-1, gridGap*2, buswidth, gridGap);

            g.setFont(new Font(Wrapper.FONT_NAME,Font.PLAIN,3 *
gridGap / 5));
            /*String tmp;
            if (this.pinNumber < 10){
                  tmp = "0" + this.pinNumber;
            }else{
                  tmp = Integer.toString(this.pinNumber);
            }*/
            g.drawString(this.label,6*gridGap/4,3*gridGap/4);
      }
```

```
	protected void paintFlipped_0(Graphics g)
	{
		this.paintNormal_0(g);
	}

	protected void paintFlipped_90(Graphics g)
	{
		this.paintNormal_270(g);
	}

	protected void paintFlipped_180(Graphics g)
	{
		this.paintNormal_180(g);
	}

	protected void paintFlipped_270(Graphics g)
	{
		this.paintNormal_90(g);
	}



	protected boolean canDropNormal_0()
	{
		return Wrapper.canDropJuncion(this.gridLocation.x + 3,
this.gridLocation.y + 2, this.busSize);
	}

	protected boolean canDropNormal_90()
	{
		return Wrapper.canDropJuncion(this.gridLocation.x + 2,
this.gridLocation.y + 1, this.busSize);
	}

	protected boolean canDropNormal_180()
	{
		return Wrapper.canDropJuncion(this.gridLocation.x + 1,
this.gridLocation.y + 2, this.busSize);
	}

	protected boolean canDropNormal_270()
	{
		return Wrapper.canDropJuncion(this.gridLocation.x + 2,
this.gridLocation.y + 3, this.busSize);
	}

	protected boolean canDropFlipped_0()
```

```
	{
		return this.canDropNormal_0();
	}

	protected boolean canDropFlipped_90()
	{
		return this.canDropNormal_270();
	}

	protected boolean canDropFlipped_180()
	{
		return this.canDropNormal_180();
	}

	protected boolean canDropFlipped_270()
	{
		return this.canDropNormal_90();
	}




	protected void dropedNormal_0()
	{
		this.input = Wrapper.setPinAt(this.gridLocation.x + 3,
this.gridLocation.y + 2, this.busSize);
		this.changeColor(Color.black);
		this.oldBusSize = 0;
		numPins++;
		nextNum = numPins;
	}

	protected void dropedNormal_90()
	{
		this.input = Wrapper.setPinAt(this.gridLocation.x + 2,
this.gridLocation.y + 1, this.busSize);
		this.changeColor(Color.black);
		this.oldBusSize = 0;
		numPins++;
		nextNum = numPins;
	}

	protected void dropedNormal_180()
	{
		this.input = Wrapper.setPinAt(this.gridLocation.x+1,
this.gridLocation.y + 2, this.busSize);
		this.changeColor(Color.black);
		this.oldBusSize = 0;
		numPins++;
```

```
            nextNum = numPins;
      }

      protected void dropedNormal_270()
      {
            this.input = Wrapper.setPinAt(this.gridLocation.x + 2,
this.gridLocation.y + 3, this.busSize);
            this.changeColor(Color.black);
            this.oldBusSize = 0;
            numPins++;
            nextNum = numPins;
      }

      protected void dropedFlipped_0()
      {
            this.dropedNormal_0();
            numPins++;
            nextNum = numPins;
      }

      protected void dropedFlipped_90()
      {
            this.dropedNormal_270();
            numPins++;
            nextNum = numPins;
      }

      protected void dropedFlipped_180()
      {
            this.dropedNormal_180();
            numPins++;
            nextNum = numPins;
      }

      protected void dropedFlipped_270()
      {
            this.dropedNormal_90();
            numPins++;
            nextNum = numPins;
      }



      protected void adjustToChanges()
      {
      }

/*
```

```
=================================================================
      Popup Part
      =================================================================
=== */
      private int oldBusSize = 0;

      public boolean hasProperties()
      {
            return true;
      }

      public Component getPropertyWindow()
      {
            return (new PinProperties(this.busSize,this.label));
      }

      public void respondToChanges(Component property)
      {
            this.brush = Color.white;
            CentralPanel.ACTIVE_GRID.paintComponent(this); //ERASE
OLD DRAWING
            this.brush = Color.black;

      this.setBusSize(((PinProperties)property).getPinBusSize());
            this.setLabel(((PinProperties)property).getLabel());
            CentralPanel.ACTIVE_GRID.paintComponent(this);
      }



      public void restoreOriginalProperties()
      {
            if(this.oldBusSize != 0)
            {
                  this.setBusSize(this.oldBusSize);
                  this.oldBusSize = 0;
            }
      }
}
```

Source Code (PinProperties.java)

```
package sim.lib.outputs;

import java.awt.*;
import java.awt.event.*;

import sim.util.SimSeparator;

public class PinProperties extends Container implements
ActionListener {

	private TextField editBus = new TextField(10);
	private TextField editName = new TextField(10);
	private Label name = new Label("Name");
	private Label bus = new Label("Bus Size");
	private int old;

	public PinProperties(int initbus, String initlabel) {
		super();
		this.setLayout(new GridBagLayout());


		this.editName.setText(initlabel);
		this.editBus.setText(Integer.toString(initbus));

		this.editName.addActionListener(this);
		this.editBus.addActionListener(this);

		//pin name
		Panel big = new Panel(new BorderLayout(0, 15));

		Panel p = new Panel(new BorderLayout());
		p.add(this.name, BorderLayout.WEST);
		p.add(new SimSeparator(), BorderLayout.CENTER);
		big.add(p, BorderLayout.NORTH);

		p = new Panel(new GridBagLayout());

		GridBagConstraints c = new GridBagConstraints();
		c.gridy = 0;
		c.gridwidth = 1;
		c.gridheight = 1;
		c.anchor = GridBagConstraints.WEST;
		c.weighty = 0;

		c.fill = GridBagConstraints.HORIZONTAL;
		c.weightx = 1;
		c.gridx = 0;
```

```
            p.add(this.editName, c);
            big.add(p, BorderLayout.CENTER);

            c.gridx = 0;
            c.insets = new Insets(0, 0, 0, 0);
            c.weightx = 1;
            c.fill = GridBagConstraints.HORIZONTAL;

            this.add(big, c);

            // bus
            big = new Panel(new BorderLayout(0, 15));

            p = new Panel(new BorderLayout());
            p.add(this.bus, BorderLayout.WEST);
            p.add(new SimSeparator(), BorderLayout.CENTER);
            big.add(p, BorderLayout.NORTH);

            p = new Panel(new GridBagLayout());

            c = new GridBagConstraints();
            c.gridy = 1;
            c.gridwidth = 2;
            c.gridheight = 1;
            c.anchor = GridBagConstraints.WEST;
            c.weighty = 0;

            c.fill = GridBagConstraints.HORIZONTAL;
            c.weightx = 1;
            c.gridx = 0;

            p.add(this.editBus, c);
            big.add(p, BorderLayout.CENTER);

            c.gridx = 0;
            c.insets = new Insets(0, 0, 0, 0);
            c.weightx = 1;
            c.fill = GridBagConstraints.HORIZONTAL;

            this.add(big, c);

      }

      public void addNotify() {
            super.addNotify();
            this.setSize(290, 7 *
this.name.getPreferredSize().height);
      }
```

```
	public void actionPerformed(ActionEvent e) {
		int newBus;

		try {
			newBus =
Integer.valueOf(this.editBus.getText()).intValue();

			if (newBus < 2) {

	this.editBus.setText(Integer.toString(this.old));
			} else {
				this.old = newBus;
			}
		} catch (NumberFormatException nfe) {
			this.editBus.setText(Integer.toString(this.old));
		}

	}

	public int getBusSize() {
		int newBus;

		try {
			newBus =
Integer.valueOf(this.editBus.getText()).intValue();

			if (newBus >= 2) {
				this.old = newBus;
			}
		} catch (NumberFormatException nfe) {
		}

		return this.old;
	}

	public int getPinBusSize() {
		int newBus;

		try {
			newBus =
Integer.valueOf(this.editBus.getText()).intValue();

			if (newBus >= 1) {
				this.old = newBus;
			}
		} catch (NumberFormatException nfe) {
		}
```

```
            return this.old;
      }

      public String getLabel() {

            return this.editName.getText();
      }

      public Dimension getPreferredSize() {
            return this.getSize();
      }

      public Dimension getMinimumSize() {
            return this.getSize();
      }

      public Dimension getMaximumSize() {
            return this.getSize();
      }
}
```

Pin Action

A number of action occurs in different phases of the lifetime of a pin.

a) Phase 1 – loading
Phase 1 starts soon after the user click the pin icon in the standard library. During this phase, it initialises itself and generates a mouse tooltip.

Methods called:

```
public Pin();
public void initializeGridSize();
public void setBusSize(int size);
public void paint(Grphaics g);
```

b) Phase 2 – drawing
Phase 2 starts when the user places down a pin. During this phase, pin checks to see if the given location is valid and the size of each bus is matched. If all requirements are valid, it will generate a pin object and connect itself to the bus.

Methods called:

```
public void paint(Grphaics g);
public boolean canDrop();
public void droped();
public void createEnginePeer(EnginePeerList epl);
```

c) Phase 3 – simulation
Phase 3 starts as soon as the user hit the simulation button. During this phase, pin receives data from input port of a module and sends data to the output port of a module.

Methods called:

```
public void evaluateOutput(double currentTime, Data[] currentInputs,
    EnginePeer peer);
```

d) Phase 4 – clean up
Phase 4 starts when the user deletes a pin. It clears itself from the canvas and deletes all related objects.

Methods called:

```
public void eraseComponent(Component comp, boolean update);
//in BufferedContainer.java
```

Appendix C Workspace

Source Code

```
//using Java Swing
package sim;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.*;
import javax.swing.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Workspace extends JPanel implements ActionListener,
ItemListener {

	private JFrame frame;
	private JComboBox dropList;
	private JFileChooser chooser;
	private Vector<String> recent;
	JCheckBox defaultButton;
	private int defaultIndex = -1;
	private String newPath = null;
	private final String DEFAULT_PATH_HEADER = "[Default
Workspace]";
	private final String RECENT_PATH_HEADER = "[Recent
Workspaces]";

	public Workspace() {
		//GUI
		readConfig();
		//normal case
		if (defaultIndex == -1) {
			initialiseGui();
		} //default check box is set
		else if (defaultIndex == 0) {
			MainWindow.WAIT_BLOCK.countDown();
		} //invalid
		else {
			MainWindow.setWorkspace(null, null);
			MainWindow.WAIT_BLOCK.countDown();
		}
	}
```

```
	private void initialiseGui() {
		frame = new JFrame(MainWindow.PROGRAM_NAME + " " +
MainWindow.VERSION + " - Please choose a workspace");

	frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
		frame.setResizable(false);
		frame.pack();
		frame.setSize(600, 300);
		frame.setLocation(MainWindow.getMiddleOfScreen(frame));

		JPanel panel = (JPanel) frame.getContentPane();
		panel.setLayout(new BorderLayout());

		//description label
		JTextArea description = new JTextArea();
		description.setEditable(false);

		InputStream input = null;
		String str = "";
		try {
			input =
Workspace.class.getResourceAsStream("workspace.txt");
			Scanner sc = new Scanner(input);
			while (sc.hasNextLine()) {
				str += sc.nextLine() + "\n";
			}
		} catch (NullPointerException e) {
			System.out.println("Can't find 'workspace.txt'");
		}

		description.setFont(new Font("Arial", Font.PLAIN, 13));
		description.setText(str);

		panel.add(description, BorderLayout.NORTH);

		//textfield label
		JPanel p = new JPanel(new GridBagLayout());
		GridBagConstraints c = new GridBagConstraints();
		//p.setSize(600, 50);
		c.gridx = 0;
		c.gridy = 0;
		JLabel text = new JLabel("Workspace:  ");
		p.add(text, c);

		//dropList
		c.gridx = 1;
		dropList = new JComboBox(recent);
		dropList.setEditable(true);
```

```
            dropList.addActionListener(this);
            dropList.setPreferredSize(new Dimension(400, 30));
            p.add(dropList, c);

            //Browse
            c.gridx = 2;
            JButton browse = new JButton("Browse");
            browse.setActionCommand("browse");
            browse.addActionListener(this);
            p.add(browse, c);

            panel.add(p);

            //set as default checkbox
            c.gridy = 1;
            c.gridx = 1;
            defaultButton = new JCheckBox("Set as default and do not
ask again");
            defaultButton.setSelected(false);
            defaultButton.addItemListener(this);
            //p.add(defaultButton, c);

            //Clear
            c.gridy = 2;
            c.gridx = 1;
            JButton clear = new JButton("Clear all recent history");
            clear.setActionCommand("clear");
            clear.addActionListener(this);
            p.add(clear, c);

            panel.add(p);

            //OK and Cencel
            p = new JPanel(new FlowLayout());

            JButton ok = new JButton("OK");
            ok.setActionCommand("ok");
            ok.addActionListener(this);
            p.add(ok);

            JButton cancel = new JButton("Cancel");
            cancel.setActionCommand("cancel");
            cancel.addActionListener(this);
            p.add(cancel);

            panel.add(p, BorderLayout.SOUTH);

            //file chooser
            chooser = new JFileChooser();
```

```
            chooser.setDialogTitle("Please choose a workspace");

      chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
//          try {
//
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassNa
me());
//
//          } catch (Exception e) {
//                System.err.println("Error: " + e.getMessage());
//          }
            frame.setVisible(true);
      }

      private void readConfig() {
            recent = new Vector<String>();
            try {
                  File file = new File("config.ini");
                  // config file doesn't exist, return
                  if (!file.exists()) {
                        return;
                  }
                  FileReader fr = new
FileReader(file.getAbsoluteFile());
                  BufferedReader br = new BufferedReader(fr);
                  //default path
                  String str = br.readLine();
                  //default path - incorrect format
                  if (!DEFAULT_PATH_HEADER.equals(str)) {
                        writeConfig();
                        br.close();
                        return;
                  }
                  str = br.readLine();
                  if (str == null) {
                        writeConfig();
                        br.close();
                        return;
                  }
                  defaultIndex = Integer.parseInt(str);
                  if (defaultIndex != 0 && defaultIndex != -1) {
                        defaultIndex = -1;
                        writeConfig();
                        br.close();
                        return;
                  }

                  //recent path
                  str = br.readLine();
```

```
                    //recent path - incorrect format
                    if (!RECENT_PATH_HEADER.equals(str)) {
                          writeConfig();
                          br.close();
                          return;
                    }
                    //read first line
                    str = br.readLine();
                    if (str == null) {
                          writeConfig();
                          br.close();
                          return;
                    }
                    //set as default - WORKSPACE PATH is found
                    if (defaultIndex == 0) {
                          File dir = new File(str);
                          str = trim(str);
                          if
(!(str.equals(dir.getAbsoluteFile().toString()))) {
                                JOptionPane.showMessageDialog(frame,
                                            "Please enter a valid path",
                                            "",
JOptionPane.WARNING_MESSAGE);
                                defaultIndex = -1;
                          } else {
                                //exist given folder
                                if (dir.exists()) {
                                      MainWindow.setWorkspace(str, null);
                                      br.close();
                                      return;
                                }
                                //doesn't exist
                                //can create
                                if (dir.mkdir()) {
                                      MainWindow.setWorkspace(str, null);
                                      br.close();
                                      return;
                                } //can't create
                                else {
                                      JOptionPane.showMessageDialog(frame,
                                                  "Fail to open workspace",
                                                  "",
JOptionPane.WARNING_MESSAGE);
                                      defaultIndex = -1;
                                }
                          }
                    }
                    while (str != null) {
                          recent.add(str);
```

```
                    str = br.readLine();
                }
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        private void writeConfig() {
            try {
                File file = new File("config.ini");
                if (file.exists()) {
                    file.delete();
                }
                file.createNewFile();

                FileWriter fw = new
FileWriter(file.getAbsoluteFile());
                BufferedWriter bw = new BufferedWriter(fw);
                bw.write(DEFAULT_PATH_HEADER + "\n");
                bw.write(defaultIndex + "\n");
                bw.write(RECENT_PATH_HEADER + "\n");
                if (newPath != null) {
                    bw.write(newPath + "\n");
                }
                for (int i = recent.size() - 1; i >= 0; i--) {
                    if (recent.get(i).equals(newPath)) {
                        continue;
                    }
                    bw.write(recent.get(i) + "\n");
                }
                bw.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        private void clearConfig() {
            recent = new Vector<String>();
            defaultIndex = -1;
            dropList.removeAllItems();
            writeConfig();
        }

        private String trim(String str) {
            if (str.charAt(str.length() - 1) == File.separatorChar) {
                return str.substring(0, str.length() - 1);
            } else {
                return str;
```

```
            }
      }

      @Override
      public void actionPerformed(ActionEvent e) {
            if (e.getActionCommand().equals("ok")) {
                  newPath = dropList.getEditor().getItem().toString();
                  //empty path
                  if (newPath.equals("")) {
                        JOptionPane.showMessageDialog(frame,
                                    "Please enter a valid path",
                                    "", JOptionPane.WARNING_MESSAGE);
                        return;
                  }
                  //incorrect path format
                  File dir = new File(newPath);
                  newPath = trim(newPath);
                  if
(!(newPath.equals(dir.getAbsoluteFile().toString()))) {
                        JOptionPane.showMessageDialog(frame,
                                    "Please enter a valid path",
                                    "", JOptionPane.WARNING_MESSAGE);
                        return;
                  }
                  //directory doesn't exist and can't create
successfully
                  if (!dir.exists() && !dir.mkdir()) {
                        JOptionPane.showMessageDialog(frame,
                                    "Please enter a valid path",
                                    "", JOptionPane.WARNING_MESSAGE);
                        return;
                  }
                  MainWindow.setWorkspace(newPath, null);
                  writeConfig();
                  frame.dispose();
                  MainWindow.WAIT_BLOCK.countDown();
            }
            if (e.getActionCommand().equals("cancel")) {
                  frame.dispose();
                  MainWindow.WAIT_BLOCK.countDown();
            }
            if (e.getActionCommand().equals("browse")) {
                  if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {

      dropList.getEditor().setItem(chooser.getSelectedFile().toStrin
g());
                  }
            }
```

```
            if (e.getActionCommand().equals("clear")) {
                clearConfig();
            }
        }

        @Override
        public void itemStateChanged(ItemEvent e) {
            Object source = e.getItemSelectable();
            if (source == defaultButton) {
                if (e.getStateChange() == e.SELECTED) {
                    defaultIndex = 0;
                }
                if (e.getStateChange() == e.DESELECTED) {
                    defaultIndex = -1;
                }
            }
        }
}
```