Domain-Specific Analog Accelerators for Artificial Intelligent Algorithms Implementation

by

Guoxin Liu, B. Eng

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of
the requirements for the degree of

Master of Applied Science, M.A.Sc

(Electrical and Computing Engineering)

Carleton University

Ottawa, Ontario

Abstract

This thesis discusses some circuit designs for AI algorithm acceleration. Instead of using digital computing components for algorithm implementations, this thesis describes new ideas to design and implement algorithms directly at the circuit-level. The first large section is about feedforward algorithm implementations that include using 1*4 analog multiply-accumulation arrays for DSP algorithm implementation and 2*3*3 analog multiply-accumulation matrices for computer vision algorithm and artificial intelligent algorithm implementations. The second large section concerns backward algorithm implementations that include using programmable resistor-based feedback loop, 'Add-division circuit' for convolutional kernel training algorithm implementation, and 'Random matrix generator' for solving Diophantine equations of neural networks.

Moreover, some ideas for the system improvements are mentioned. These include multiple systems expansion for constructing larger analog multiply-accumulation cores so that the system can implement more complicated algorithms and adding modulation and demodulation components for overlapping multiple inputs so that all convolution operations can be processed in parallel.

Acknowledgements

First, I would like to thank all the professors of graduated classes that I took in Carleton University: the supervisor of my M.A.Sc thesis Steven P. McGarry, professor of digital signal processing Sreeraman Rajan, professor of signal processing electronics Ralph Mason, professor of VLSI design Leonard MacEachern, professor of advanced topics in VLSI (radio frequency system) John Rogers, and professor of microelectronic sensors Ravi Prakash. Their classes gave me a lot of ideas and knowledge required for finishing this thesis.

Also, I would like to thank my parents. They provided financial support and helped me to purchase devices and books from China which could not be found in Canada. The progress of my experiments would not have been so smooth without their help.

Table of Contents

List of Tables

List of Figures

Nomenclature

GPU	Graphic processing unit
TPU	Tensor processing unit
VLSI	Very large-scale integration
IC	Integrated circuit
DSP	Digital signal processing
FIR	Finite impulse response
IIR	Infinite impulse response
ISP	Image signal processing
CNN	Convolutional neural network
V	Voltage potential [V]
R	Resistance [Ω]
I	Current [A]
C	Capacitance [F]
OPAMP	Operational amplifier
PCB	Printed circuit board
ADC	Analog-digital converter
DAC	Digital-analog converter
MUX	Multiplexer
SQNR	Signal quantization noise ratio
PID	Proportional integrational differential
FFT	Fast Fourier transform
MCU	Micro controller unit

Chapter 1

Introduction

1.1. Domain-Specific Accelerators for Algorithms

Conventional platforms for executing algorithms are digital systems which rely on logic-level circuits based on densely integrated transistors [1]. Their execution includes multiple steps such as reading memory, loading instructions and data, adding or multiplying the loaded data, and saving results [2].

Highly customized linear and non-linear analog systems can accelerate specific algorithms by circuit-level execution. For example, a 'near-sensor neural network' is a resistive hardware neural network integrated with sensors so that signal processing speed is accelerated by the pure analog signal system without the delays of digital-analog conversions [3][4]. For high performance computing applications, programmable analog computing units can substitute for transistor-based computing units for dense computing applications obtaining higher efficiency [5] [6] [7].

1.2. Analog Implementation

Nowadays, most of artificial intelligent applications are based on software that needs high power consumption processing units such as ARM platforms or X86 platforms.

For example, general purpose GPUs of Nvidia is used for parallel computing [8] and TPUs of Google [9] is used for neural network acceleration. Digital computing units are based on Boolean logic, but analog computing units are based on I-V relationship which leads much smaller sizes and higher efficiency.

For dense computation tasks such as artificial intelligence and digital signal processing this is especially true and large scale integrated analog accelerators can compete with state of art GPUs. For example, the research paper 'Fully hardware implemented memristor convolutional neural network' from 'Nature' shows that for convolutional neural network processing tasks, their memristor matrix system has "110 times better efficiency and 30 times better performance density compare with Tesla V100 GPU" [5].

1.3. Application Circumstances

Analog algorithm circuits can be applied for circumstances where we require low power consumption and mobility. For example, medical piezoelectricity powered in-body devices [10] and near-sensor neural networks [3][4]. Also, they can be applied for high performance computing circumstances. For example in integrated circuits using VLSI technology for artificial intelligence acceleration [7]. Another example is neuromorphic mixed systems which contain digital system for management and analog part for acceleration [5].

1.4. State of the Art Work

1.4.1. Resistor Matrices as Signal Filters

Memristor matrices for brain machine FIR filter: a brain wave filter came is described in a paper “Neural signal analysis with memristor arrays towards high-efficiency brain-machine interfaces” of Nature communications, August 25, 2020. It demonstrates a memristor-based FIR analog filtering system for brain-wave signal processing. The author mentioned that applying digital signal processing algorithms on Von Neumann architecture can not satisfy the increasing numbers of brainwave signal capturing devices today. A memristor-based near-sensor neural network is connected after the capturing devices directly for obtaining better processing performance. Test results show that power efficiency of the filter system is 400 times higher than state of art transistor-based systems [4].

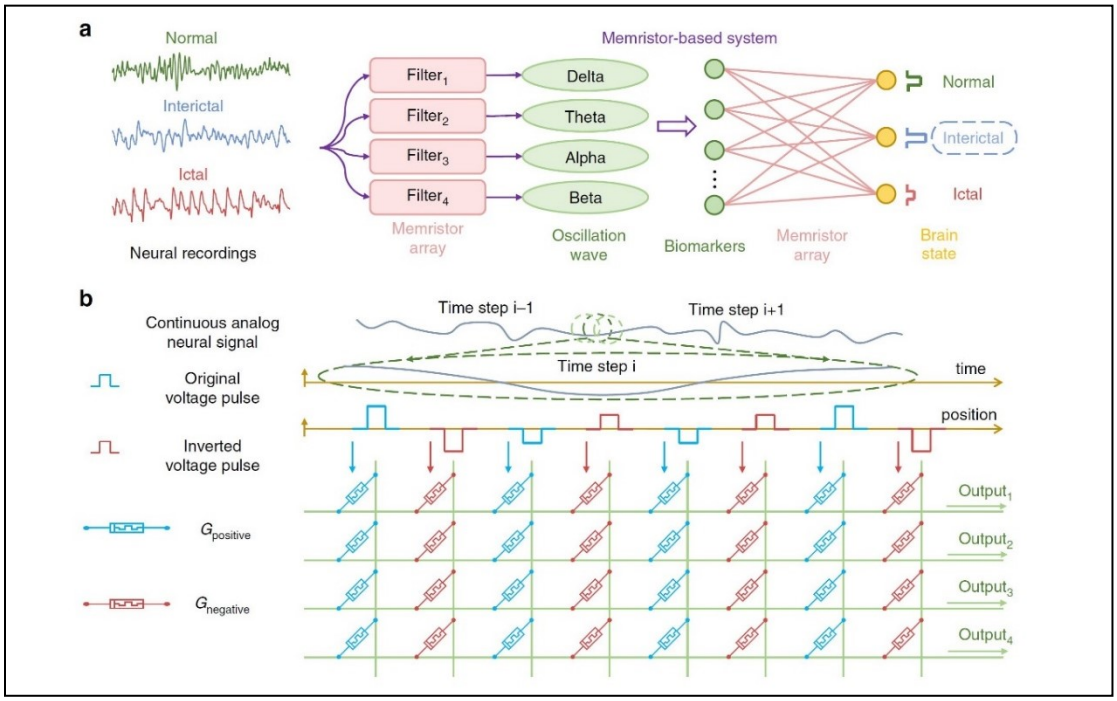


Figure 1.1 Architecture of the memristor-based FIR filter [4].

As shown in figure 1.1, the brainwaves are filtered by the memristor-based FIR filters. As shown in part 'b' of the figure, parameters of the digital filter are preloaded into the memristor matrix. As seen in part 'a' of the figure, the brainwaves are feed into the memristor neural network directly without analog-digital conversion. Next, the neural network output nodes feed decisions to the next devices.

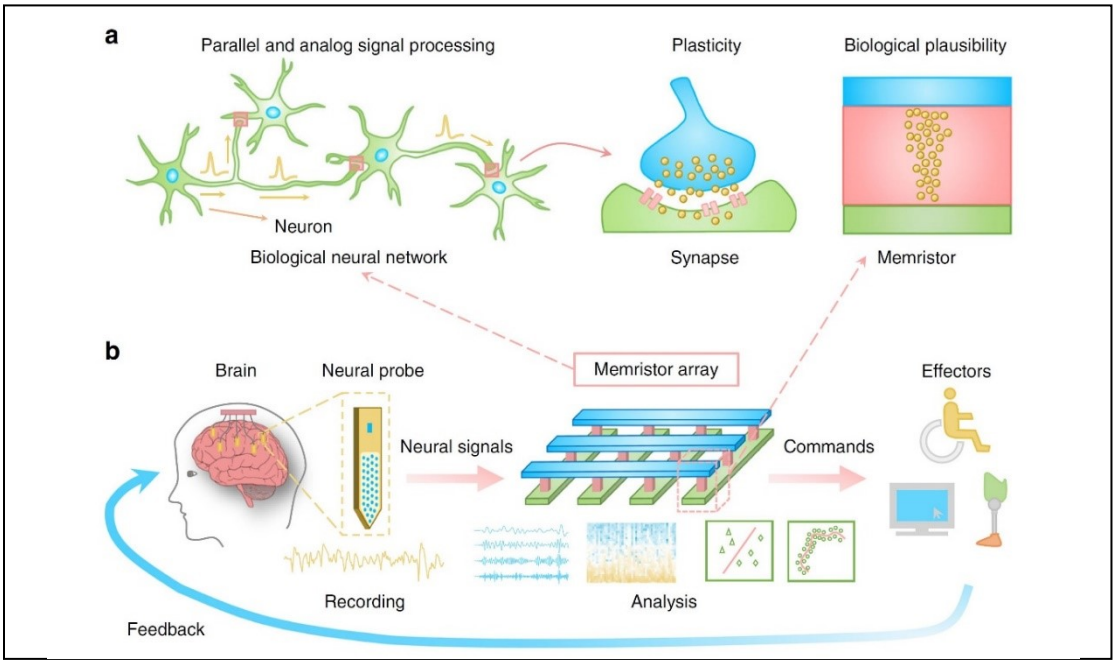


Figure 1.2 Memristor array-based brainwave filter application [4].

As the figure 1.2 shows the memristor array is connected after the neural probe so that the analog signal from the neural probe is captured and processed by the memristor array directly. Comparing with conventional methods which require analog-digital conversions, the filter with 'near sensor neural network' has higher efficiency. Also, instead of being a processing unit, the computer only collects the results.

1.4.2. One-Step Linear Regression

Memristor matrices and op-amps for solving regression problems of artificial intelligence: this memristor feedback loop came from a research paper "One-Step regression and classification with cross-point resistive memory arrays" of Science Advances, May13, 2020. The author demonstrated the memristor based new computing architecture for one-step computation of linear regression of artificial intelligence algorithms in the memristor memories [6].

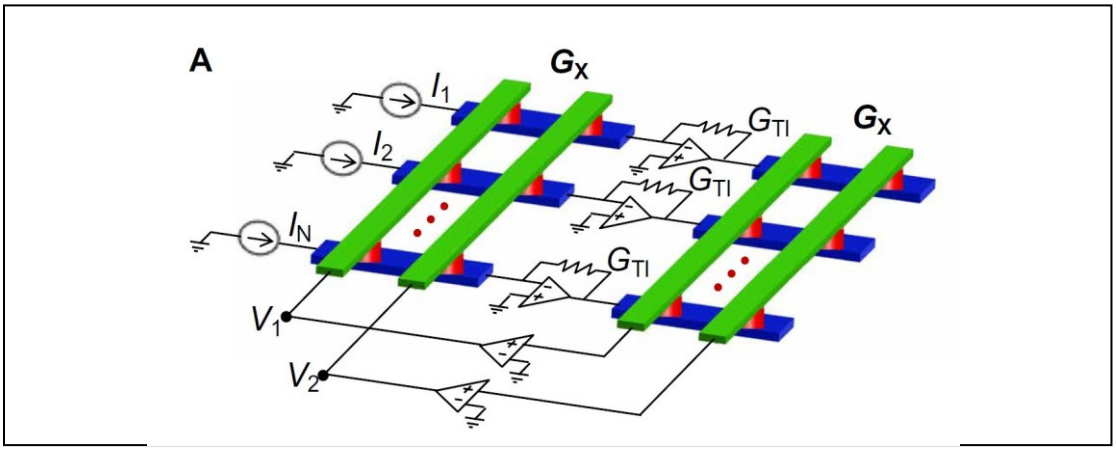


Figure 1.3 Memristor based matrix solver [6].

As shown in figure 1.3, the circuit utilizes plasticity characteristic of memristors and self-adjustment characteristic of feedback loops. Data is converted to analog signals and processed by the analog system on the circuit level. Next, the results are converted to digital data and read back to computers, which shows the computing capability of the analog circuit.

1.4.3. Fully Hardware Artificial Intelligent Neural Network

Large scale memristor matrices system for applying convolutional neural networks on the hardware level: This circuit implements convolutional neural network fully by memristor matrix. It came from the paper “Fully hardware-implemented memristor convolutional neural network” Nature, January 29,2020. The author demonstrated a memristor system for a convolutional neural network. Power efficiency and power density of the system are much better than the state of art GPU Tesla V100 for convolutional neural network applications [5].

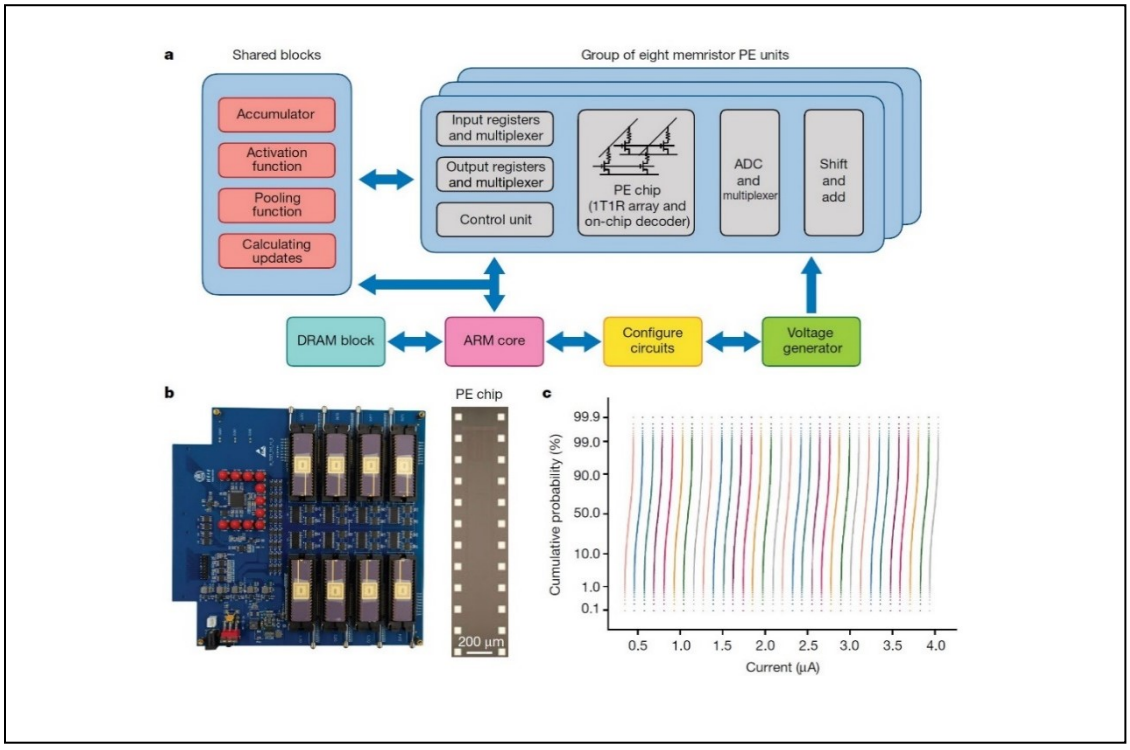


Figure 1.4 Architecture of the fully hardware implemented convolutional neural network [5].

As shown in the architecture block diagram of figure 1.4, the system contains eight processing elements (PE) which integrates a memristor array, input/output units, and

control units. A shared block is applied for implementing the general functions of convolutional neural networks such as accumulation, activation, and pooling. An ARM core acts as a master node of the system which dispatches all units of the system. Figure 1.5 shows that the major computing units of the system are the memristor arrays. Part a) shows the hardware implementing method: PE 1~3 cores are configured for the convolution layers and PE 5~7 cores are configured for the FC (fully-connected) layer. Parts b) and c) of the figure illustrate the convolution kernels are recorded by the 1T1R (1 transistor for control, 1 resistor for memorizing/processing) memristor matrix.

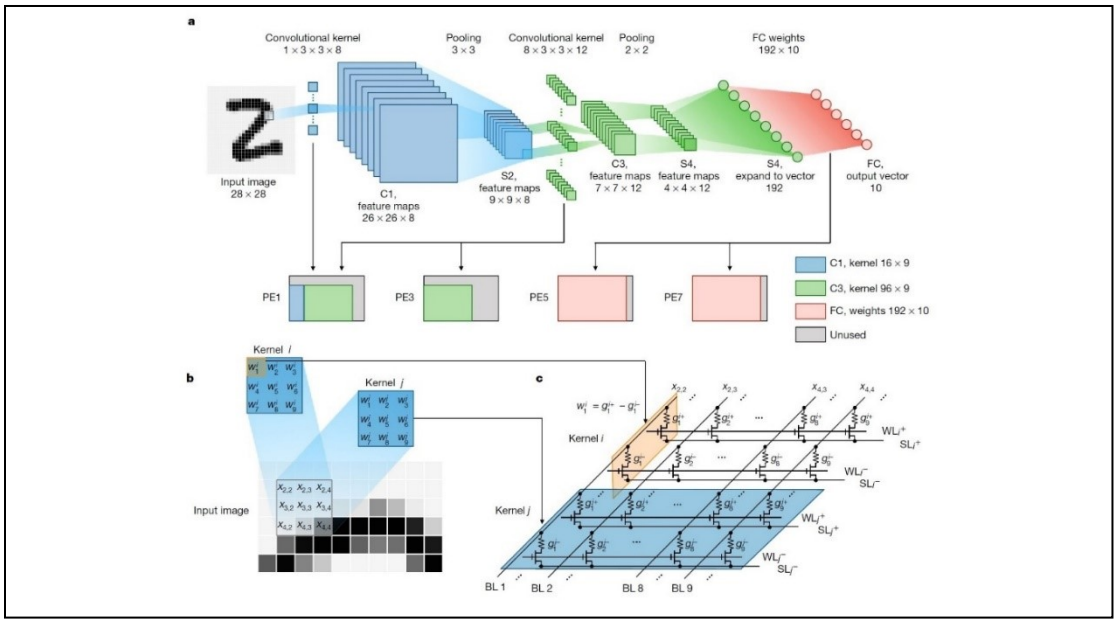


Figure 1.5 Principle of the memristor array operation [5].

1.5. Thesis Overview

Goal of this thesis is to implement algorithms on circuit-level. Instead of the conventional design methods of computing systems, circuit designs of this thesis are based on algorithms, such as digital signal processing and artificial intelligence. Chapter 2

will provide background of algorithms for digital signal processing (DSP), image signal processing (ISP), and convolutional neural networks (CNN). Chapter 3 will provide background for basic analog ICs and architectures for algorithm implementations such as op-amp circuits, Wheatstone bridges, and analog multiply-accumulation circuits. Chapter 4 describes using analog multiply-accumulation units for implementing feedforward architecture algorithms. Chapter 5 shows the use of programmable resistor-based feedback loops for implementing backward architecture algorithms. Chapter 6 provides some ideas for improvements that include signal enhancements, system expansion, and parallelism enhancements. Finally, chapter 7 and chapter 8 present some discussion and conclusions.

Chapter 2

Background for Digital Signal Processing, Image Signal Processing, and Convolutional Neural Networks

2.1. Theories of Digital Signal Processing

2.1.1. Introduction for Digital Signal Processing

Digital signal processing is based on discrete time by sampling continuous signals. Digital signal processing relies on digital computation system components such as accumulators, multipliers, registers, memories. The processed data is stored in memories or output through a DAC.

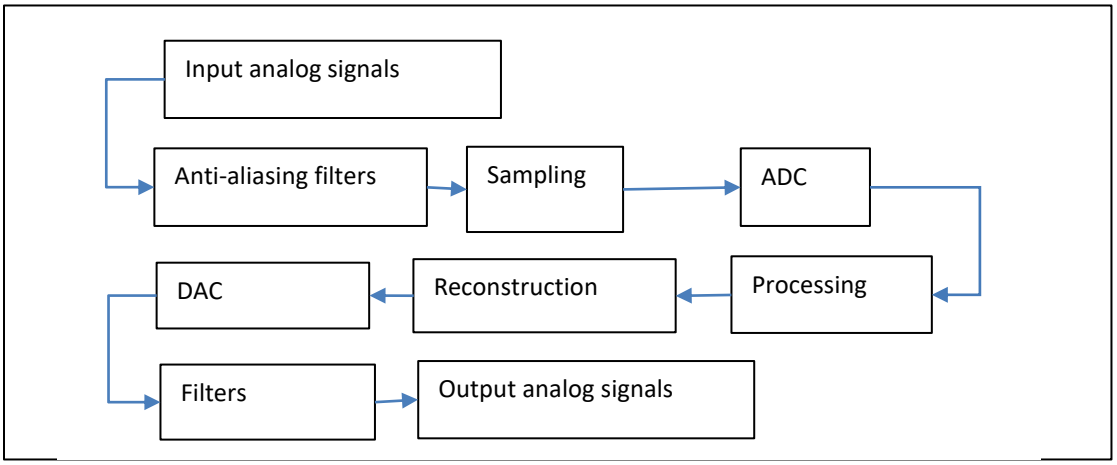


Figure 2.1 Digital signal processors architecture [12].

Figure 2.1 shows digital signal processing for general signals. Firstly, the signals feeding in pass through an anti-aliasing filter so that its input frequencies are lower than 2*sampling frequency (Nyquist rate) [11]. Then, the signals are sampled to convert the

signal from continuous-time domain to discrete-time domain. Next, the discrete-time analog signals are converted to discrete-time digital signal for the digital processing units. Next, some digital signals processing algorithms are applied by the multipliers and accumulators of the processors. Next, the digital discrete-time signals are applied by reconstructing algorithms for reconstruction into analog signals. Next, the digital signals are converted to analog signals. Finally, some filters are used for removing noise of the system [12].

2.1.2. Discrete-Time Signals and Continuous-Time Signals

Continuous-Time Signal

Continuous-Time signal is the signal that is changing continuously in the continuous time domain. From an electrical perspective, all the analog circuits' outputs are continuous-time signals. For example, cos(*t*), sin(*t*), sinc(*t*), etc. [13].

Discrete-Time Signal

Discrete-time signals are the signals in the discrete time domain that are usually represented by integers. Discrete-time signals are generated by sampling continuous-time signals in digital signal processers [14].

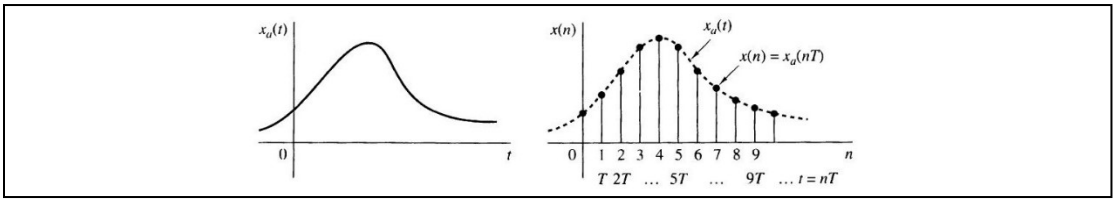


Figure 2.2 Continuous-time signal (left) and discrete-time signal (right) comparison [15]

Figure 2.2 shows an example of continuous-time signal and discrete-time signal that was sampled by samplers.

Normalized Frequency

Consider continuous-time cosine signals that are converted to discrete-time signal by a sampler (eqn. 2.1).

$$\cos(n) = \cos\left(\frac{2*\pi*F*n}{F_s}\right) = \cos\left(2 * \pi * f * n\right) \quad \textbf{(2.1)}$$

Where n is the index of discrete-time signals, F is the frequency of continuous-time signal and F_s is the sampling frequency of the sampler, f is normalized frequency [16].

The frequency of continuous-time signal is normalized by samplers, and it is called 'normalized frequency' with the relationship equation (eqn. 2.2).

$$f = \frac{F}{F_s} \quad \textbf{(2.2)}$$

Where f is normalized frequency, F is the frequency of continuous-time signal and Fs is the sampling frequency [16].

The sampling frequency should be higher than the 'Nyquist rate' [11] so that 'aliasing' is avoided during the sampling process (eqn. 2.3).

$$F_s \geq 2 * F \text{ or } -\frac{1}{2} \leq f = \frac{F}{F_s} \leq \frac{1}{2} \quad \textbf{(2.3)}$$

Where F is the continuous-time signal frequency and Fs is sampling frequency [16].

From the equation can get frequency range of the normalized frequency [16] (eqn. 2.4).

$$-\frac{1}{2} \leq f \leq \frac{1}{2} \text{ or } -\pi \leq w \leq \pi \quad \textbf{(2.4)}$$

2.1.3. Convolution of Two Matrices

Most operations of digital signal processing are based on convolution which reflects systems' responses under an input signal. Convolution is based on equation 2.5 [17].

$$\mathrm{y}(n) = \sum_{\mathrm{k}=-\infty}^{+\infty} \mathrm{x}(k) * \mathrm{h}(n-k) \quad \textbf{(2.5)}$$

Where y(n) is the output matrix, x(k) is the input matrix, and h(n-k) is the impulse response of the system.

2.1.4. Correlation and Normalization

Cross-correlation shows two matrices' similarity by equation 2.6 [18]. Higher cross-correlation means higher similarity and vice versa.

$$\mathrm{r}_{\mathrm{yx}}(l) = \sum_{\mathrm{n}=-\infty}^{+\infty} \mathrm{y}(n) * \mathrm{x}(n-l) \quad \textbf{(2.6)}$$

Where x(n-l) and y(n) are two matrices, l is the shifting distance between the two matrices (When l=0, r_{yx}=**y·x**).

When input matrix x(n) is equal to matrix y(n), cross-correlation of the matrices is called auto-correlation (eqn. 2.7).

$$\mathrm{r}_{\mathrm{xx}}(l) = \sum_{\mathrm{n}=-\infty}^{+\infty} \mathrm{x}(n) * \mathrm{x}(n-l) \quad \textbf{(2.7)}$$

Where x(n-l) and x(n) are the same input matrix, x(n-l) is shifted by l.

Correlation of two matrices should be calculated under the same scale. Two

matrices with different scales should be normalized to range [-1,1] before calculating their correlation.

Normalization is a common operation in digital signal processing tasks which rescales input signals to the same ranges for further processing [19]. For auto-correlation operations, normalized matrices are expressed by equation 2.8.

$$\rho_{xx}(l) = \frac{r_{xx}(l)}{r_{xx}(0)} \quad \textbf{(2.8)}$$

Where $\rho_{xx}(l)$ represents normalized auto-correlation matrix, $r_{xx}(l)$ represents original auto-correlation matrix, $r_{xx}(0)$ represents divider (the highest correlation occurs at l=0 where the two matrices are the same in both time and amplitude).

For cross-correlation operations, normalized matrices are expressed by equation 2.9.

$$\rho_{xy}(l) = \frac{r_{xy}(l)}{\sqrt{r_{xx}(0) * r_{xy}(0)}} \quad \textbf{(2.9)}$$

Where $\rho_{xy}(l)$ represents normalized cross-correlation matrix, $r_{xy}(l)$ represents original auto-correlation matrix, $\sqrt{r_{xx}(0) * r_{xy}(0)}$ represents divider (the highest correlation result).

2.1.5. Frequency Response and Z-domain

The demonstrated equations of above sections are in the discrete-time domain. Most digital signal processing algorithms designs are in the frequency domain [20]. Recalling the convolution operation of section 2.1.3. Assuming a random signal x(n) passes through a system which has impulse response h(n), the output y(n) is equation

2.10.

$$y(n) = \sum_{k=-\infty}^{+\infty} x(k) * h(n-k) = \sum_{k=-\infty}^{+\infty} h(k) * x(n-k) \quad \textbf{(2.10)}$$

When x(n) is a sine wave, x(n) can be expressed by equation 2.11

$$x(n) = A * e^{jwn} \qquad -\infty < n < \infty \quad \textbf{(2.11)}$$

Where n is an index of discrete-time, w is a normalized angle frequency in range [-π,π], and A is the amplitude of the sine wave.

The output y(*n*) becomes equation 2.12,

$$y(n) = \sum_{k=-\infty}^{+\infty} h(k) * A * e^{jw(n-k)} = A * [\sum_{k=-\infty}^{+\infty} h(k) * e^{-jwk}] * e^{jwn} \quad \textbf{(2.12)}$$

When we observe frequency components of a signal at a specific time n, $A * e^{jwn}$ can be regarded as a parameter. The equation becomes equation 2.13,

$$H(e^{jw}) = \sum_{k=-\infty}^{+\infty} h(k) * e^{-jwk} \quad \textbf{(2.13)}$$

Where $H(e^{jw})$ represents the frequency combination of the signal, h(*k*) is a parameter for its corresponding frequency component, and e^{-jwk} is a frequency component.

For simplifying calculations, most frequency domain calculations are in the z-domain by use of the z-transform. Z-domain is also based on frequency components, but the calculation focuses on signals' amplitudes. In other words, signals' phases can only be calculated by their frequency response equations. The above frequency equation has the equation in the z-domain (eqn. 2.14),

$$H(z) = \sum_{k=-\infty}^{+\infty} h(k) * z^{-k} \quad \textbf{(2.14)}$$

Where h(*k*) is a parameter for its corresponding frequency component, z represents e^{jw}.

2.1.6. FIR System

There are two categories of DSP systems which are finite impulse response (FIR) and infinite impulse response (IIR). Finite impulse response (FIR) systems do not need memory devices since their outputs only relate to inputs. Infinite impulse response (IIR) systems need memory devices for output feedback [21]. FIR systems are suitable for the hardware of this thesis for simplifying the system architecture.

Equation 2.15 shows impulse response of a general FIR system in the z-domain [22]

$$\mathrm{H(z)} = \sum_{\mathrm{k=0}}^{\mathrm{N}} \mathrm{h}(k) * \mathrm{z}^{-k} \quad \textbf{(2.15)}$$

Where h(k) are the parameters of the FIR filter and z^{-k} is the frequency in z-domain. Its impulse response in the discrete-time domain is given by equation 2.16 [23].

$$\mathrm{h}(n) = \sum_{\mathrm{k=0}}^{\mathrm{N}} \mathrm{h}(k) * \delta(n-k) \quad \textbf{(2.16)}$$

Where h(k) are the parameters of the FIR filter and δ (n-k) is an impulse function.

When applying a FIR filter to an input discrete-time signal, its output discrete-time signal y(n) is equation 2.17.

$$\mathrm{y}(n) = \mathrm{x}(n) \otimes \mathrm{h}(n) \quad \textbf{(2.17)}$$

Where y(n) is the convolution result, x(n) is the input matrix, and h(n) is impulse response of the filter.

2.1.7. FIR High Pass and Low Pass Filters

FIR systems can implement high pass and low pass filters in discrete-time domain.

Ideal discrete filters: consider two ideal digital high pass and low pass filters. The digital filters have frequency range [-π， π] according to the equation 2.4. As shown in figure 2.3, for digital filters, low pass filters have - $w_c \leq w \leq w_c$ and high pass filters have - $\pi \leq w \leq wc$ or $w_c \leq w \leq \pi$ [24].

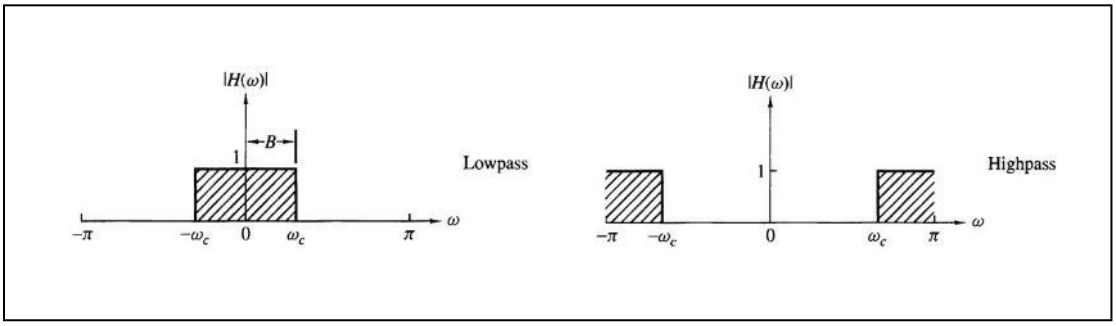


Figure 2.3 Ideal high and low pass digital filters frequency responses [24].

Ideal filters cannot be implemented by causal systems. In other words, it is impossible to process real-time signals. For example, an ideal low pass filter has frequency response [25] (eqn. 2.18)

$$\mathrm{H}(w) = \begin{cases} 1 & , |w| \leq \mathrm{w_c} \\ 0 & , \mathrm{w_c} < |w| \leq \pi \end{cases} \quad \textbf{(2.18)}$$

Where, w_c is corner frequency of the filter, H(w) is the amplitude of the filter' frequency response. The low pass filter has impulse response at discrete-time domain (eqn. 2.19).

$$x(n) = \begin{cases} \frac{w_c}{\pi} & , n = 0 \\ \frac{w_c}{\pi} * \frac{\sin(w_c * n)}{w_c * n} & , n \neq 0 \end{cases} \quad \textbf{(2.19)}$$

The function $\frac{\sin(w_c * n)}{w_c * n}$ is also called sinc function which is not a periodic function and exists at discrete-time coordinate n= $(-\infty, \infty)$. Obviously, the filter needs cover all the time coordinates, but devices cannot receive signals from the future. Practical digital filters do not have sharp frequency responses as equation 2.18, but they are realizable.

Practical FIR filters design: the process of designing FIR filters is calculating parameters of required filters' impulse responses. Difference equations, frequency response equations, and zero-pole equations are three equations that should be considered during design.

In the discrete-time domain, filters' impulse responses can be expressed by equation 2.20 [26]

$$y(n) = -\sum_{k=1}^{N} a_k * y(n-k) + \sum_{k=0}^{M} b_k * x(n-k) \quad \textbf{(2.20)}$$

Where a_k are parameters of y(n-k) and b_k are parameters of x(n-k). For FIR filters, a_k are 0 (FIR systems do not rely on feedbacks) and b_k are the required parameters found during design.

In the frequency domain, filters' frequency responses are expressed by equation 2.21 [27].

$$H(z) = \frac{\sum_{k=0}^{M} b_k * z^{-k}}{1 + \sum_{k=1}^{N} a_k * z^{-k}} \quad \textbf{(2.21)}$$

The parameters are the same, a_k are parameters of y(n-k) and b_k are parameters of x(n-

k). For FIR filters, a_k are 0 (FIR systems do not rely on feedbacks) and b_k are the required parameters found during design.

We can convert equation 2.21 to equation 2.22 which is the primary equation that should be considered during digital filter designs [27].

$$H(z) = b_0 * \frac{\prod_{k=1}^{M}(1-z_k*z^{-1})}{\prod_{k=1}^{N}(1-p_k*z^{-1})} \quad \textbf{(2.22)}$$

Where z_k are required zeros and p_k are required poles. For FIR filters, p_k is always 0, z_k are selected zeros found during design.

FIR filters frequency response can be set by placing poles and zeros on pole-zero plots. Placing poles for intensifying the magnitude of outputs at specific frequencies and placing zeros to attenuate the magnitude of outputs at specific frequencies. After confirming equation 2.22, it should be converted to equation 2.21, which is the normal form of the frequency response equation. Next, converting equation 2.21 to equation 2.20 based on a_k and b_k. Finally, the system can be realized by referring equation 2.20. Actual designs for this thesis are shown in section 5.1.

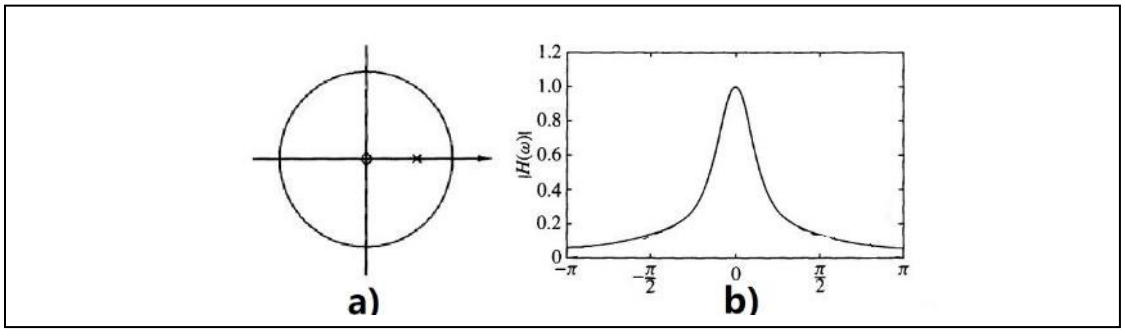


Figure 2.4 An example of digital low pass filter [28].

Figure 2.4 shows a zero-pole plot of a digital low pass filter and a magnitude plot of its frequency response [28]. The filter contains a pole at *w*=0 for amplifying signals with frequencies around *w*=0 so that attenuating signals with higher frequencies.

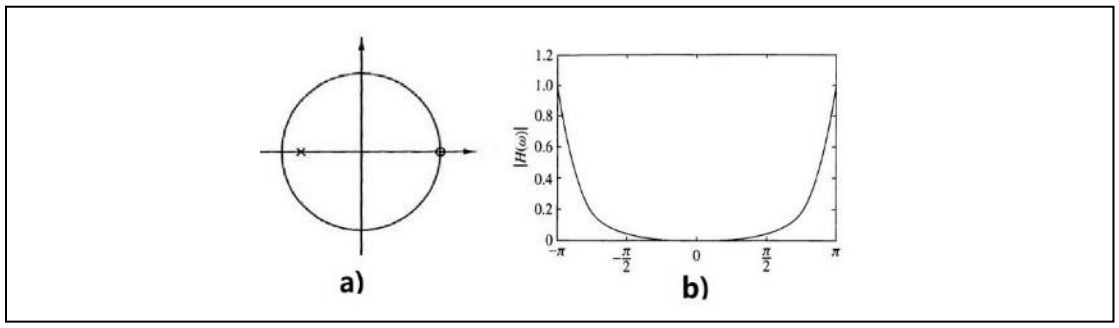


Figure 2.5 An example of digital high pass filter [28].

Figure 2.5 shows a zero-pole plot of a digital high pass filter and magnitude of its frequency response [28]. The filter has a pole at *w*= π for amplifying signals with frequencies around π and a zero at *w*=0 for attenuating signals with frequencies around 0.

2.1.8. Down-Sampling

Down-sampling is a common operation in digital signal processing for reducing processing loads of system by 'decimation' [29].

We consider a digital processing system which has input matrix x(n) and system impulse response h(n). The output will be equation 2.23.

$$\mathrm{v}(n) = \sum_{\mathrm{k}=0}^{\infty} \mathrm{h}(k) * \mathrm{x}(n-k) \qquad \textbf{(2.23)}$$

Next, the output matrix v(n) is passed through a down-sampling module which take a

point every D points. The output is equation 2.24.

$$y(m) = v(D * m) = \sum_{k=0}^{\infty} h(k) * x(D * m - k) \quad \textbf{(2.24)}$$

As the equation shows, if the input matrix has N elements the output of down-sampling has N/D elements.

Down-sampling is also suitable for decreasing processing loads of image processing. The 'Pooling' operation of convolutional neural network has the same principle as down-sampling by extracting the highest value or mean value from each 2*2 matrix (2*2 pooling).

2.1.9. Windows of FIR Filters

As mentioned above, the digital filters can process signals with fixed length. In practice, most real-time signals have unknown length. For example, applying digital filters for sound signals from microphones, which are used for real-time capturing of speech. 'Windows' can separate the signals into fixed length so that they are suitable for the filters [30].

In the discrete-time domain, assuming a window has length M, its equation is equation 2.25.

$$w(n) = \begin{cases} 1, & n = 0,1,\dots,M-1 \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(2.25)}$$

If a FIR filter with impulse response $h_d()$ has a window function w(n), its impulse response is equation 2.26.

$$h(n) = h_d(n) * w(n) = \begin{cases} h_d(n), & n = 0,1,\dots,M-1 \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(2.26)}$$

Where $h_d(n)$ is impulse response of the original filter, h(n) is impulse response of the original filter with the window.

2.2. Fundamentals of Image Signal Processing

2.2.1. Introduction of Image Signal Processing

Digital images are 2D digital signals that are processed by similar processes as figure 2.1. However, outputs of image signal processing are recorded in memory directly without signal reconstructions for most computer vision applications.

2.2.2. Image Signals from Sensors

Normal cameras contain three parts that are the sensor matrix, digital signal processors, and communication components. Light with different wavelengths and intensities cause different output voltage from each sensor. Next, a digital signal processor captures the voltage signal from each sensor and converts them to a digital signal. Finally, the parallel data from light sensor array is converted to serial data for output through I2C, SPI, UART etc. Image capturing devices can be a single light sensor, a sensor stripe, or a sensor matrix [31]. Figure 2.6 shows a light sensor that converting energy from light to analog signal.

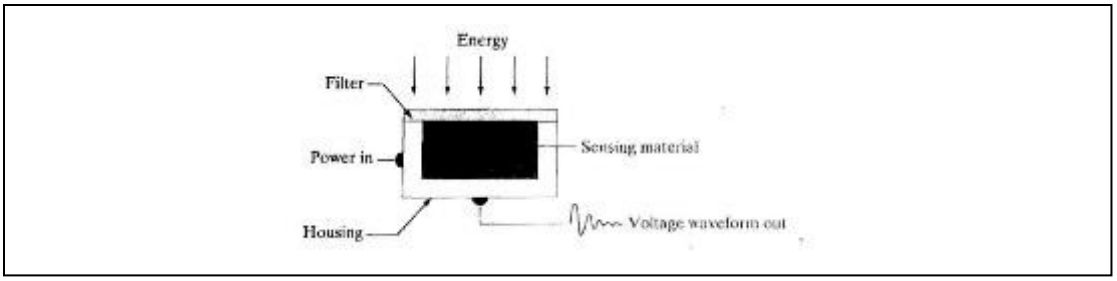


Figure 2.6 Image signals from sensors [31].

2.2.3. Frequencies of Image Signals

The digital signals from image sensors are received by computers and recorded as specific values. In a program like MATLAB, grey levels of pixels of mono-color images are represented by 0 (black)-255 (white). Digital image signals have frequencies when they are processed partially by image signal processors which is similar as taking windows for the signals (described in section 2.1.8).

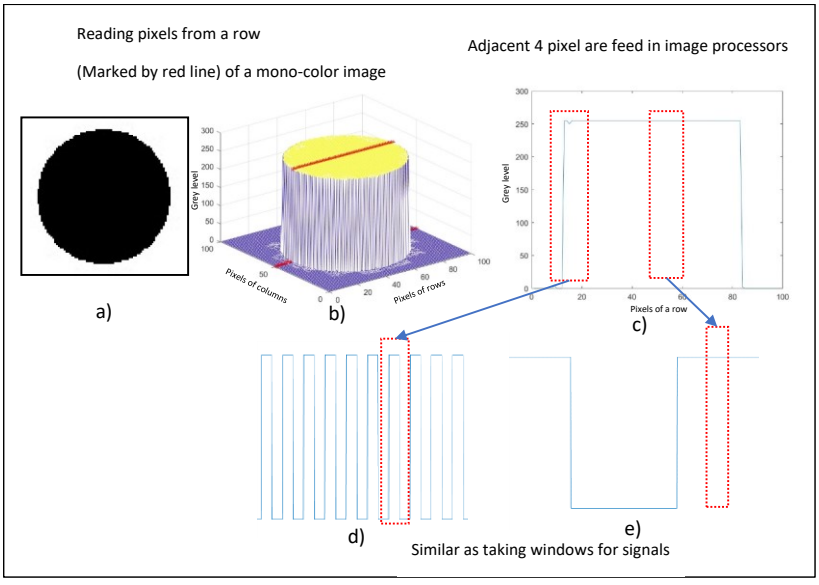


Figure 2.7 Frequencies of reading a mono-color image (for better illustration, grey levels =255-Actual Grey levels). a) The mono-color image b) plotting pixels and grey levels of the image in the 3D space c) partially processed by image processors is similar as taking windows for real-time signals d) partial signal that is treated as high frequency e) partial signal that is treated as low frequency.

Figure 2.7 shows an example of reading pixels of a row of a mono-color image for processing. For better illustration, the actual grey level 0 (black)-255 (white) is converted to 0 (white)-255 (black). An image processor reads and processes signals from a row of the image partially which is similar to taking windows for it. The signal of an edge shows high frequency feature in the window (as in figure 2.7 d) and the signal of the flat area shows low frequency feature in the window (as in figure 2.7 e).

2.2.4. FIR Filters for Image Processing

Image signals can be processed by digital filters since they have frequency content.

One-dimensional convolution: A 1D image signal array is generated by reading an image sequentially. Figure 2.8 shows processes of extracting edges of a row of the image (filter designs are explained in chapter 5).

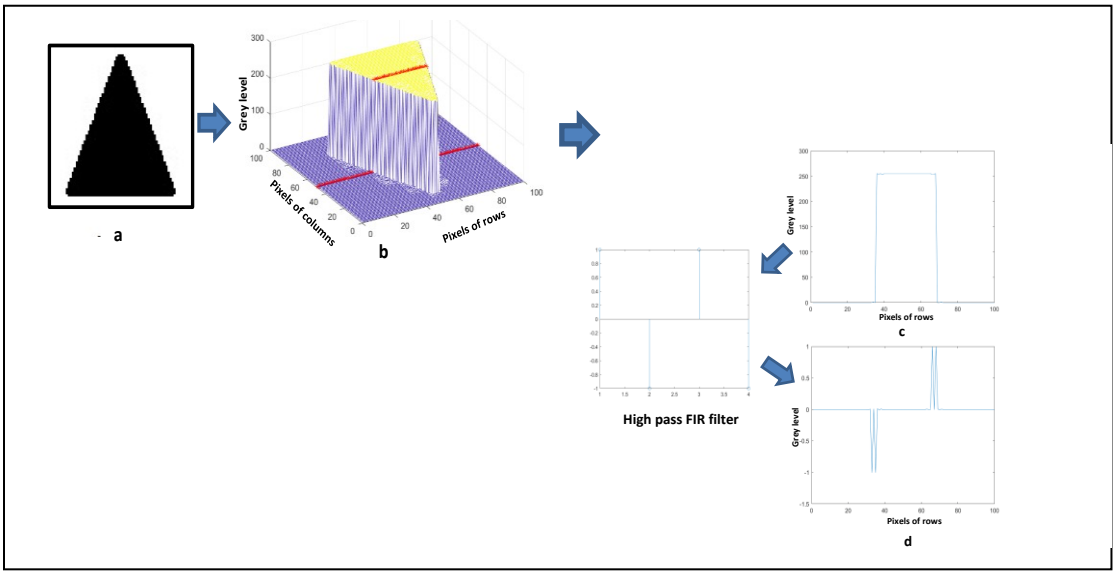


Figure 2.8 High pass filter for edges extractions a) a mono-color image of a triangle b) reading signal from a row of the image (for better demonstration white=0 black=255) c) plotting the signal d) edges detection result.

An image processor reads the signal partially from a row of a mono-color image and feeds in a FIR high pass filter. The filter only responds and outputs high frequency areas and neglects other areas. The high pass filters' design processes are explained in chapter 5.

Two-Dimensional Convolution: A 2D signal matrix is generated by a light sensor matrix. 2D convolution applies 1D convolution on rows and columns of images (eqn. 2.27).

$$\mathrm{x}(i,j) \otimes \mathrm{h}(i,j) = \sum_{\mathrm{m}=0}^{\mathrm{M}-1} \sum_{\mathrm{n}=0}^{\mathrm{N}-1} \mathrm{x}(m,n) * \mathrm{h}(i-m, j-n) \qquad \textbf{(2.27)}$$

Where x(i,j) is a pixel of an input matrix **x**, h(i,j) is an element of a 2D filter **h**, M and N are dimension of the filter [32]. This thesis does not focus on designing 2D image filters and tests only consider and implement 2D filters of the computer vision field.

2.3. Fundamentals of Convolutional Neural Networks

2.3.1. Introduction for Convolutional Neural Networks

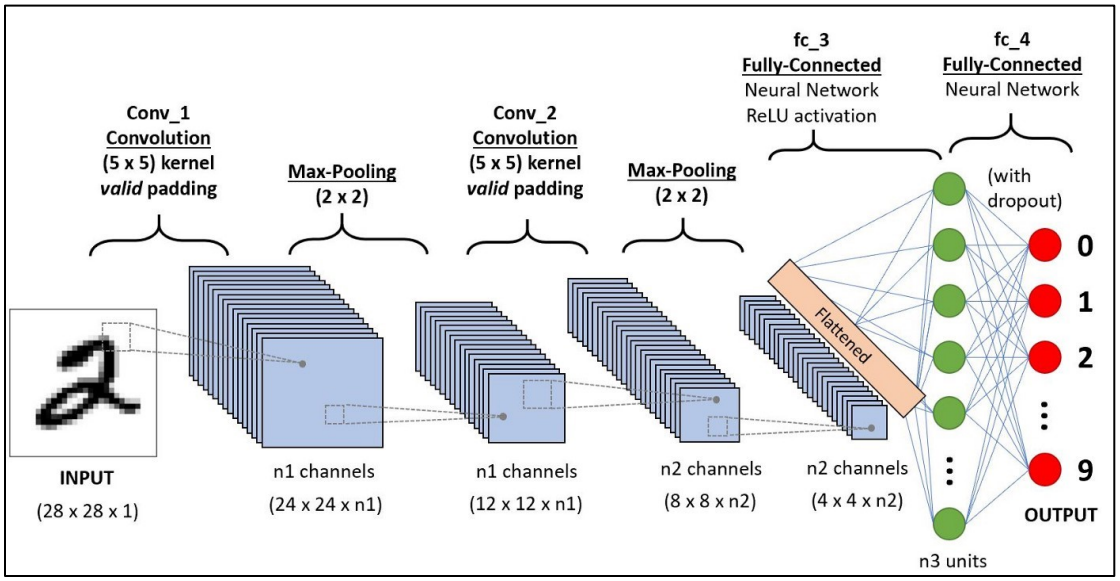


Figure 2.9 Structure of a convolutional neural network [33].

Convolutional neural networks are based on the image processing method of applying large numbers of self-learning digital image filters for feature extraction. Convolutional neural networks contain multiple convolutional layers, pooling layers, and activation units. 'Convolutional layers' are used for extracting patterns by self-learning convolution kernels. 'Pooling layers' down sample the convolution results for condensing the information. The 'Fully-connected' layer is a one-dimensional neural network for decision making [34].

2.3.2. Convolutional Layers

Convolutional layers are inspired by the human neuron. A convolutional layer always has multiple convolution kernels that have similar function as the "receptive field" [35] (figure 2.10).

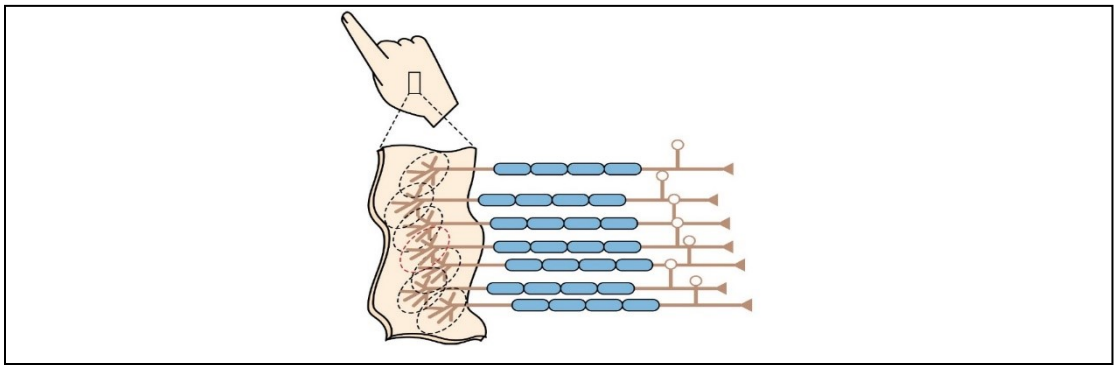


Figure 2.10 Convolutional neural network is inspired by the "receptive field" [35].

They can be regarded as image filters from an image signal processing perspective

but most of them are not understandable by humans. Sizes of convolution kernels are varied but the common size is 3*3 that is a trade-off between flexibility and effective areas.

Figure 2.11 shows a convolutional layer which is used for feature extraction. Convolutional kernels can be treated as image signal processing filters for specific features which may not be recognizable by humans [36].

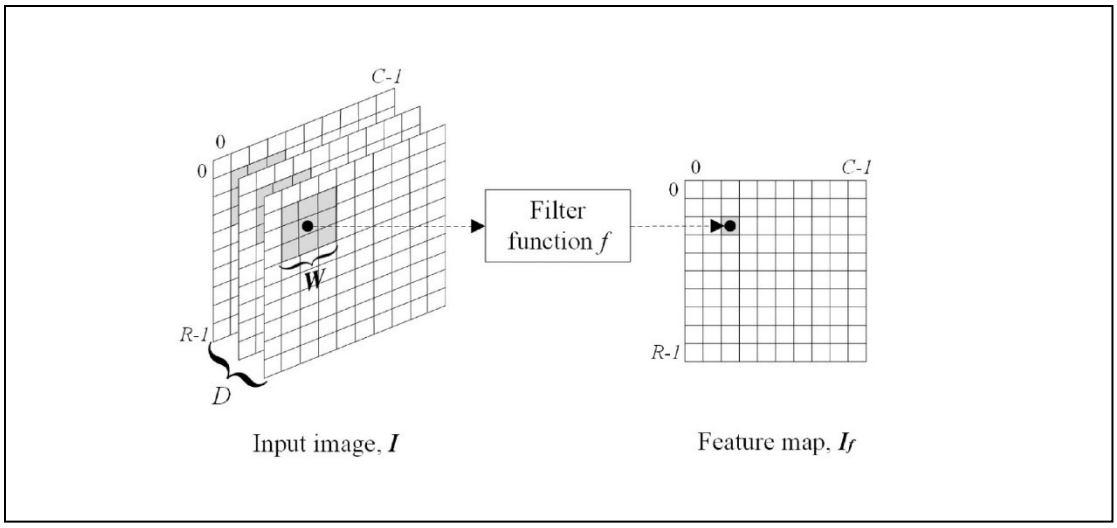


Figure 2.11 An example of a 3*3 convolutional layer [36].

Equation 2.28 shows the convolution operation between an input matrix and a M*N convolution kernel,

$$\mathrm{x}(i,j) \otimes \mathrm{w}(i,j) = \sum_{\mathrm{j}=0}^{\mathrm{N}-1} \sum_{\mathrm{i}=0}^{\mathrm{M}-1} \mathrm{x}(i,j) * \mathrm{w}(i,j) \qquad \textbf{(2.28)}$$

Where x(*i,j*) is a pixel of an input matrix **X**, w(*i,j*) is an element of a 2D weight matrix **W**, M and N are dimensions of the convolution kernel. Convolution kernels can extract different features through training algorithms. 'Gradient descent' algorithm is a common

method for decreasing error between actual outputs and expect outputs [36].

2.3.3. Pooling Layers

The convolution operations' results are down sampled by pooling layers in order to extract meaningful information for the next layer "Fully connected layer". A common pooling method is "Maximum pooling" that leaves the maximum number of a small matrix, for example, 2*2 pooling outputs the maximum number of each 2*2 matrix of the convolution results [37]. Figure 2.12 illustrates w*w pooling operations which output the maximum number of each w*w submatrix. The output matrix of the pooling layer is the input matrix for the following fully-connected layers [37].

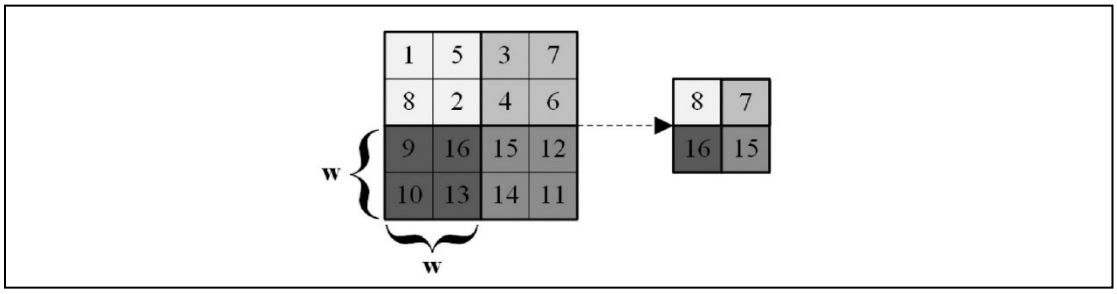


Figure 2.12 An example of a 'max pooling' layer [37].

2.3.4. Fully Connected Layers

After the convolutional layers and pooling layers, the input image has been classified according to patterns and down sampled so that the fully connected layer has sufficient computing resources. All the inputs nodes of a fully connected layer are connected to each of the output nodes. The number of input nodes are limited, and each

connection is assigned to a weight [38].

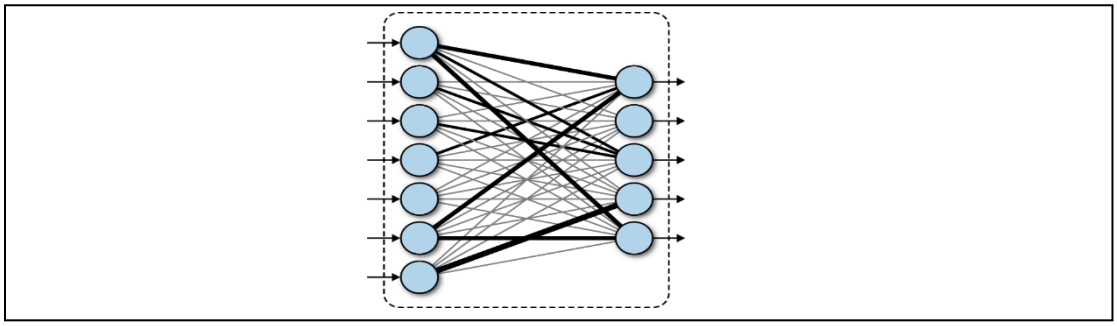


Figure 2.13 An example of a fully-connected layer [39].

As shown in figure 2.13, all input nodes of a fully-connected layer are connected to all output nodes and each connection has a weight. A fully-connected layer relies on the same calculation as the convolutional layer but it is for 1-dimensional data. Assuming M input nodes of a fully-connected layer are connected to an output node y through a weight matrix w, when j=1, equation 2.18 becomes,

$$\mathrm{y}(n) = \mathrm{x}(n) \otimes \mathrm{w}(n) = \sum_{n=0}^{\mathrm{M}-1} \mathrm{x}(n) * \mathrm{w}(n) \qquad \textbf{(2.29)}$$

For neural networks which contain multiple convolution layers and fully-connected layers. Each output node of a fully-connected neural network is feed into an activation function node for making decisions. Assuming an input matrix has size 1*N, and a weight matrix has size N*M, the fully-connected neural network has output equation,

$$y = \begin{cases} \sigma\left(x_1 * w_{1,1} + \cdots + x_N * w_{1,M}\right) \\ \quad . \\ \quad . \\ \quad . \\ \sigma\left(x_1 * w_{N,1} + \cdots + x_N * w_{N,M}\right) \end{cases} \qquad \textbf{(2.30)}$$

Where σ() is a nonlinear function, and $\sum_{\mathrm{j}=1}^{\mathrm{M}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{x}_i * \mathrm{w}_{i,j}$ are outputs of the fully-connected neural network [40].

2.4. Comparing DSP, ISP, and CNN

2.4.1. Different Convolution Operations

Convolution operations of convolutional neural networks have reversed order because convolution kernels are reversed automatically during training. In other words, convolution kernels of convolutional neural networks are only understandable for computers, but convolution kernels of digital signal processors are also understandable for humans.

Table 2.1 Comparing convolution operations.

	Equations
Convolution of DSP	$\mathrm{y}(n) = \sum_{k=-\infty}^{+\infty} \mathrm{x}(k) * \mathrm{h}(n-k)$ **(2.5)**
Convolution of image signal processing	$\mathrm{x}(i,j) \otimes \mathrm{h}(i,j) = \sum_{m=0}^{\mathrm{M}-1} \sum_{n=0}^{\mathrm{N}-1} \mathrm{x}(m,n) * \mathrm{h}(i-m, j-n)$ **(2.27)**
Convolution of convolutional neural network	$\mathrm{x}(i,j) \otimes \mathrm{w}(i,j) = \sum_{j=0}^{\mathrm{N}-1} \sum_{i=0}^{\mathrm{M}-1} \mathrm{x}(i,j) * \mathrm{w}(i,j)$ **(2.29)**

2.4.2. Causal and Non-Causal Systems

Signal systems have two categories: causal and non-causal. Causal systems' outputs only relate to current and past inputs, but non-causal systems' outputs relate to current, past, and future inputs [41]. In practice, real-time processing systems are causal

systems and off-line processing systems are non-causal systems. This thesis only considers off-line image signal processing systems that applying filters for images prestored in hard drives.

2.4.3. Algorithm Realization

Algorithms of DSP, ISP, and CNN methods require the same computations: multiply-accumulation for convolution operations. Meanwhile, if we compare DSP with ISP, the delay modules and reconstructing modules are not required by most image signal processing applications. Comparing DSP with CNN, ‘pooling’ of CNN has the same principle as ‘down-sampling’ of DSP, but CNN needs an extra non-linear component for decision making.

Chapter3

Background of Analog ICs for Signal Processing

3.1. Analog ICs for Digital Signal Processing Algorithms

As described in chapter 2, algorithm implementations rely on multiple digital accumulators and multipliers. Figure 3.1 shows a common architecture of computing sections of digital signal processors. The input analog signal is sampled and converted to digital value x[n] by ADC. Next, the memories of the system record the past two values x[n-1] and x[n-2] for convolution. Next, the current input x[n] and the past inputs x[n-1] and x[n-2] are multiplied by impulse response parameters of the system, which are h[0], h[1], and h[2]. Finally, the multiplied results are summed and outputted for signal reconstruction. The output matrix y[n] is the result of convolution x(n) ⊗ h(n) [42].

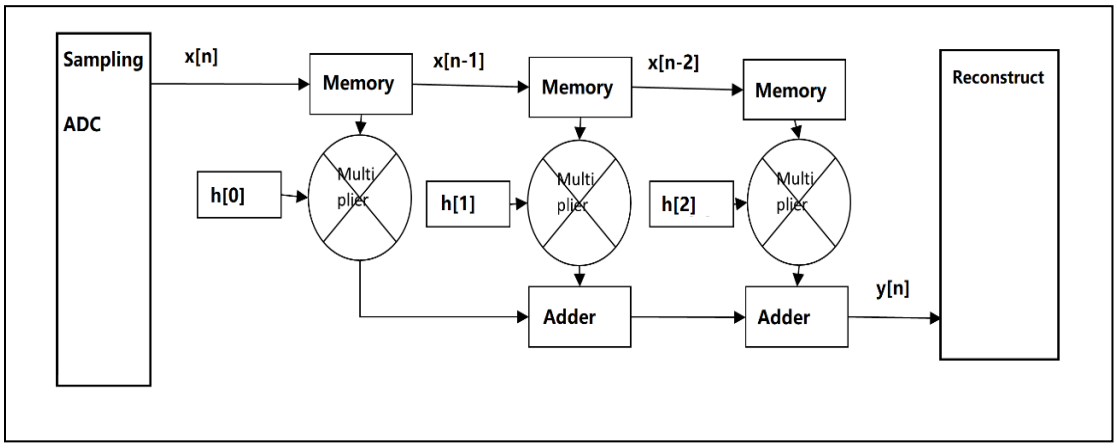


Figure 3.1 Convolution computing units of digital signal processors [42].

Conventional analog circuits have simpler structures and high real-time processing

ability but lower flexibility. Considering filter implementations, a filter can be constructed by a capacitor and a resistor but can also be constructed by running DSP algorithms on computers as described in the chapter 2. Although both systems have abilities for simple filters' implementations, conventional analog circuits have insufficient plasticity for complicated applications such as learning and recognition, 2D filters.

This thesis focusses on analog-digital complementary processing by applying digital signal processing algorithm on analog computing units to obtain advantages from both type of systems. The following sections show the details of using alternative analog ICs for algorithms.

3.2. Analog ICs in Designs

3.2.1. Digital-Analog Converters

Digital analog converters (DAC) are commonly used as input nodes for analog computing systems by converting binaries from computers to decimal voltages. After, the analog alternative computing units perform the same functions as digital computing units.

During conversions, DACs' outputs have steps that are caused by the feeding in discrete-time signal. The output voltages follow the equation 3.1 [43].

$$V_{ref} * (b_1 * 2^{-1} + b_2 * 2^{-2} + b_3 * 2^{-3} + \cdots + b_N * 2^{-N}) = V_{IN} \pm V_X \quad \textbf{(3.1)}$$

R-2R DACs are commonly used in integrated circuits since they only have R and

2R which is good for resistor matching and limits resistor sizes in IC layouts.

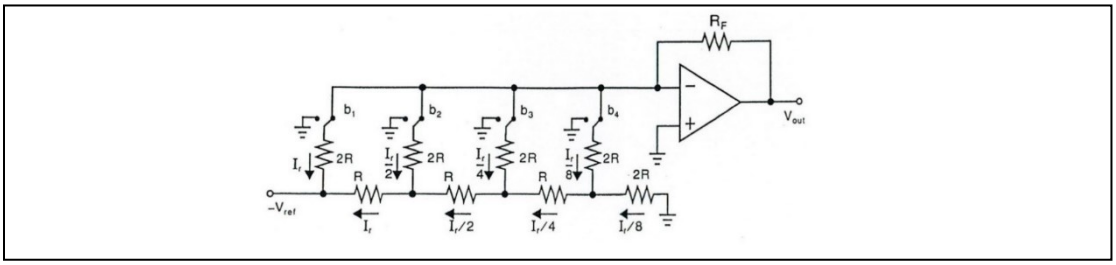


Figure 3.2 R-2R DACs structure [43].

The R-2R DACs architecture is shown in figure 3.2. The resistors are for digital-analog conversion and an op-amp for current-voltage conversion. R-2R DACs output voltages follow the equation [43].

$$V_{out} = R_F * \sum_{i=1}^{N} \left(\frac{b_i}{2^{i-1}} * \frac{V_{ref}}{2*R}\right) = V_{ref} * \left(\frac{R_F}{R}\right) * \sum_{i=1}^{N} \left(\frac{b_i}{2^i}\right) \quad \textbf{(3.2)}$$

3.2.2. Digital Potentiometers

Digital potentiometers are used as memristors' alternatives in the following designs of this thesis because they are cheaper and compatible with 3.3V and 5V GPIO outputs. They satisfy most requirements for the multiply-accumulation circuits designs on a system level. They have the same functions as conventional potentiometers which act as voltage dividers from 0V to V_{in}. Moreover, they can be controlled by digital signals which are convenient for automatic control systems.

From the perspectives of computation, analog multipliers can be both voltage dividers and amplifiers. However, voltage dividers are suitable for applying digital filters for image processing or weight matrices for artificial intelligence which have fractional

elements. Digital potentiometers satisfy this requirement and can be used for programmable analog multiply-accumulation unit designs.

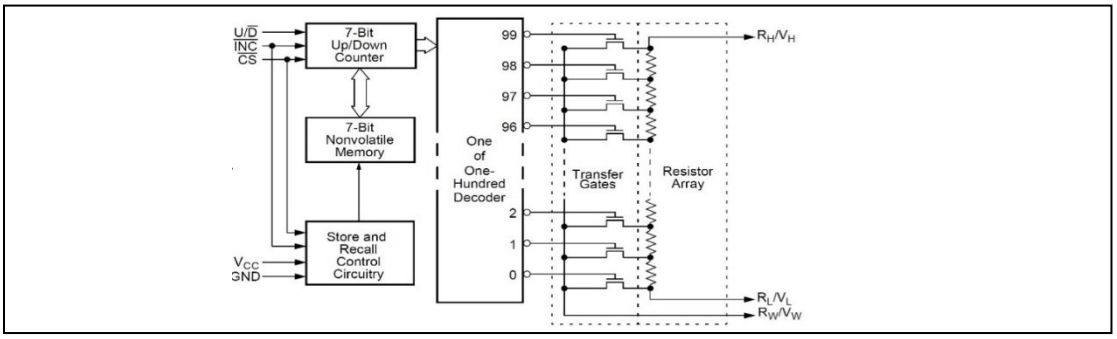


Figure 3.3 Digital potentiometer structure [44].

Figure 3.3 shows the architecture of digital potentiometers. Those used have a maximum 100kΩ adjusting range and 100 step linear voltage dividers that can output from 0V->Vin linearly. The linearity characteristic is good for algorithm implementations. Similar to conventional potentiometers, the output voltage is calculated by

$$V_{out} = V_{in} * \frac{R_{current}}{R_{total}} \quad \textbf{(3.3)}$$

Where R_{curret}/R_{total} had been matched during factory designs so that dividing ratio of the voltage divider is from 0 to 1 with accurate changing step of 0.01 [44].

The voltage divider has a ratio 0->1 that is controlled by the clock signal through INC pin and increase/ decrease one step signal through the U/D pins. The CS pin is for chip selection that enabling the chip control under low voltage level '0' and disabling and saving current dividing ratio of the voltage divider under high voltage level '1'. Also, the digital potentiometers can memorize current ratio of voltage division after their power is disconnected by the registers and the 'Store and Recall control circuitry' as shown in

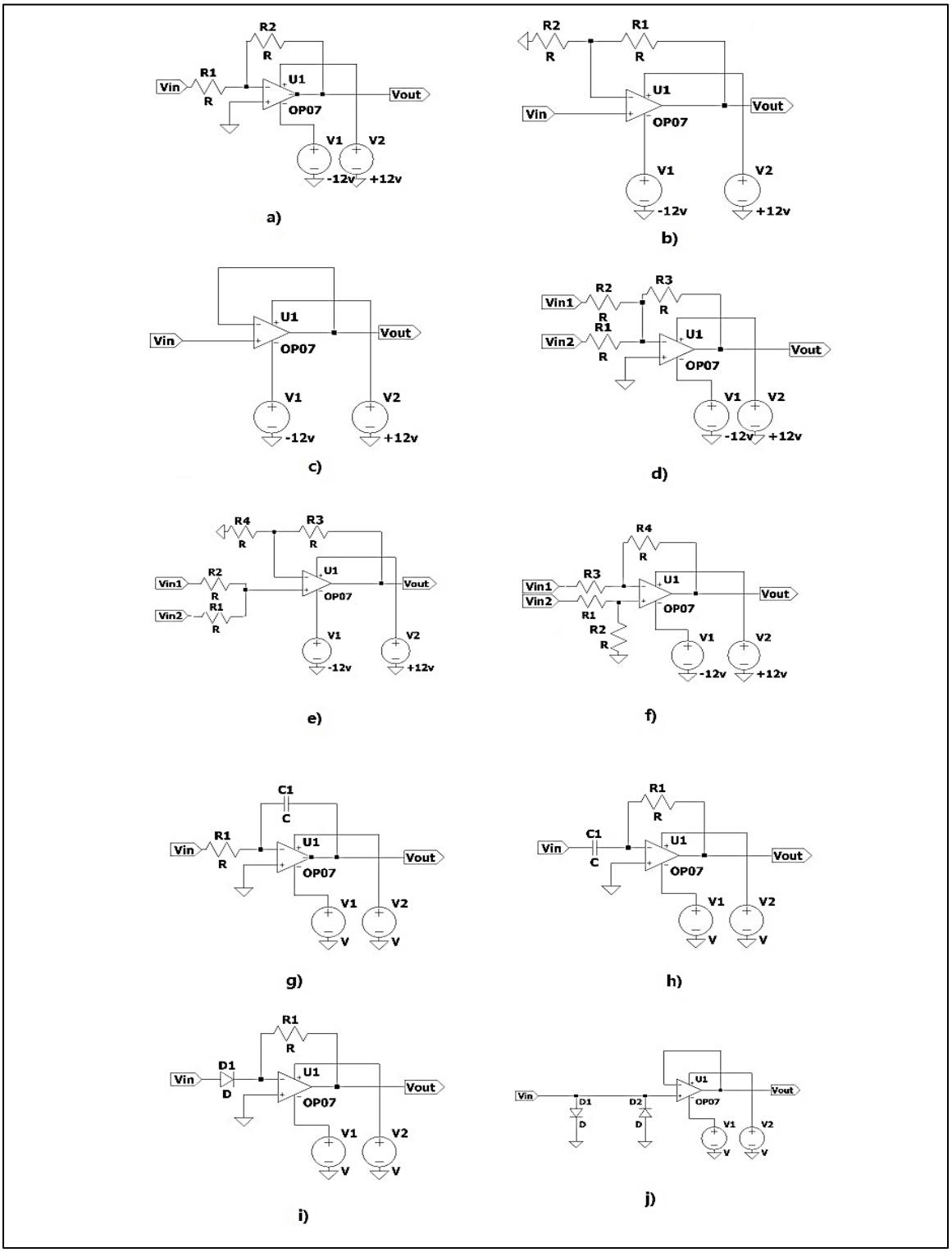
figure 3.3 [44].

3.2.3. Operational Amplifiers and Circuits

Operational amplifiers are basic components of analog signal processing hardware that can be amplifiers, analog accumulators, analog comparators, and buffers [44]. They are commonly used during analog system design. Specifically, they can implement some mathematical algorithms based on the relationship of V_{in} and V_{out}, which is the major topic of this thesis. Figure 3.4 shows some common op-amp circuits whose primary usages include amplification (figure 3.4(a) and (b)), resistance matching (figure 3.4(c)), signal addition/subtraction (figure 3.4(d), (e), and (f)), filtering (figure 3.4(g) and (h)), exponential function (figure 3.4(i)), and clamping (figure 3.4(j)). They also require complementary components for parameters adjustments such as resistors for closed loop gain control and capacitors for signal coupling.

From the perspective of computation, they implement multiplication, accumulation, subtraction, integration, differentiation, and exponentiation. Op-amp analog accumulators are the core of the multiply-accumulation systems which output the summed voltage of all inputs. They require op-amp voltage buffers for resistance matching so that the input voltages are not affected by R_{out} of the previous components and output voltages are not affected by R_{in} of the next components. Op-amp buffers are connected between computing units to keep the accuracy of the input/output voltages. Instrumentation amplifiers (figure 3.4 (k)) are based on three op-amps where two of them are for voltage buffers and one is for analog subtractions. Integrated instrumentation

amplifiers AD620 are used as buffered subtractors in this thesis for substituting for the analog subtractors (figure 3.4 (f)) so that the circuits are compacted.



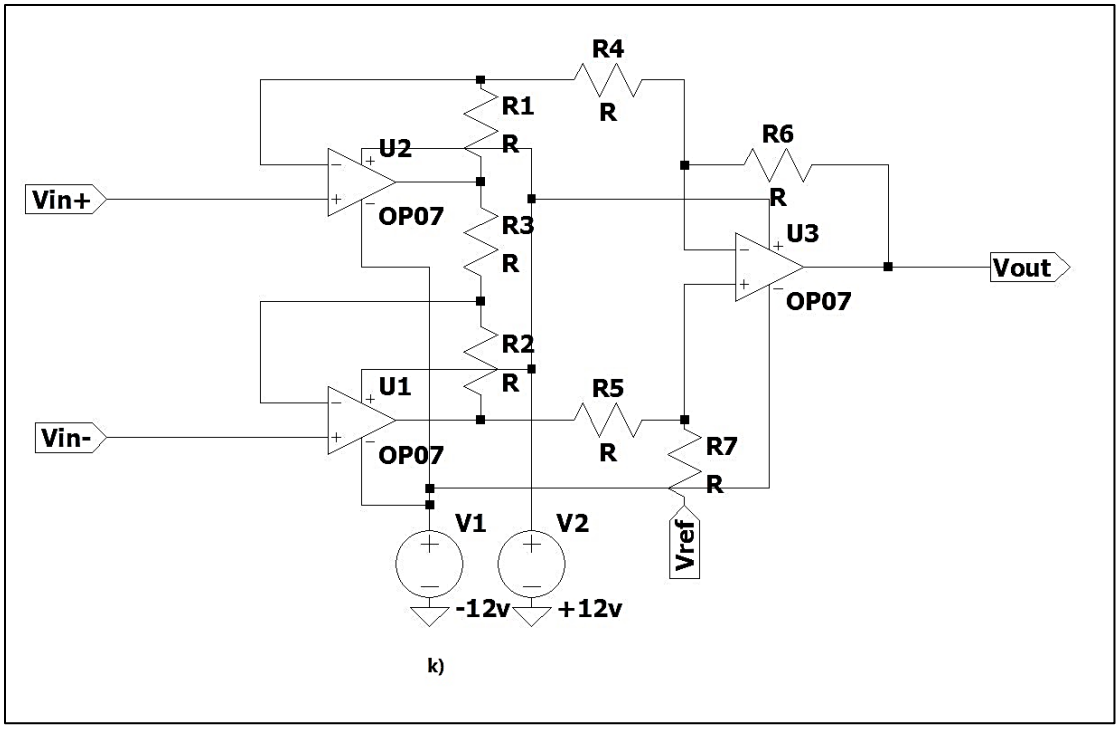


Figure 3.4 (a) Inverting amplifier (b) non-inverting amplifier (c) voltage follower (d) inverting accumulator (e) non-inverting accumulator (f) subtractor (g) integrator (h) differentiator (i) exponential circuit (j) clamping circuit (k) instrumental amplifier [44].

Table 3.1 lists functions of these circuits and algorithms for implementation [45].

Table 3.1 Summary of op-amp analog computation.

Figure	Analog circuits	Possible computation	
a	inverting amplifier	$V_{out} = -\frac{R_2}{R_1} * V_{in}$	**(3.4)**
b	non-inverting amplifier	$V_{out} = \frac{R_1+R_2}{R_2} * V_{in}$	**(3.5)**
c	Voltage followers	$V_{out} = V_{in}$	**(3.6)**
d	Inverting accumulators	$V_{out} = -(V_1 + V_2)$	**(3.7)**
e	non-inverting accumulators	$V_{out} = V_1 + V_2$	**(3.8)**
f	subtractors	$V_{out} = V_2 - V_1$	**(3.9)**

g	Integrators	$V_o = -\frac{1}{R*C} * \int_{t_1}^{t_2} V_i(t)\, dt + V_c\|_{t_1}$	**(3.10)**
h	differentiators	$V_o = -R * C * \frac{dV_i(t)}{dt}$	**(3.11)**
i	Exponential	$V_o = -RI_s e^{\frac{VI}{VT}}$	**(3.12)**
j	Clamping	$\begin{cases} V_o = \text{Vin}\ (-5.7\text{v} \le \text{Vin} \le 5.7\text{v}) \\ V_o = -5.7\ (\text{Vin} \le -5.7\text{v}) \\ V_o = 5.7(\text{Vin} >= 5.7\text{v}) \end{cases}$	**(3.13)**
k	Instrumental amplifiers	$V_o = (V_{in+} - V_{in-}) * Gain + V_{bias}$	**(3.14)**

3.3. Analog Architectures in Designs

3.3.1. Wheatstone Bridges of Resistor Matrices

Normal resistor-based voltage dividers have non-linear outputs characteristics which reduces resolution of analog computation systems. Wheatstone bridges are widely used in analog signal processing for generating linear V_{ref}/V_{out}.

3.3.1.1. Full Wheatstone Bridge

Full Wheatstone bridges are constructed by four changeable resistors that has outputs rang $-V_{ref}$->V_{ref} under single input voltage [46].

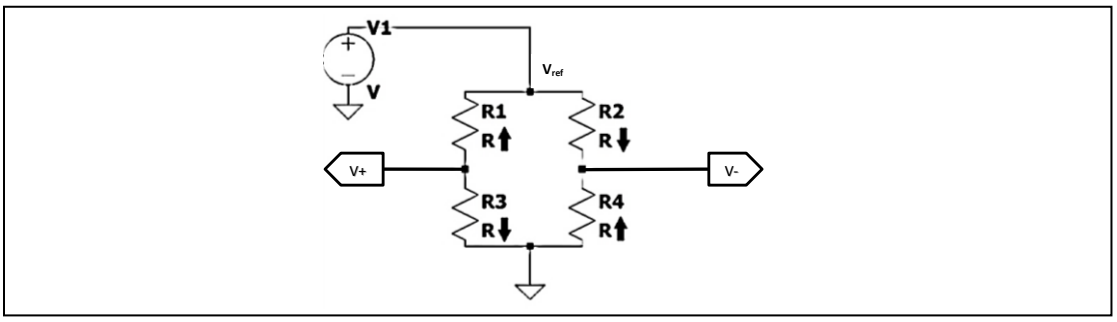


Figure 3.5 Full Wheatstone bridge: R1 & R4 increase R3 &R2 decrease or R1 & R4 decrease R3 &R2 increase [46].

The relationship of V_{out} , V_{ref} ,and R1~R4 is (eqn. 3.15).

$$V_o = V_+ - V_-$$

$$= V_{ref} * \left[\frac{R_0*(1+X)}{R_0*(1+X)+R_0*(1-X)} - \frac{R_0*(1-X)}{R_0*(1+X)+R_0*(1-X)}\right] = V_{ref} * \left[\frac{1+X}{2} - \frac{1-X}{2}\right] = V_{ref} * X \qquad \textbf{(3.15)}$$

Where V+ and V- is a pair of differential output voltages, X is a changing percentage of R_0, R_0 is the intrinsic resistance, and V_{ref} is voltage of the power supply. The circuit starts from a balanced state where R_1=R_2=R_3=R_4=R_0. Next, the differential output voltage is positive when X is above 0 (R1, R4 are increasing and R2, R3 are decreasing) and negative when X is below 0 (R1, R4 are decreasing and R2, R3 are increasing). However, as described in equation 3.15, the final output voltage only relates to V_{ref} and X linearly, but is not related to R_0.

3.3.1.2. Half Wheatstone Bridge

Half Wheatstone bridges are constructed by two changeable resistors and two fixed resistors that has output range -V_{ref}/2~V_{ref}/2 [46].

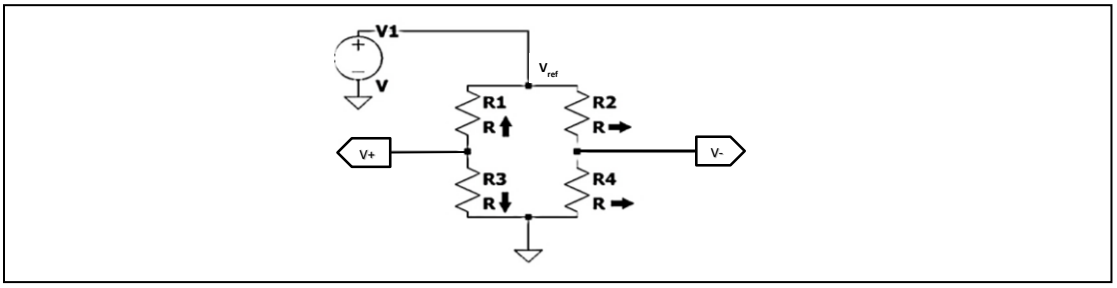


Figure 3.6 Half Wheatstone bridge: R1 increase R3 decrease or R1 decrease R3 increase and R2 & R4 fixed [46].

The relationship of V_o, V_{ref}, V+, V-, and R1~R4 is (eqn. 3.16).

$$V_o = V_+ - V_-$$

$$= V_{ref} * \left[\frac{R_0*(1+X)}{R_0*(1+X)+R_0*(1-X)} - \frac{1}{2}\right] = V_{ref} * \left[\frac{1+X}{2} - \frac{1}{2}\right] = V_{ref} * \frac{X}{2} \quad \textbf{(3.16)}$$

Where V+ and V- is a pair of differential output voltage, X is a changing percentage of R_0, R_0 is intrinsic resistance, and V_{ref} is voltage of power supply. Similar to the full Wheatstone bridge, the circuit starts from a balanced state where R_1=R_2=R_3=R_4=R_0. The differential output voltage is positive when X is above 0 (R1 is increasing and R3 is decreasing) and negative when X is below 0 (R1 is decreasing and R3 is increasing). However, the output range of half bridges are x/2 which are half of the full bridge.

3.3.2. Resistive Multiply-Accumulation Matrices

Resistive circuits have the ability for doing multiply-accumulation calculations which are common in digital signal processing fields. The resistor matrices can be constructed by resistive components such as normal resistors, potentiometers, memristors, etc. As described in section 3.1.2, the resistive voltage dividers act as multipliers for fractional numbers. Although resistors are analog devices, the processing algorithms are based on discrete-time signals. The inputs are analog sources such DACs or sensors' outputs and the programmable resistor matrices are the preset parameters of applied algorithms.

The circuit level multiply-accumulation computations are based on the well-known Kirchhoff laws which provide accumulation computation.

$$\sum_{i=1}^{n} Input\ current_i = \sum_{j=1}^{m} Output\ current_j \quad \textbf{(3.17)}$$

Also, the well-known Ohm's law provides multiplication computation.

$$I = \frac{U}{R} \quad \textbf{(3.18)}$$

Based on Kirchhoff laws and Ohm's laws we can find that

$$\sum_{i=1}^{n} \frac{Input\ voltage_i}{R_i} = \sum_{j=1}^{m} \frac{output\ voltage_j}{R_j} \quad \textbf{(3.19)}$$

For multiply-accumulation, the output is just one node. If m=1, j=1,

$$R * \sum_{i=1}^{n} \frac{Input\ voltage_i}{R_i} = output\ voltage \quad \textbf{(3.20)}$$

We can find that equation 3.21 has the same form as the multiply-accumulation of digital processing system by expanding equation 3.20.

$$Input\ voltage_1 * \frac{R}{R_1} + \cdots + Input\ voltage_n * \frac{R}{R_n} = output\ voltage \quad \textbf{(3.21)}$$

3.3.3. Analog Multiply-Accumulation Unit for Image Processing

As described in section 2.1.7, memories are required for general purpose digital signal processing systems for memorizing signals from the past. However, the memory is not necessary for off-line image processing. As discussed in section 2.4.2, image signals of image files for off-line image processing are non-causal which means that signals from past, current, and future are known. Therefore, hardware of analog algorithm

implementations can be simplified to only multiply-accumulation units for all steps of image processing which is playing to its strengths on device sizes and efficiencies.

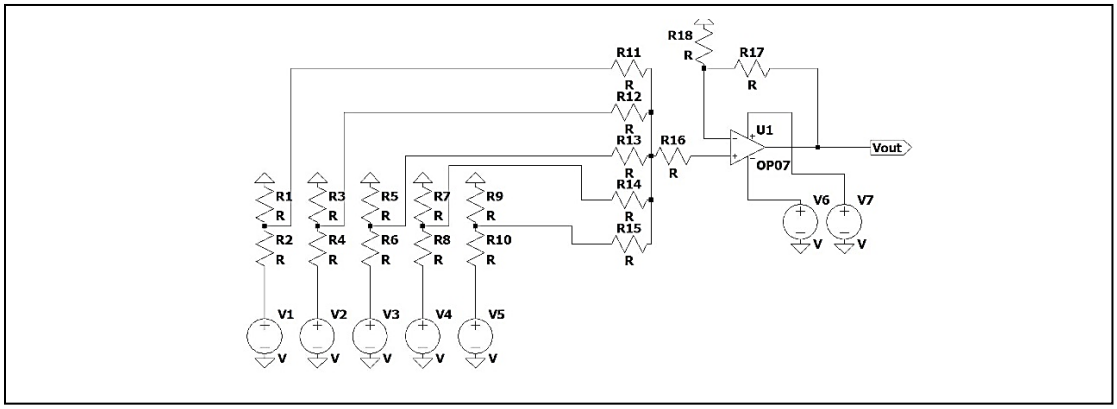


Figure 3.7 Analog multiply-accumulation unit.

Figure 3.7 shows a multiply-accumulation unit for a FIR system which has six voltage inputs as x[n] and six weight elements h[n] and an op-amp based analog accumulator. The ideal output will be equation 3.22

$$\mathrm{V_{out}} = \mathrm{V_1} * \frac{R1}{R1+R2} + \mathrm{V_2} * \frac{R3}{R3+R4} + \mathrm{V_3} * \frac{R5}{R5+R6} + \mathrm{V_4} * \frac{R7}{R7+R8} + \mathrm{V_5} * \frac{R9}{R9+R10} \qquad \textbf{(3.22)}$$

3.3.4. Artificial Neural Networks on a Resister Array

With the help of VLSI technologies, it is possible to implement artificial intelligence neural network on large-scale resistive multiply-accumulation circuits. Figure 3.8 shows a system with multiple multiply-accumulation units for artificial intelligence acceleration [4] (Schematic of figure 1.1 part b).

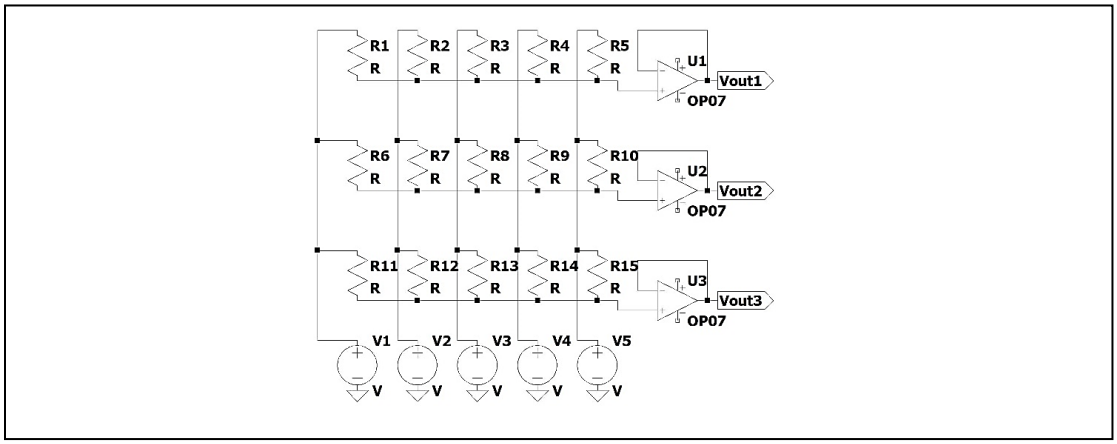


Figure 3.8 Resistor arrays implement artificial intelligence.

Chapter 4

Feedforward Structure Programmable Resistor-Based Circuits for Preset Neural Networks Implementation

4.1. Analog Circuits for DSP Algorithm Implementations

4.1.1. Introduction

Tests of this section implemented convolution operations of digital signal processing algorithms on a 1*4 digital potentiometer array. After calculation and computer simulation, we preset a selected convolutional kernel for the digital potentiometer array and recorded output images. Next, the 1*4 digital potentiometer array was expanded to 2*3*3 digital potentiometer matrices. Also, instead of the DSP algorithm, more complicated algorithms such as filters for computer vision and an artificial neural network were applied.

4.1.2. Algorithm Design

As described in Section 2.3.3, a high pass filter can extract edges of images. This section explores relationship between angular frequency ω_c of a high pass filter and the performance of edge extraction through computer simulations. As described in section 2.1.7, high pass digital filters attenuate signals with normalized angular frequencies from

0 to π/2. For computer simulations, setting the high pass digital filters with corner normalized angular frequencies π/8, π/4, 3π/4, π/2 by placing zeros at ±π/8, ±π/4, ±3π/4, ±π/2 in zero-pole plots (the points of zeros must be both positive and negative in zero-pole plots to create stable systems). Also, attenuating areas of constant values by placing zeros at ω =0 so that results of edge detection only relate to frequencies of the signals (same function as coupling capacitors of analog circuits). If we recall the equation 2.22 (eqn. 4.1),

$$H(z) = b_0 * \frac{\prod_{k=1}^{M}(1-z_k * z^{-1})}{\prod_{k=1}^{N}(1-p_k * z^{-1})} \quad \textbf{(4.1)}$$

The parameters p_k of FIR systems are always 0, and the parameters z_k are ±π/8, ±π/4, ±3π/4, ±π/2. Plotting the zero-pole plots and frequency response curves of these filters (figure 4.1).

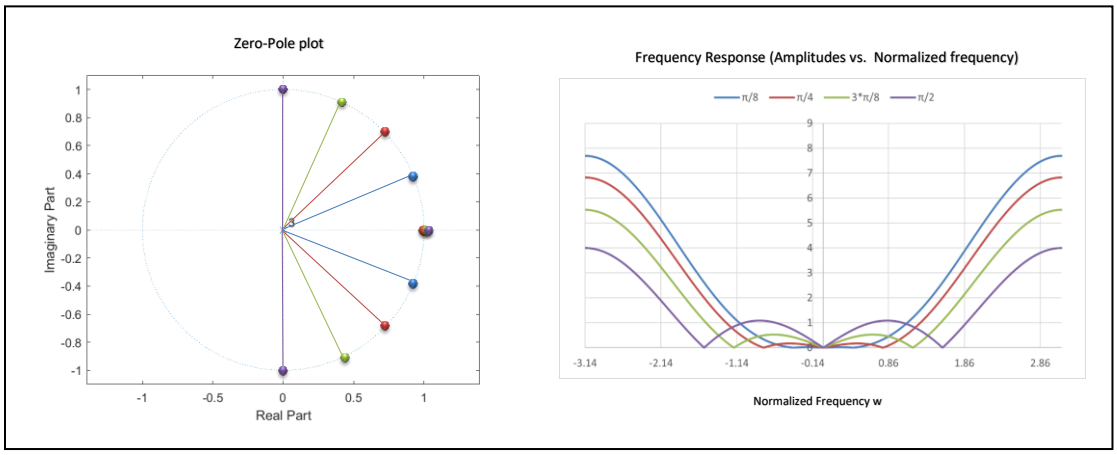


Figure 4.1 Zero-pole diagram and amplitude of frequency response of the testing corner frequencies.

Next, recalling the equation 2.20 (eqn. 4.2) and equation 2.21 (eqn. 4.3).

Converting the zero-pole equations from the form of equation 4.1 to equation 4.2. Next, converting the frequency response equation form of equation 4.2 to difference equation form of equation 4.3. Finally, we calculate the filters' parameters b_k.

$$H(z) = \frac{\sum_{k=0}^{M} b_k * z^{-k}}{1+\sum_{k=1}^{N} a_k * z^{-k}} \quad \textbf{(4.2)}$$

$$y(n) = -\sum_{k=1}^{N} a_k * y(n-k) + \sum_{k=0}^{M} b_k * x(n-k) \quad \textbf{(4.3)}$$

The parameters b_k represent impulse responses h(n) of the high pass digital filters (table 4.1). Appendix F shows details of the calculations for DSP filters.

Table 4.1 Corner frequency with filters' parameters.

	h(0)	**h(1)**	**h(2)**	**h(3)**
π/8	1	-2.8477	2.8477	-1
π/4	1	-2.4142	2.4142	-1
3π/8	1	-1.7653	1.7653	-1
π/2	1	-1	1	-1

Next, we recall the convolution operations of equation 2.10 (eqn. 4.4),

$$y(n) = \sum_{k=-\infty}^{+\infty} x(k) * h(n-k) = \sum_{k=-\infty}^{+\infty} h(k) * x(n-k) \quad \textbf{(4.4)}$$

Regarding the parameters *x(k)* as pixels of images and parameters h(n-k) as the impulse responses of table 4.1. The filters are then applied to the images.

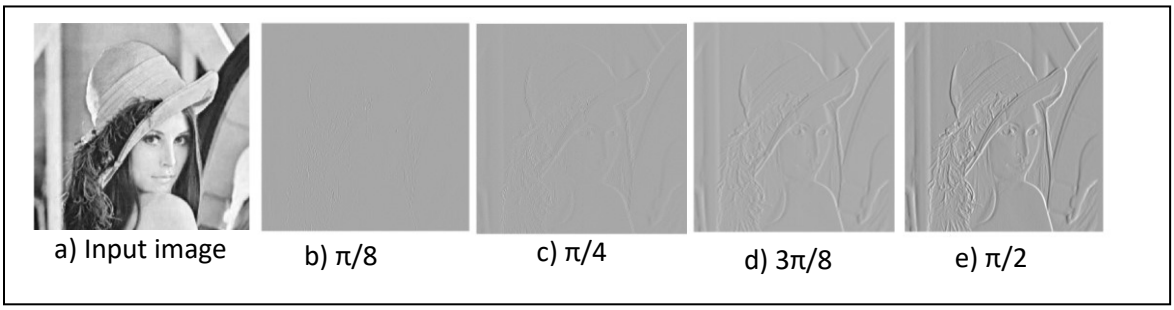


Figure 4.2 Computer simulation for corner frequency and edge detection performance a) input image b) zeros are at ω=± π /8 c) zeros are at ω=± π /4 d) zeros are at ω=±3 π /8 e) zeros are at ω=± π /2.

By observing the images of figure 4.2, when $\omega_c = \pi/2$, the output has the best performance. Therefore, the convolutional kernel [1, -1, 1, -1] was selected for hardware implementation.

Next, expanding the 1D convolutional kernel to 2D convolutional kernel so that they are sensitive for two directions. This section shows computer simulations for edge detection by a 2D filter. Creating a 4*4 filter for edge detection which expands the 1D filter from section 4.1.1 to two directions (table 4.2).

Table 4.2 A horizontal filter, a vertical filter and a 2D filter.

Horizontal filter					**Vertical filter**					**2D filter**			
1	-1	1	-1		1	-1	1	-1		1	-1	1	-1
1	-1	1	-1		1	-1	1	-1		-1	1	-1	1
1	-1	1	-1		1	-1	1	-1		1	-1	1	-1
1	-1	1	-1		1	-1	1	-1		-1	1	-1	1

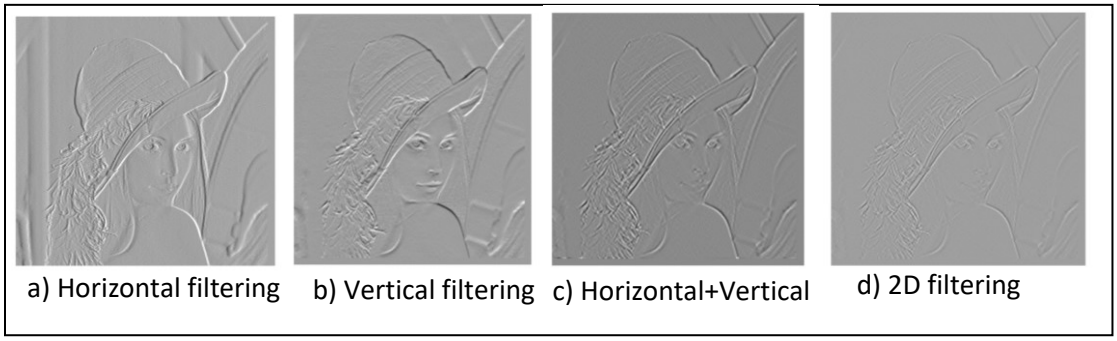


Figure 4.3 Comparing filters' performance a) applying high pass filter horizontally b) applying high pass filter vertically c) applying horizontal and vertical filter d) applying 2D filter.

As shown in figure 4.3, 1D convolutional kernels can only extract edges at one dimension but gradients of grey levels remain in the other dimension. A 2D convolutional

kernel has capability of extracting edges in both directions. The 4*4 convolutional kernel can be simplified to a 2*2 convolutional kernel for hardware realization which is $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$.

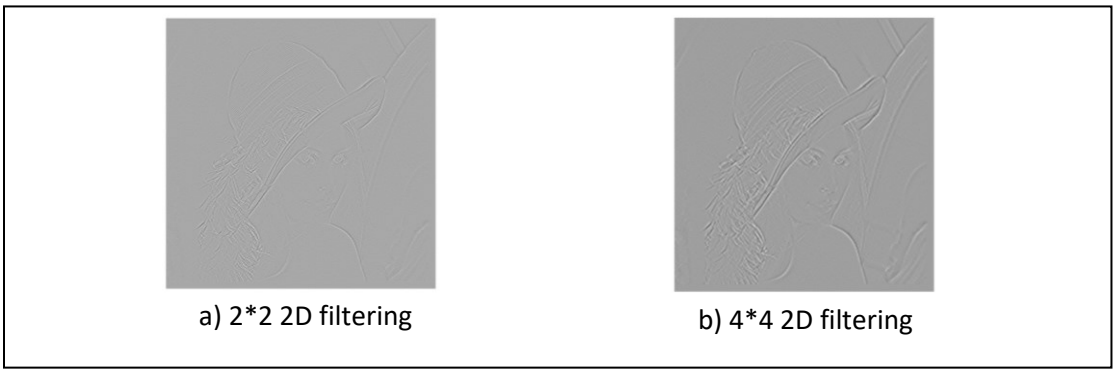


Figure 4.4 Comparing 2*2 high pass filter with 4*4 high pass filter.

We can create a block diagram which has parameters [1, -1, 1, -1] for algorithm realization. For off-line image processing, the memory blocks can be removed since all signals have been recorded in images (figure 4.5).

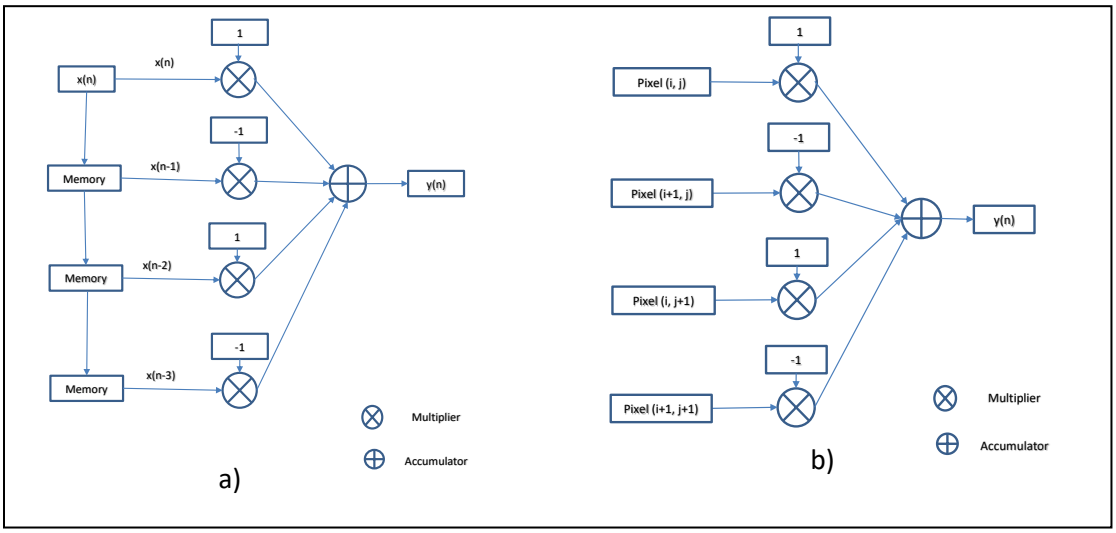


Figure 4.5 Block diagram for algorithm realization a) a digital signal processing system for real-time signals b) a digital signal processing system for off-line image processing.

4.1.3. Algorithm Hardware Implementation

As shown in figure 4.6, the implementation for the **analog 1*4 multiply-accumulation array** includes four DACs, four multipliers, and an accumulator.

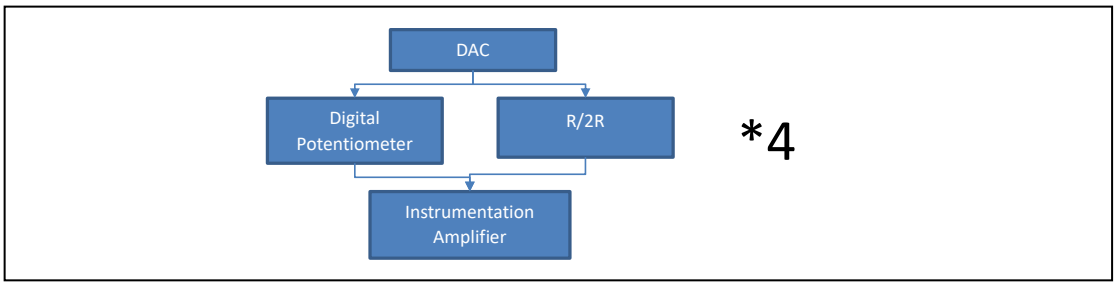


Figure 4.6 Analog multiplier implementation.

As shown in figure 4.6, an **analog multiplier** is constructed by a digital potentiometer, two fixed resistors, and an instrumentation amplifier. The digital potentiometer has voltage dividing ratio [0V/V, 1V/V] and the voltage divider R/2R has voltage dividing ratio of 0.5. The instrumentation amplifier creates a half Wheatstone bridge with output range [-0.5*V_{in}, 0.5*V_{in}] by V_{in}*(Dividing $ratio_{DP}$-0.5). Therefore, it is possible to implement both positive and negative parameters by programing the digital potentiometer. For this test, we program the digital potentiometer to 100kΩ to represent 1 and to 0Ω to represent -1. Swapping range tests of different analog multipliers are shown in appendix G – section 1.

As shown in figure 4.7, **the accumulator circuit** includes an attenuation section, an analog accumulator, and a bias voltage. The attenuation section is used for keeping the summed voltage in a range of [-10V, 10V] to avoid saturation of the op-amp. The bias voltage is applied for negative outputs so that they are recognizable by MCUs. Its actual

circuit implementation is shown in Appendix D, figure D.4.

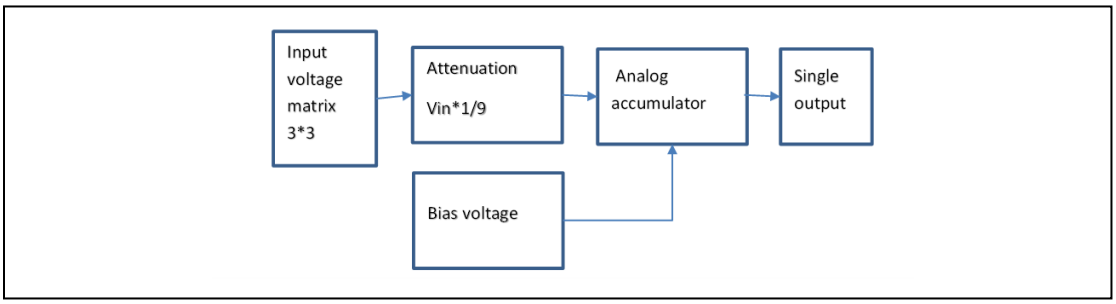


Figure 4.7 Diagram of the analog accumulator.

Pooling Units: as shown in figure 4.8, 'Pooling' operation of AI algorithm can down-sample the input data by taking the highest value among a 2*2 inputs array. A 2*2 pooling circuit is based on four forward bias diodes that the highest output branch attenuates the other branches by reverse biasing the diodes (its schematic is in figure Appendix C.4).

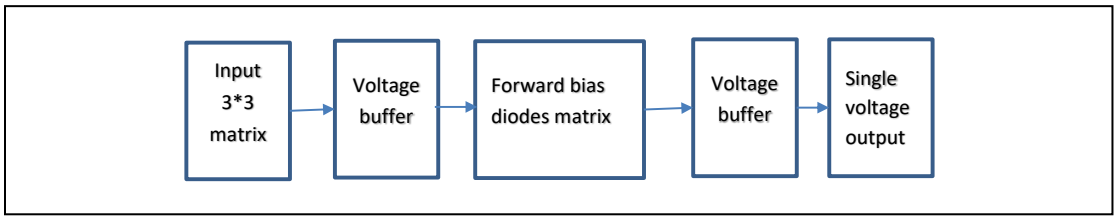


Figure 4.8 Block diagram of the pooling circuit.

4.1.4. Test Bench and Test Method

The test is set up by connecting the test bench as in table 4.3 and the system block diagram of figure 4.9. As shown in figure 4.9, MATLAB controls the DAC matrix through

the DAC controller that is a STM32 MCU. For testing the convolution operations, MATLAB loads each 1*4 array from the input images to the DAC controller. Next, the signals are switched to the path of the analog accumulator. Finally, the multiply-accumulation signal is captured by the Arduino Mega and recorded by MATLAB. For testing the pooling operations, the digital potentiometers are programmed to 1V/V and the signals are switched to the 4 diodes. Finally, the signal of pooling operation is captured by the Arduino Mega and recorded by MATLAB. Its actual circuit implementation is shown in Appendix C figure C.13.

Table 4.3 Pre-setting convolution kernel test bench.

Inputs	Measurement Device	Circuit being tested	Analysis Software
DAC array	Arduino Mega	Potentiometer Wheatstone bridges	MATLAB

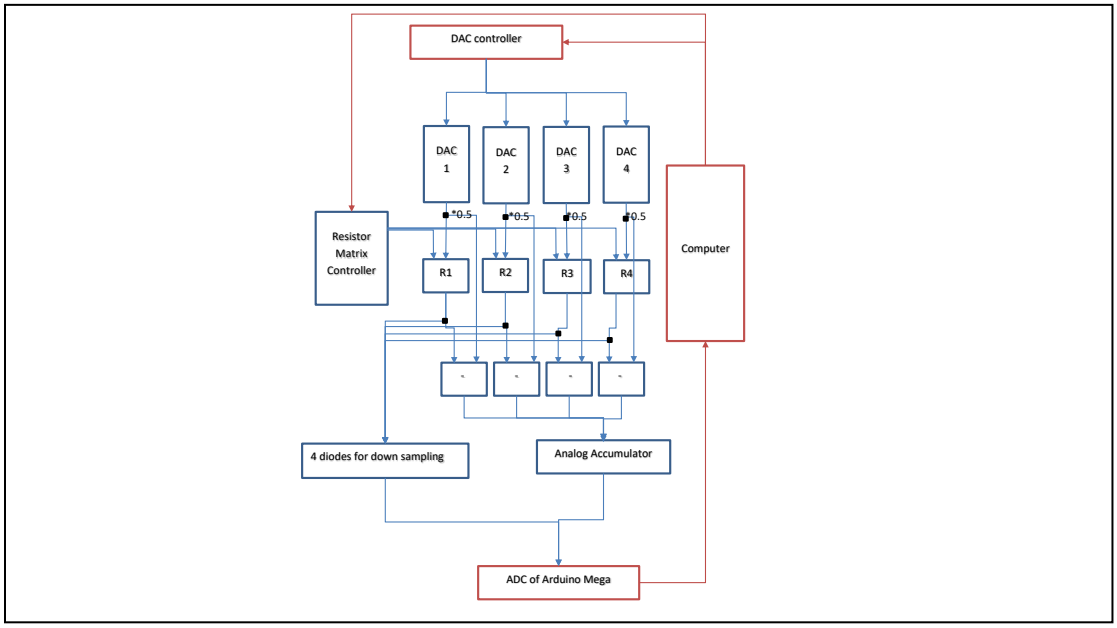


Figure 4.9 Block Diagram of the test bench for testing analog multiply-accumulation circuits for convolution operations (Red blocks are testing devices).

4.1.5. Test Results

Figure 4.10 shows results of implementing 1D DSP high pass filter on the 1*4 digital potentiometer array. The results show image processing capability of the digital potentiometer array. Edges of the image are detected by the high pass filter. Also, figure 4.10 d), h), and i) show the pooling operation capability. Figure 4.11 shows a result of implementing the 2*2 convolutional kernel on the 1*4 digital potentiometer array. The 1D DSP filter can be converted to a 2D DSP filter by folding it.

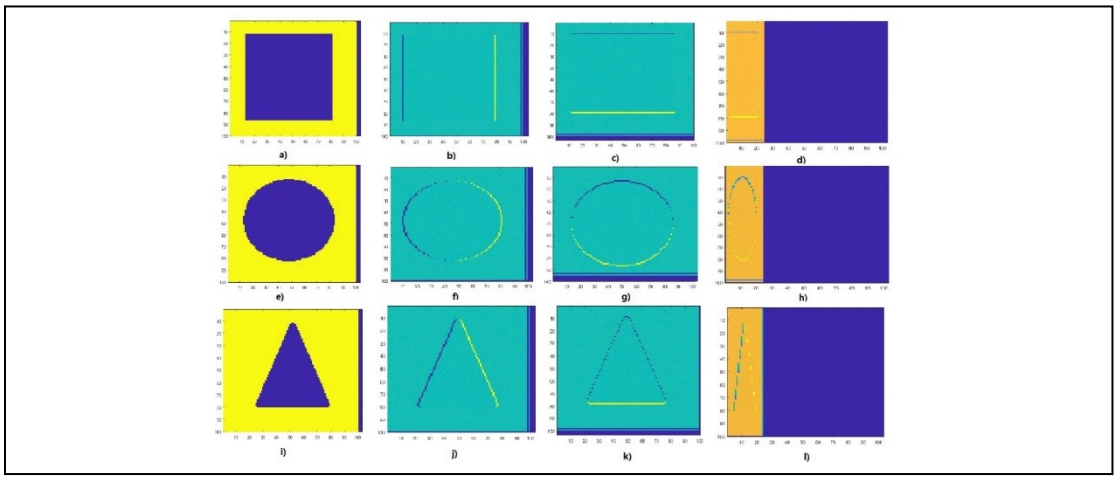


Figure 4.10 1-dimentional edge detection result a) input square b) doing horizontal 1*4 edge detection by 1*4 resistors array c) doing vertical 1*4 edge detection by 1*4 resistors array d) doing 1*4 pooling by 1*4 diodes array e) input circle f) doing horizontal 1*4 edge detection by 1*4 resistors array g) doing vertical 1*4 edge detection by 1*4 resistors array h) doing 1*4 pooling by 1*4 diodes array i) input circle j) input triangle k) doing horizontal 1*4 edge detection by 1*4 resistors array l) doing 1*4 pooling by 1*4 diodes array.

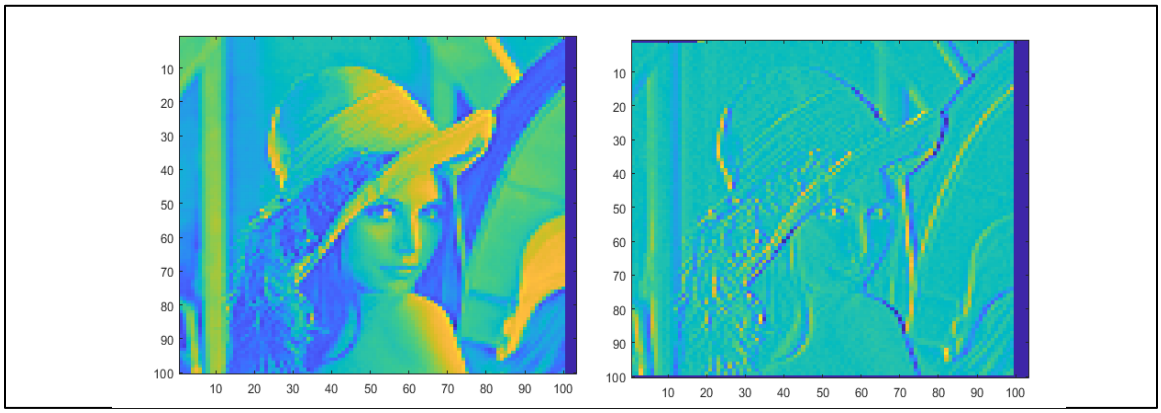


Figure 4.11 Doing 2-dimension convolution.

4.2. 2*3*3 Analog Multiply-Accumulation Computing System

4.2.1. Introduction

The results from section 4.1 confirms that DSP algorithms can be implemented on a programmable resistor array. This section shows a matrix computing system that expands the 1*4 digital potentiometer array to a 2*3*3 digital potentiometer matrix for applications to computer vision and artificial intelligence.

4.2.2. Algorithm Design

The analog multiply-accumulation computing systems can be used under practical circumstances. Focuses of this thesis are computer vision and artificial intelligence.

For implementing **computer vision algorithms** on this system, we select two sharpening filters which are Laplacian operator $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ and Sobel $\begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix}$ [47] and two smoothing filters which are Mean $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ and Median $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ [48]. The matrices of the filters require conversion for programming the digital potentiometers. Their calculations are given in Appendix H. Table 4.4 and table 4.5 show conversions from the filters to voltage divider matrices.

Table 4.4 Applied sharppening filters.

	Laplacian operator				**Sobel**		
Filters	-1	-1	-1		-1	2	-1
	-1	8	-1		0	0	0
	-1	-1	-1		-1	2	-1
Corresponding voltage divider ratios	-0.05	-0.05	-0.05		-0.25	0.5	-0.25
	-0.05	0.4	-0.05		0	0	0
	-0.05	-0.05	-0.05		-0.25	0.5	-0.25

Table 4.5 Applied smoothing filters.

	Mean				**Median**		
Filters	1	1	1		1	2	1
	1	1	1		2	4	2
	1	1	1		1	2	1
Corresponding voltage divider ratios	0.5	0.5	0.5		0.1	0.25	0.1
	0.5	0.5	0.5		0.25	0.5	0.25
	0.5	0.5	0.5		0.1	0.25	0.1

For implementing a **convolutional neural network**, algorithm tests include a convolutional kernel training algorithm and a convolutional neural network model design. In terms of **convolutional kernel training algorithm**, we can apply the knowledges of correlation operations (algorithm simulation is in section 1.1 of appendix E) and gradient descent algorithm [49] (algorithm simulation is in section 1.5 of appendix E). In sections 1.2, 1.3, 1.4 of appendix E, some simulations are presented to illustrate the algorithms used for pattern recognition. We designed a simplified 3*3 binary convolutional kernel

training algorithm. Assuming an input 3*3 input matrix **X** and a 3*3 weight matrix **W** (eqn. 4.5).

$$\begin{cases} \mathrm{y} = \sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{x}_{i,j} * \mathrm{w}_{i,j} \\ \sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{w}_{i,j} = 1 \end{cases} \quad \textbf{(4.5)}$$

Instead of exploring decreasing gradients at high-level dimensional space, we will explore using the matrix **W** for maximizing the final output y. During the exploration, the weight matrix starts from a random weight matrix (eqn. 4.6).

$$\mathrm{w}_{m,n} = \frac{Temp_{m,n}}{\sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{Temp}_{i,j}} \qquad ,1 \le m \le 3, 1 \le n \le 3 \quad \textbf{(4.6)}$$

Next, we explore y by moving variable $\mathrm{Temp}_{1,1}$ a step and refreshing the weight matrix **W** (eqn. 4.7),

$$\begin{cases} w_{1,1} = \frac{Temp_{1,1}+step}{(\sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{Temp}_{i,j})+step} & ,m = 1, n = 1 \\ w_{m,n} = \frac{Temp_{m,n}}{(\sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{Temp}_{i,j})+step} & ,others \end{cases} \quad \textbf{(4.7)}$$

Next, we plug the new weight matrix **W** into equation 4.5 and recalculate y. Changing names for the past output y and the recalculated output y'. We can adjust the next exploring step based on (eqn. 4.8),

$$step = \begin{cases} step & ,y' > y \\ -step & ,y' < y \end{cases} \quad \textbf{(4.8)}$$

Finally, we finish exploring $\mathrm{w}_{1,1}$ after repeating 100 times and moving to $\mathrm{w}_{1,2}$ and applying the same processes. By using this algorithm, we find that y is increasing when a step is assigned to '1' but y is decreasing if a step is assigned to '0'. Figure 4.12 shows a flow chart diagram of this algorithm.

In terms of **CNN model design**, we designed a simple convolutional neural network with a pooling layer, a convolutional layer, a fully connected layer, and a non-linear module for decision making (simulations and details are given in section 1.7 of appendix E). Figure 4.13 shows the CNN model, during training stage, the hand-writing images are down sampled by the pooling layer and learnt by the training algorithm. Next, the learnt convolutional kernels are recorded by MATLAB as the convolutional layer. During image recognition stage, each of the hand-writing images is fed into the neural network through the pooling layer, the convolutional layer, the fully connected layer, and the non-linear decision-making module.

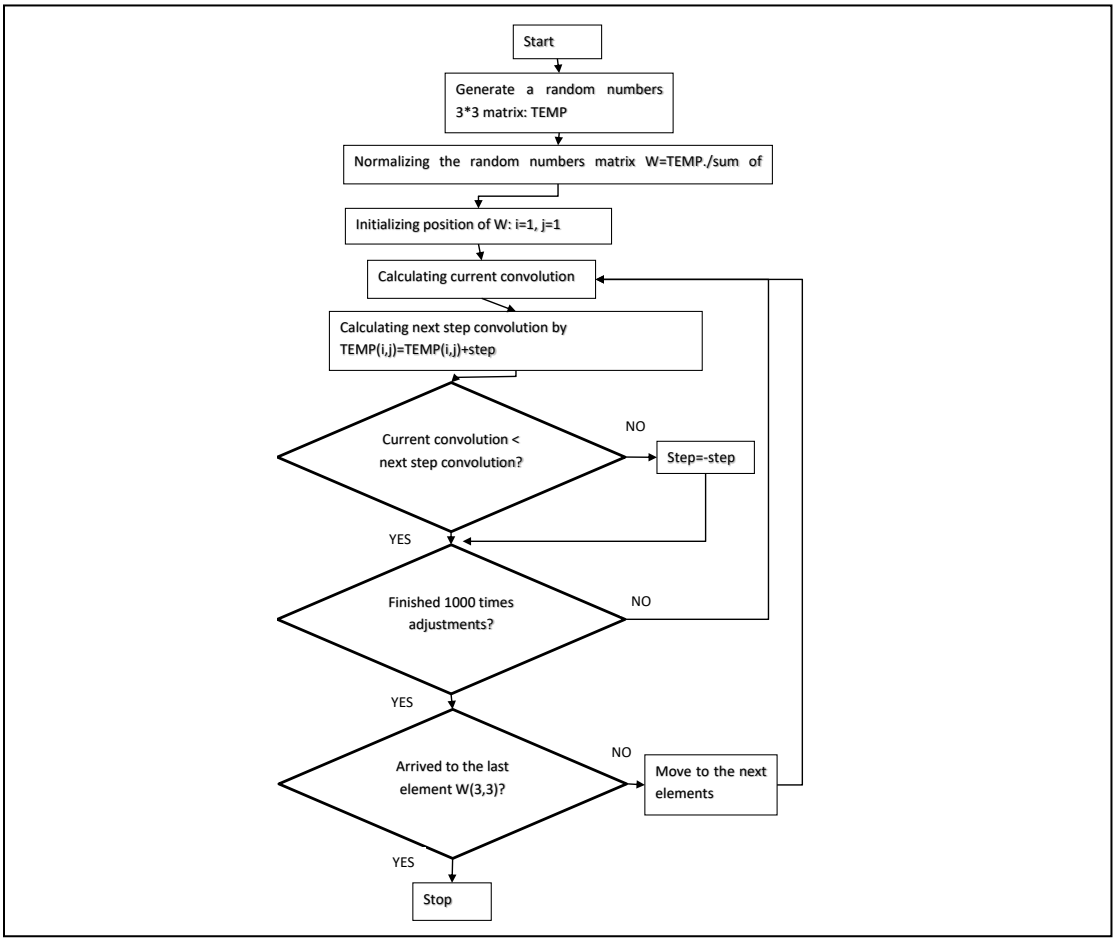


Figure 4.12 Patterns' self-learning algorithm.

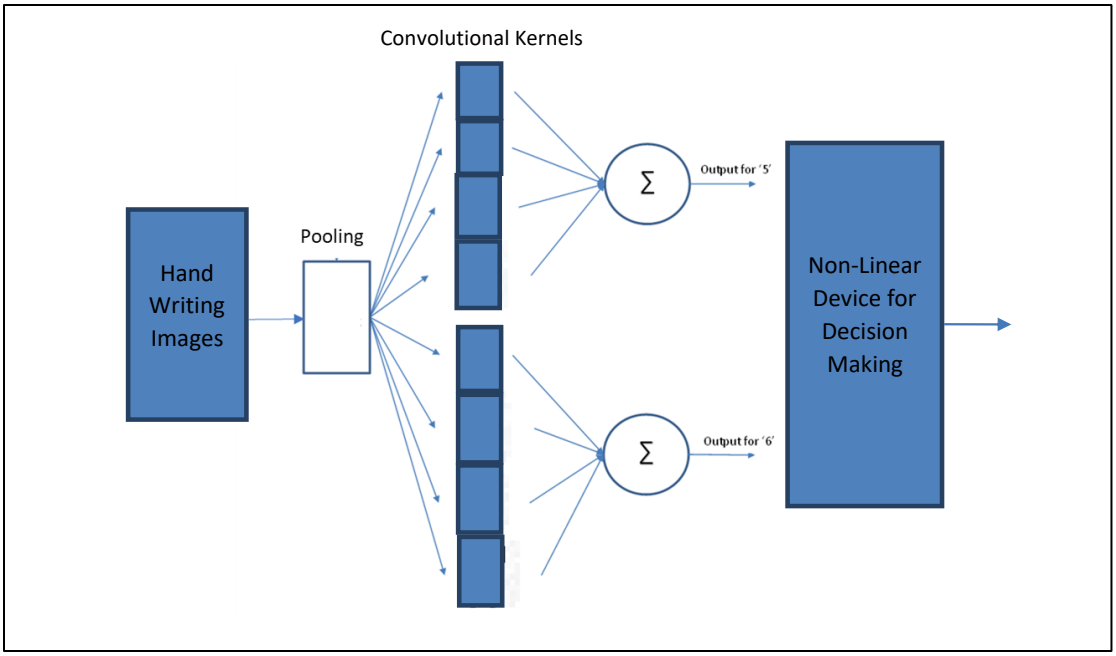


Figure 4.13 A simple convolutional neural network.

4.2.3. Algorithm Hardware Implementation

4.2.3.1. System Block Diagram

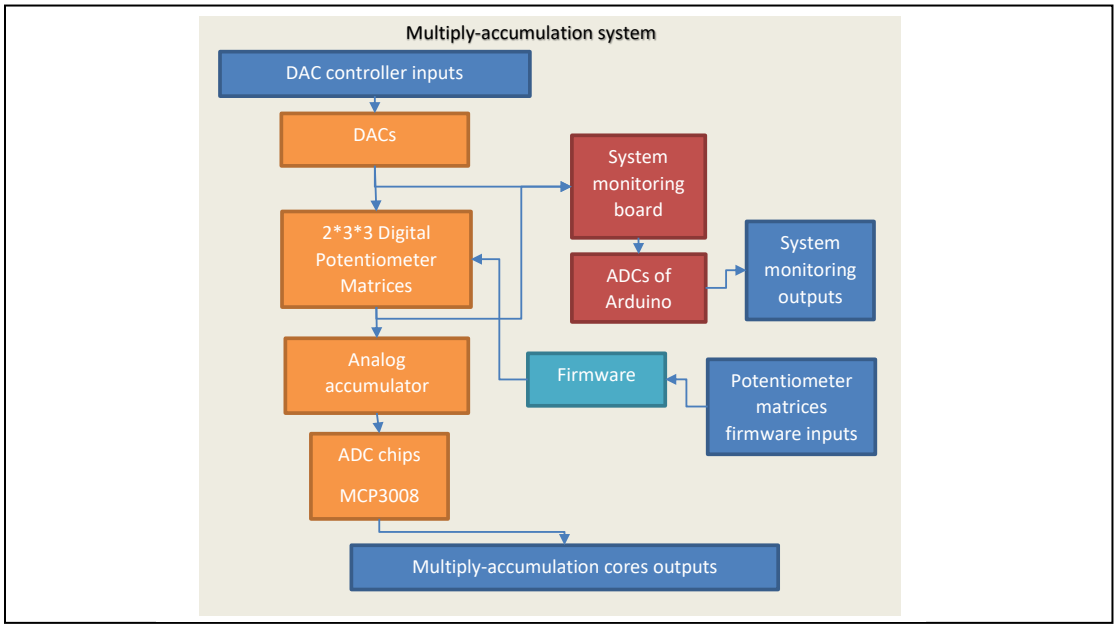


Figure 4.14 Schematic of a full multiply-accumulation system a) detail of a multiply-accumulation system b) two multiply-accumulation systems with controlling and data collecting devices.

Figure 4.14 shows a block diagram for the system implementation. The blocks, which are DACs, 2*3*3 digital potentiometer matrices, and analog accumulator, are designed by the same principle as the 1*4 digital potentiometer array. Outputs of multiply-accumulation signals are captured by the ADC chip. There is a system monitoring module for reading the input matrix from the DACs and output matrix from the 2*3*3 digital potentiometer matrices. There is firmware for managing the 2*3*3 digital potentiometer matrices. Its actual circuit implementation is shown in figure D.10 of Appendix D.

4.2.3.2. Block Designs

We designed analog devices for the system block diagram of figure 4.14. The analog devices include a DAC matrix, a 2*3*3 digital potentiometer matrices, an analog accumulator, a system monitoring board, and the firmware. There are also required supplementary devices such as power modules, communication modules, etc.

DAC Matrix Controller: The DAC array and ADC array translate signals between analog and digital so that they help the computer assigning tasks and capturing results. The DAC matrix controller includes a STM32F407 development board and an MCP23017 I2C GPIO expander board for address expansion (Block diagram of figure 4.15). It communicates with a computer through an UART port and generates data and address signals for the DACs of the multiply-accumulation cores. Its actual circuit implementation is shown in figure Appendix D.1 of Appendix D.

2*3*3 analog multiply-accumulation core: First, we need to design one section of 1*3 Resistor array. By Kirchhoff laws, which is mentioned in section 3.3.1, applying multiply-accumulation algorithm on pure resistor matrices is possible. We use digital potentiometers that have outputs V_{DAC}*$R_{current}$/100kΩ for representing weight 0->1 and two fixed resistors for representing 0.5. Instrumentation amplifiers act as subtractors to change the weights from [0V/V,1V/V] to [-0.5V/V,0.5V/V]. Figure 4.16 shows the schematic, each unit has 3 groups of DACs, voltage dividers, and instrumentation amplifiers (its schematic is in figure Appendix C.1 and its actual circuit implementation is in figure Appendix D.2). An analog multiply-accumulation core requires 6 units.

Next, we assemble an analog multiply-accumulation core by 6 units. The DACs are controlled by the DAC controller and the 2*3*3 digital potentiometers are controlled by the firmware (figure 4.17) (its schematic is in figure Appendix C.2 and actual circuit implementation is in figure Appendix D.3). Finally, an analog accumulator for 9 inputs was designed. As shown in figure 4.18, the accumulator circuit includes an attenuation section, an analog accumulator, and a bias voltage. The attenuation section is used for keeping the summed voltage in a range of [-10V, 10V] to avoid saturation of the op-amp. The bias voltage is applied for negative outputs so that they are recognizable by MCUs (its schematic is in figure Appendix C.3 and its actual circuit implementation is in figure Appendix D.4).

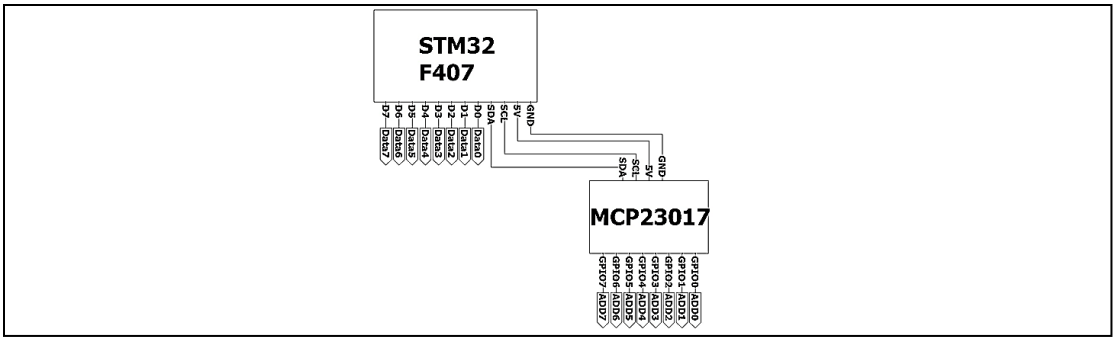


Figure 4.15 Block diagram of the DAC matrix controller circuit.

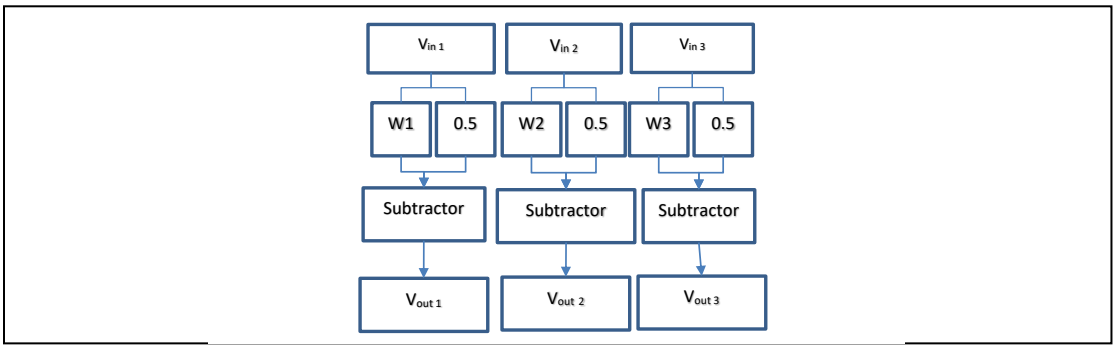


Figure 4.16 Block diagram of the analog multiply-accumulation circuit.

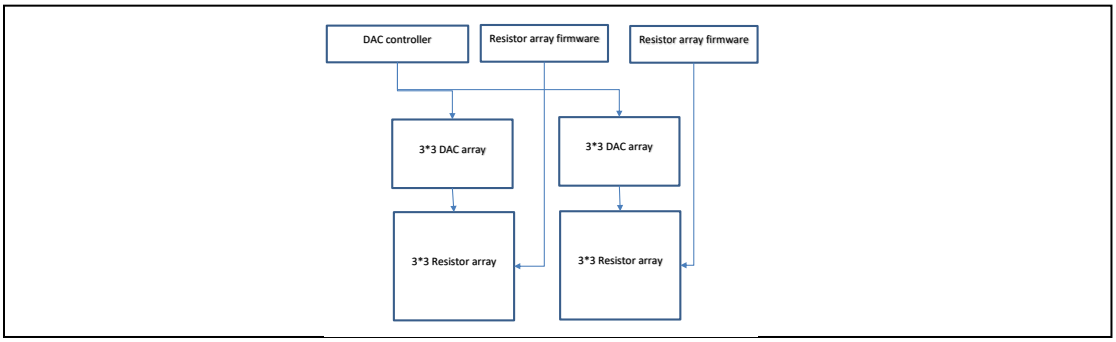


Figure 4.17 Block diagram of the 2*3*3 convolution core with controllers.

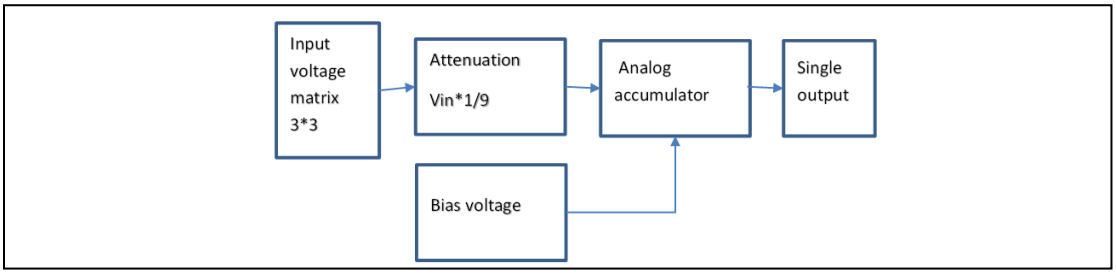


Figure 4.18 Diagram of the analog accumulator.

Signal Monitoring Circuit: we added a module for signal selection and adjustments. The signal monitoring module includes a signal switching circuit and an Arduino Mega for communications. The signal switching circuit is a matrix of multiplexers and amplifiers. As shown in figure 4.19, this module is used for monitoring input and output conditions of the analog multiply-accumulation cores to allow getting feedback and debugging the circuit. The circuit has three functions which are signal selection, signal reverse, and signal amplify so that signals are readable for the Arduino Mega(its schematic is in figure Appendix C.5 and its actual circuit implementation is in figure Appendix D.5).

Power Supply Module: the chips of the system need +12V, +5V, -12V, and -5V. The LM7805 and LM7905 linear regulator ICs are used to convert +12V to +5V and -12V to -5V. Figure 4.20 shows a schematic of the power supply module (Its actual circuit implementation is in figure Appendix D.6).

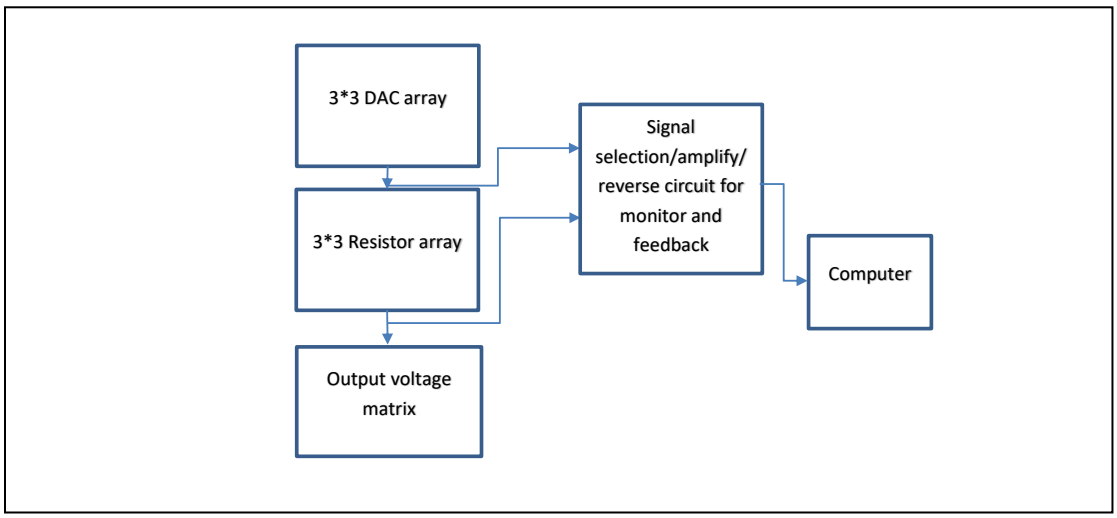


Figure 4.19 Block diagram of the signal monitoring circuit.

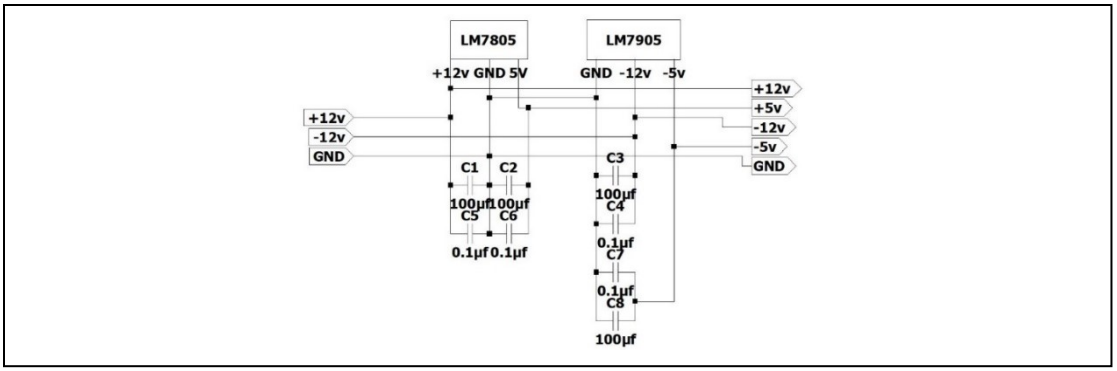


Figure 4.20 Schematic of the power supply board.

MCU Firmware for Digital Potentiometer Matrices: figure 4.21 shows an MCU-based digital potentiometer controller. The MCU is used for initializing the digital potentiometers, recording current states of them, and communicating with MATLAB through UART. We have used Arduino Nano development boards as the MCU controllers since they have sufficient speed and 5V compatible GPIOs. Figure 4.21 shows the schematic for the firmware control of the potentiometer matrices. The essence of the MCU algorithm implementation is still based on software which requires digital computing units for sequential executions (its schematic is in figure Appendix C.6).

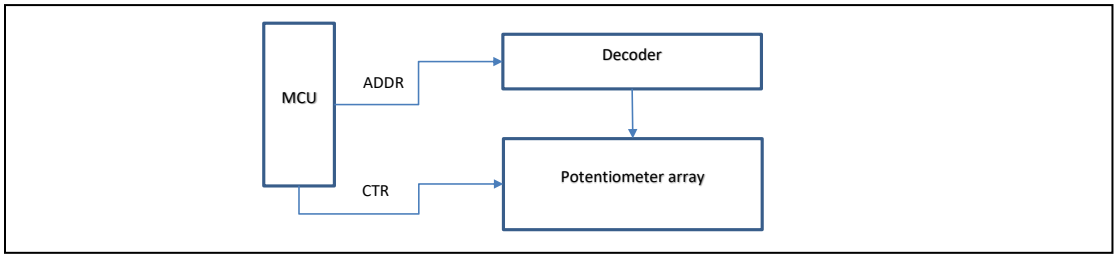


Figure 4.21 Block diagram of a MCU resistor array controller.

4.2.4. Test Bench and Test Methods

Before applying the algorithms, some basic tests for the system are presented in appendix G that include binary patterns memorizing, multiple gray-levels patterns memorizing, pattern learning, and pattern recognition.

In terms of the **computer vision algorithms implementation**, Figure 4.22 shows the procedures of the test, MATLAB programs the digital potentiometer matrices by referring to the voltage dividers ratios first. Next, we check the digital potentiometer matrices through the monitoring module. Finally, MATLAB loads each 3*3 matrix of the input image to the DAC controller. During the tests, we apply the sharpening filters for input images directly, but we add Gaussian noise for testing the smoothing filters.

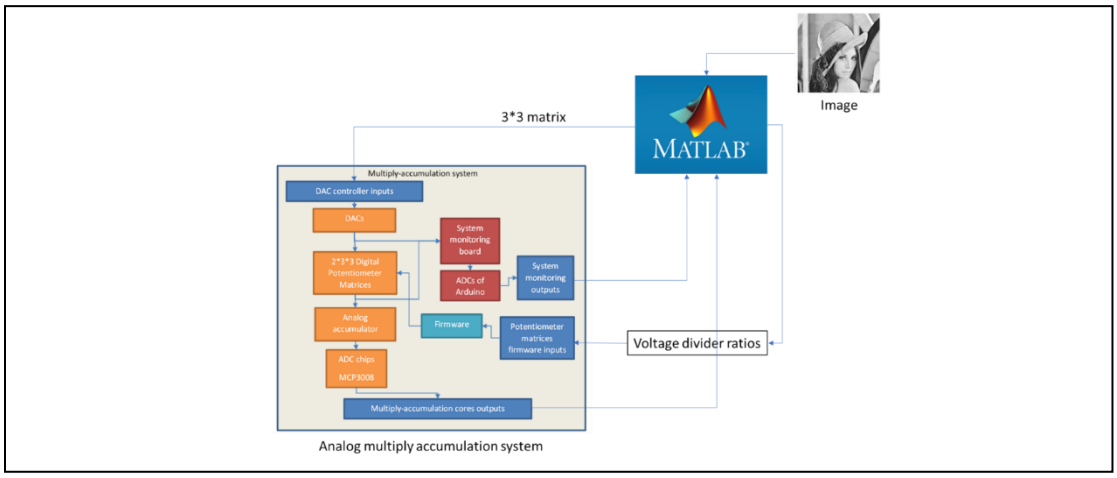


Figure 4.22 Test bench for the computer vision algorithms implementation.

In terms of tests for **artificial intelligence implementation**, the first step is to apply 4*4 pooling operation to down sample the images of handwriting numbers since they have much larger size than the 3*3 analog multiply-accumulation cores. Next, we apply

the convolutional kernels training algorithm and saving the results into MATLAB. We then feed in each handwriting image for the recognition tests. MATLAB is programed to control the multiply-accumulation system to load the learnt patterns partially and execute convolution operations multiple times. Next, the 8 convolution results are fed into the fully-connected neural network and passed through a non-linear module for making final decisions. Figure 4.23 shows the hand-writing images for tests. Figure 4.24 illustrates the 'partial convolution' for small size matrices (eqn. 4.9). Figure 4.25 shows a CNN model for the following tests. The 8 convolutional kernels are learnt by MATLAB for recognizing '5' and '6'.

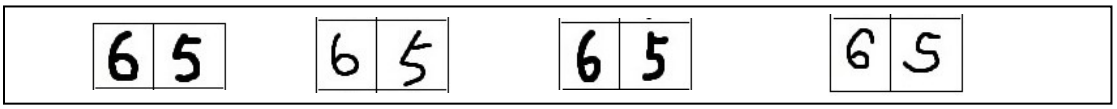


Figure 4.23 Handwriting numbers for training.

$$\mathrm{x(i,j,k)}\ \mathrm{w}(i,j,k) = \sum_{k=1}^{12}\sum_{\mathrm{j}=1}^{3}\sum_{\mathrm{i}=1}^{3}\mathrm{x}(i,j,k) * \mathrm{w}(i,j,k) \qquad \textbf{(4.9)}$$

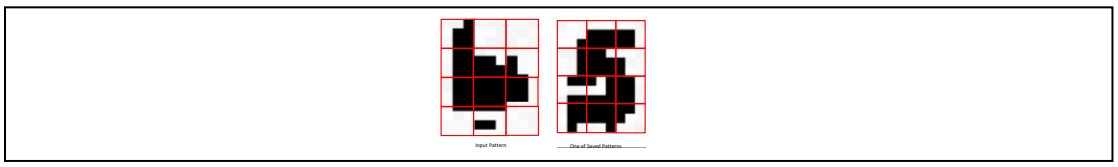


Figure 4.24 Multiple-time convolution for the 3*3 digital potentiometer matrices.

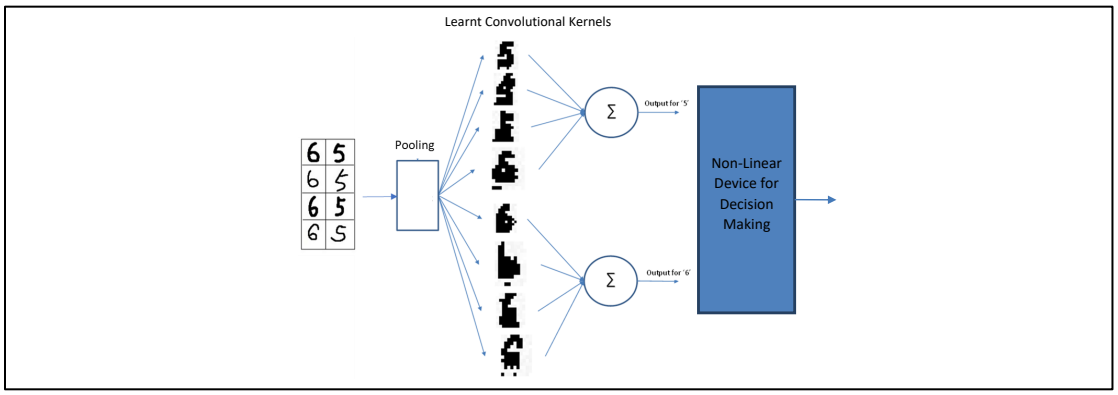


Figure 4.25 Test for convolutional neural network implementation.

4.2.5. Test Results

Figure 4.26 shows test results of the Laplacian operator and Sobel filters. We also execute the computer simulation by MATLAB for comparison. Figure 4.26 a) is result of Laplacian operator and Figure 4.26 b) is computer simulation by MATLAB. Figure 4.26 c) is result of Sobel and Figure 4.26 d) is computer simulation by MATLAB.

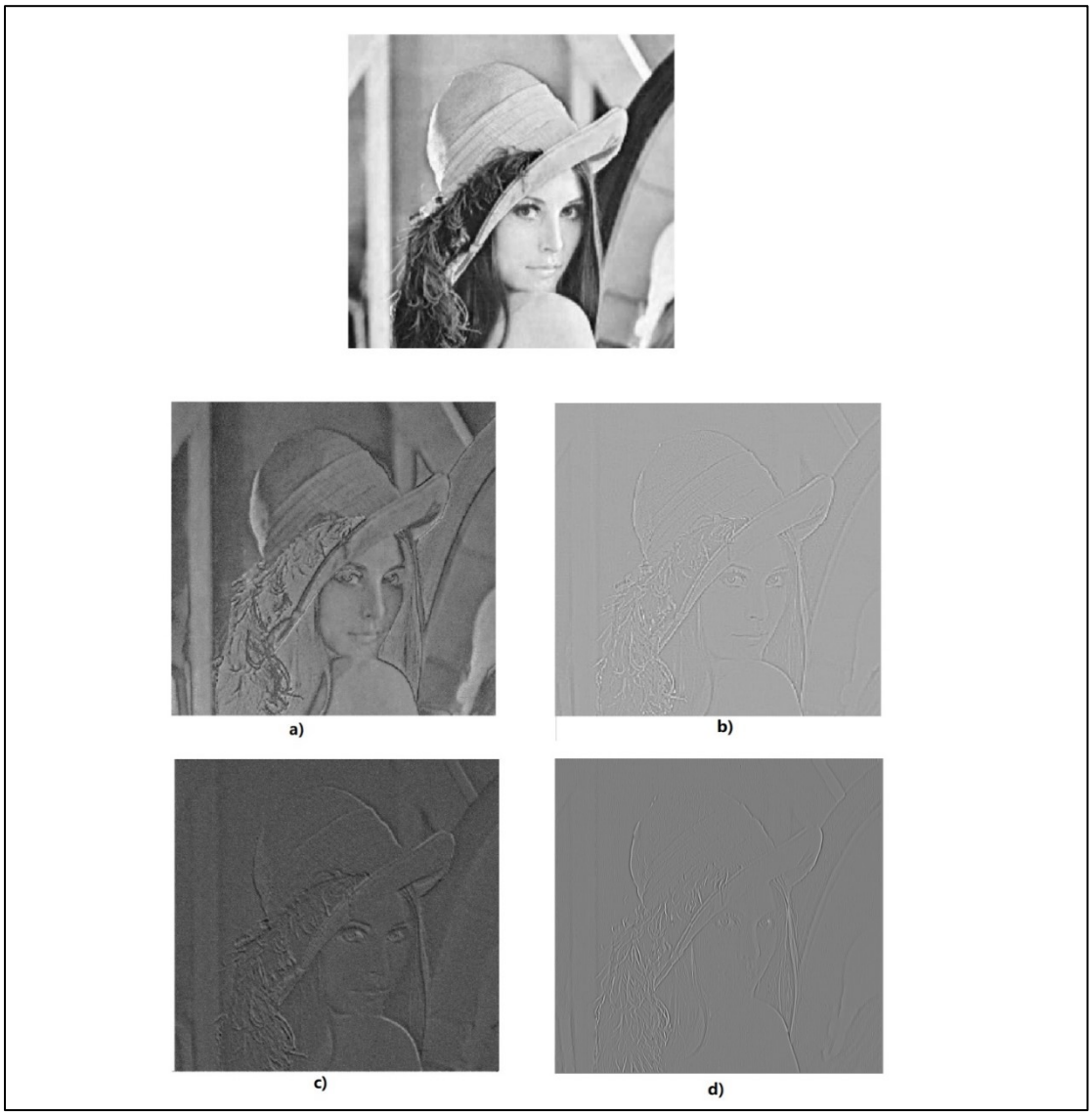


Figure 4.26 Image processing test a) Applying Laplacian Operator filter by the digital potentiometer matrices b) Applying Laplacian Operator filter by MATLAB c) Applying Sobel filter by the digital potentiometer matrices d) Applying Sobel filter by MATLAB.

Figure 4.27 shows test result of Mean and Median filters implementation. As in the description of test method, the input image is added with Gaussian noise. Figure 4.27 e), f), g), and h) shows details of the images.

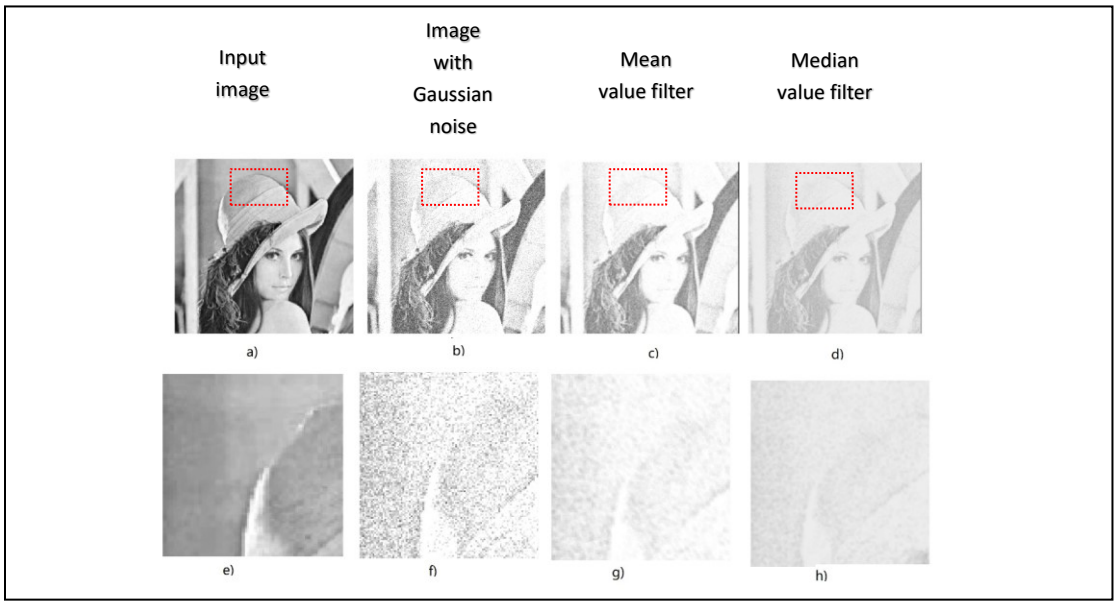


Figure 4.27 Results of smoothing filters a) the original image b) image with Gaussian noise c) using Mean Value filter to remove noise d) using Median Value filter to remove noise.

Table 4.6 shows test results of the artificial intelligence implementation. The decision-making shows that the hand-writing images can be recognized by the system.

Table 4.6 Convolution results of each input hand-writing image.

	Output for recognize '5'	**Output for recognize '6'**	**Decision**
	327	350	'6'
	275	253	'5'
	344	403	'6'
	279	265	'5'
	352	384	'6'

	291	272	‘5’
	319	380	‘6’
	340	336	‘5’

4.3. Conclusion and Analysis

In Chapter 4 we have shown programmable resistor-based circuit designs for feedforward neural network architecture implementations. The first circuit was used for edge detection by a **1*4 digital potentiometer array**. In the test, the 1*4 digital potentiometer array works with other analog devices such as resistors, instrumentation amplifiers, DACs and ADCs for DSP algorithm realizations. Figures 4.10 and 4.11 show processed images by a one dimensional ‘analog DSP filter’ [1, -1, 1, -1] and a two dimensional ‘analog DSP filter’ $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. As shown in the test results, edges can be successfully detected by the circuit. Next, based on the same principle, a **2*3*3 multiply-accumulation system** is used for matrix computing applications, such as computer vision and artificial intelligence. Figure 4.26 shows test results and computer simulations of applying the sharpening filters that are based on the Laplacian operator and Sobel. Figure 4.27 shows test results of applying the smoothing filters that are Mean and Median. Table 7.4 shows test results of hand-writing number images recognitions. The above test results show that the analog multiply-accumulation system is compatible with current algorithm applications. The following paragraphs provide more detailed analysis.

1*4 digital potentiometer array: as a starting point of the circuit designs, this test shows that it is possible to design analog circuits by digital algorithms. Instead of

conventional methods for algorithm implementations on Von Neumann architecture, some linear algorithms can be implemented by programmable resistor matrices based on Kirkoff's Law. As shown in Table 4.1, different 'zeros' of filters can be converted to their corresponding convolutional kernels, which shows connections between DSP and ISP algorithms. As shown in figures 4.10 and 4.11, the 'analog DSP filters' are able to detect edges as expected.

2*3*3 multiply-accumulation system: the system is a matrix computer from the perspective of computing, but it is an 'analog DSP filter' for multiple frequency components from the perspective of DSP. Based on this idea, we only need to expand the 1*4 digital potentiometer array to a larger digital potentiometer matrix. Test results of this section show that it is possible to 'transplant' algorithms of computers to this platform. Figure 4.28 shows an analysis for the Laplacian operator test. The red area with high intensity gradients is filtered by the system successfully but the yellow area with low gradients is not filtered totally. The errors are caused by inaccuracy during parameters-resistances conversions.

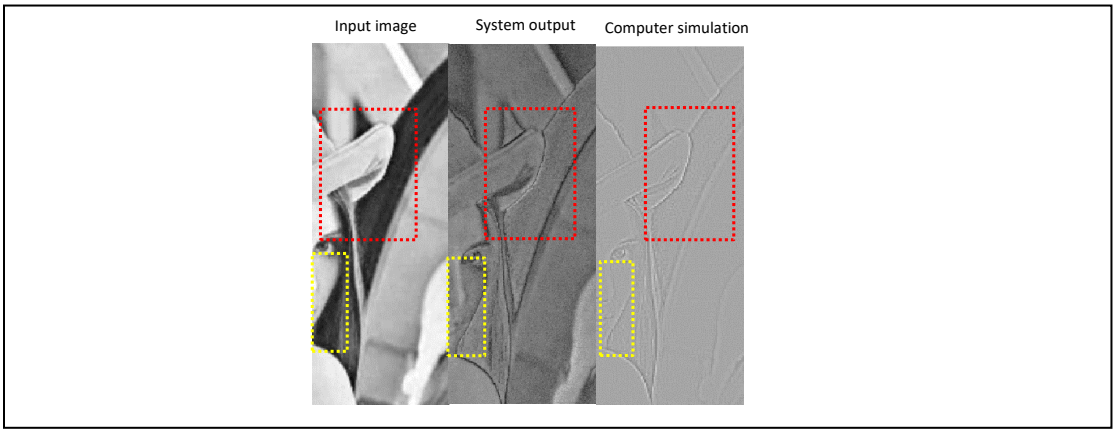


Figure 4.28 Comparing image processing results of the multiply-accumulation system with the computer simulation.

When we apply Sobel filter, as shown in figure 4.29, the edge detection result become better. The red areas show the accuracy of the system is lower than the computer simulation. The lip area is not clear in the expected output image since the Sobel filter is only sensitive for a single direction, so it remained in the output of the system. In conclusion, the system is functional for convolution and image processing tasks. For improvements, better components and circuit designs can be implemented.

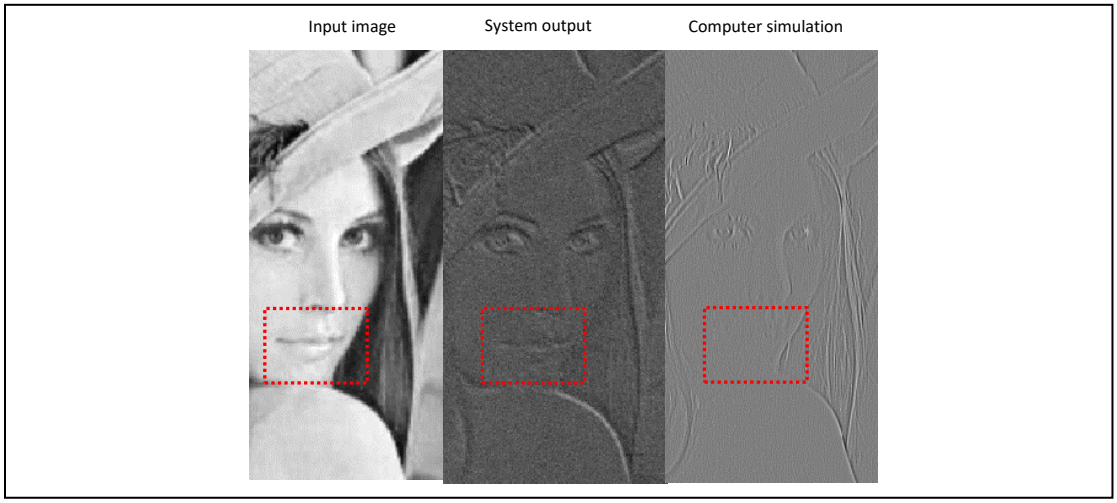


Figure 4.29 Comparing image processing results of the multiply-accumulation system with the computer simulation.

Figure 4.27 shows details of noise removing capability by applying smoothing filters for images with noise. Figure 4.27 (g) and (h) show that the smoothing filters are compatible with the system. By comparing the results of smoothing filters with sharpening filters, the system has better performance for smoothing filters. We consider the smoothing filters that are based on addition, but the sharpening filters are based on differentiation, the inaccuracy may be caused by unbalanced outputs of the digital

potentiometer matrices. Therefore, as the sharpening filters cannot remove low frequency areas totally, but for the smoothing filters this has less influence.

Next, we tried to implement artificial intelligent algorithms on this system. In order to load all the convolutional kernels and input images, the analog multiply-accumulation system has to be helped by a computer because of it has insufficient units. Test results of table 4.4 show that the system can learn and recognize the 4 images of '5' and '6' correctly. Its relative DSP algorithms include convolution, correlation, and down-sampling. In the chapter 6, an idea for system expansion is discussed.

Chapter 5

Backward Structure Programmable Resistor Circuits for Solving Neural Networks

5.1. Programmable Resistor Based Feedback Loop

5.1.1. Introduction

The 'programmable resistor-based feedback loop' is the most basic component for implementing circuits with the backward structure. On one hand the feedback loops are self-adapting circuits, on the other hand they are analog calculators for solving ratios of two values.

5.1.2. Algorithm Design

As shown in figure 5.1, instead of only implementing neural networks, circuits of this section are self-adaptive. A programmable resistor can be adjusted by a feedback loop under a group of input/output when the circuit is set to training mode. The circuit can implement a neural network when the circuit is set to implementing mode. There are two relative algorithms that are analog divider and proportional control for removing fluctuations.

Analog divider: a feedback loop can act as an analog divider for solving a neural network with equation form a*w=b where a and b are known values and w is an unknown

value. Assuming we apply an input voltage V_{in} as 'a' and a target voltage V_{ref} as 'b', the memorized ratio of the voltage divider is 'w' (eqn. 5.1).

$$w = \frac{R_{recorded}}{100k\Omega} = \frac{V_{ref}}{V_{in}} = \frac{b}{a} \quad \textbf{(5.1)}$$

Next, we can feed in a new voltage V_{sig} as a new 'a' for implementing the neural network. Assuming we apply an input signal V_{sig} (eqn. 5.2),

$$V_{out} = V_{sig} * \frac{R_{recorded}}{100k\Omega} = a_{new} * w = b_{new} \quad \textbf{(5.2)}$$

From the above equations, the circuit has the capabilities of self-adapting and self-implementing.

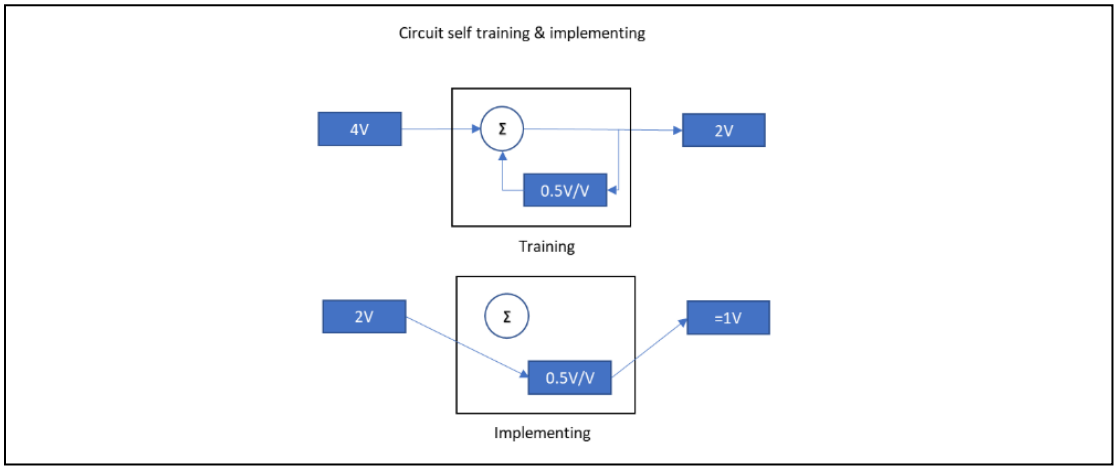


Figure 5.1 Self-adapting characteristic of feedback loop.

Proportional control for fluctuation removal: conventionally, clocks of digital ICs are fixed frequency signals. However, fixed frequency clocks lead to fluctuations at stable state of feedback loops because they keep adjusting the loops with time. Therefore, we apply a proportional control algorithm [50] to stop the loop when V_{out}=V_{ref} and operate

the loop when $V_{out} \neq V_{ref}$ (algorithm simulation is in section 1.6 of appendix E). Figure 5.2 illustrates the algorithm by computer simulation and the algorithm block diagram is shown in figure 5.3.

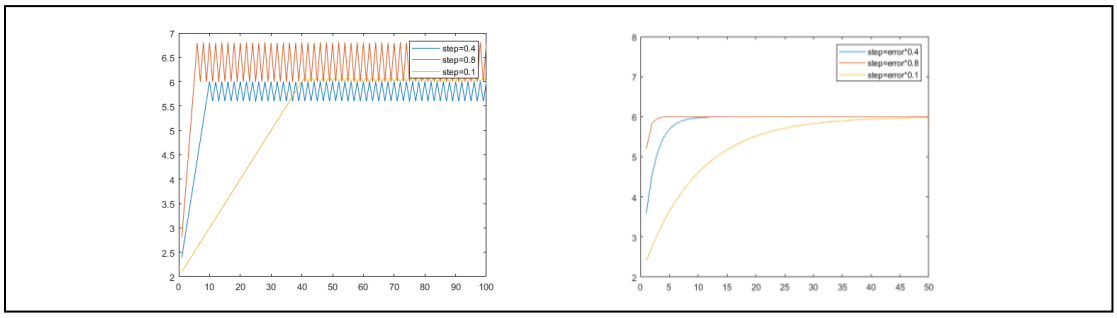


Figure 5.2 Illustration of fixed step control and proportional control.

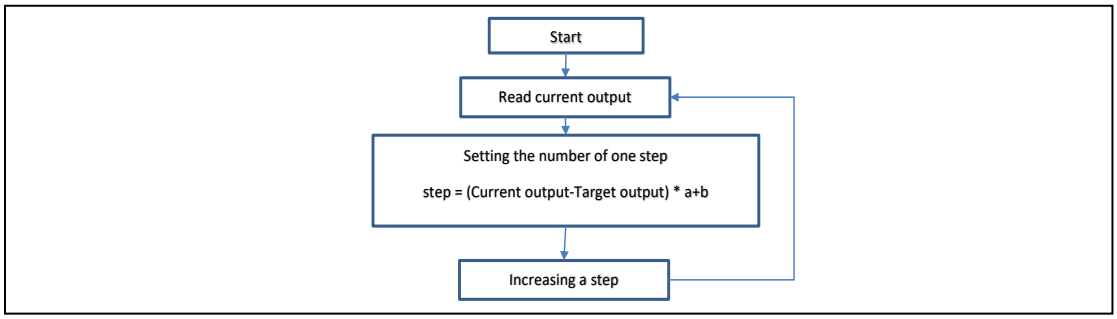


Figure 5.3 Proportional control algorithm.

5.1.3. Algorithm Hardware Implementation

In terms of the **analog divider**, figure 5.4 shows hardware implementation for the programmable resistor-based feedback loop, it has a digital potentiometer as a voltage divider and a comparator. The comparator keeps the output voltage of the voltage divider equal to the reference voltage by generating increase/decrease signals. Its schematic is in figure Appendix C.7 and its actual circuit implementation is in figure Appendix D.7.

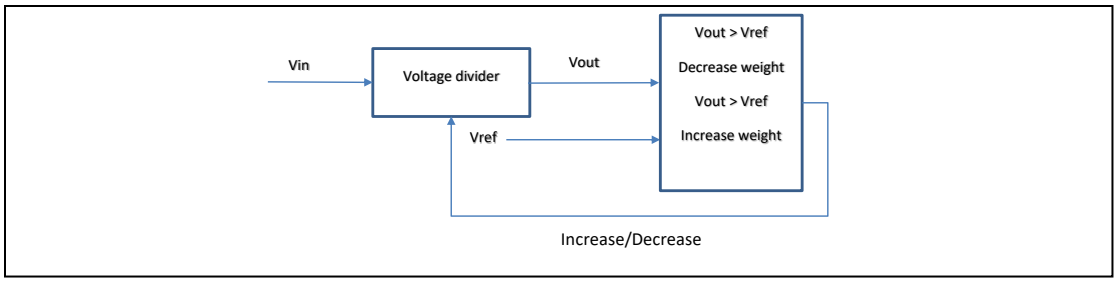


Figure 5.4 Programmable resistor-based feedback loop hardware implementation.

In terms of the **proportional controlled clock strategy**, we apply an equation for calculating real-time frequency of clock signal in a MCU for the clock generator control (eqn. 5.3).

$$CLK = \left[\frac{V_{in}*\left(\frac{R_w}{R_{total}}\right)-V_{ref}}{5} * 1024\right] \text{Hz} \quad \textbf{(5.3)}$$

Where V_{in} is the input voltage, V_{ref} is the reference voltage, R_w is current resistance of the digital potentiometer, and R_{total} is the total resistance of the digital potentiometer.

5.1.4. Test Bench and Test Method

Tests of this section include proportional control for fluctuation removal and analog divider computing. We connect the test bench as in table 5.1, the system block diagram is given in figure 5.5. Its schematic is in figure Appendix C.14.

Table 5.1 Proportional control self-adjustment controller test bench**.**

Inputs	Measurement Device	Circuit being tested	Analysis Software
DAC array	Digilent analog discovery 2	Programmable resistor-based feedback loop	Digilent Waveforms
	Arduino Mega		MATLAB

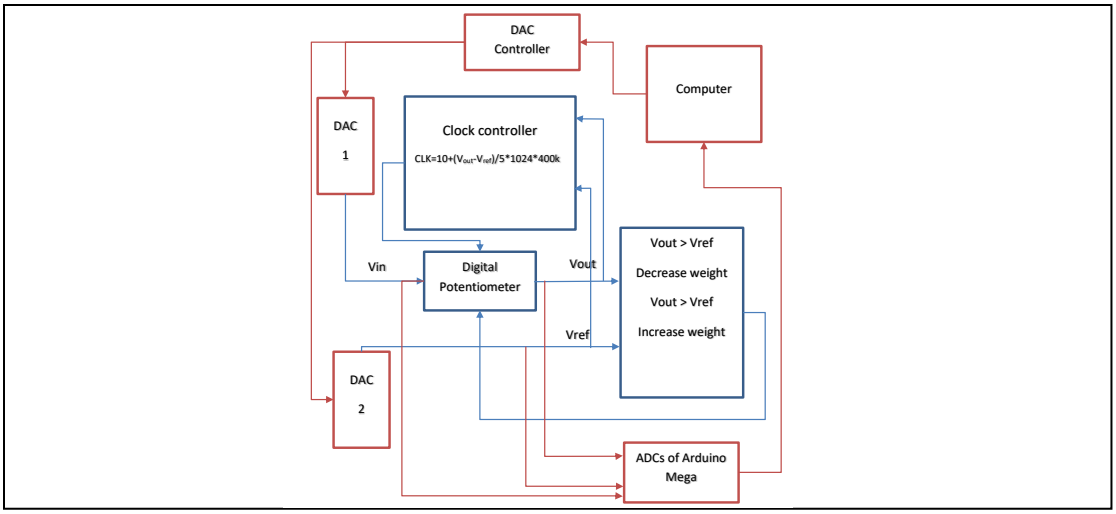


Figure 5.5 Block diagram of the testbench for testing the resistor-based feedback loop circuit (Red blocks are testing devices).

During the tests, the test bench generates required input voltage V_{in} and reference voltage V_{ref} through the DAC1 and DAC2. The clock controller, which is a MCU with the algorithm of proportional control, reads voltages of V_{out} and V_{ref} and changes the frequency of the square wave for the digital potentiometer.

For testing the **proportional control clock strategy**, first we switch the target voltage V_{ref} between 0V and 4V frequently for testing the voltage following capability of the circuit. After this we apply fixed clock frequencies of 10kHz and 100kHz and do the same voltage following tests for comparisons. Simultaneously, we capture the output voltages through the Analog Discovery 2 using the software Digilent Waveforms.

For testing the **analog divider** computation, first we generate a random input x (0~5) and a random weight w (0~1). We then calculate the expected solution y by solving equation 5.4 in MATLAB. The values of x, w, and y are recorded for comparison.

$$y = x * w \tag{5.4}$$

Next, we convert the x and y to voltages through the DAC matrix and feeding into the feedback loop circuit where x is the input voltage and y is the target voltage. Then, the solution is solved by the feedback loop as the resistance in the digital potentiometer as equation 5.5.

$$w = y/x \tag{5.5}$$

Next, we set the DAC matrix to 5V for reading the data w from the digital potentiometer. Also, we do the conversion for expressing 'w' as a fraction (eqn. 5.6). We store the w in MATLAB for comparison.

$$w = V_{read}/5 \tag{5.6}$$

Finally, the process is repeated for 1000 times and the differences between the w solved by the feedback loop with the expected w are plotted.

Next, we tried to implement an analog divider for image memorizing. First, we convert the color matrix of a 256-color image to an analog matrix through the DAC array and feed in the analog matrix to the feedback loop. Next, we set all elements of DAC matrix to 5v for reading the memorized data and save them into MATLAB. Finally, we compare the memorized image with the input image.

5.1.5. Test Results

Figure 5.6 shows details of **proportional control clock strategy**. There are output

curves from each of the three clock strategies. Comparing with the curve of the 10kHz clock, the curve of the changing frequency signal has a faster adjusted speed. Comparing with the curve of the 100kHz clock, the curve of the changing frequency signal has lower fluctuations at the stable stage. Figure 5.7 shows the test result of the **analog divider**, the curve is calculated by equation 5.7,

$$R = W_{MATLAB} - W_{Feedback\ loop} \quad \textbf{(5.7)}$$

Where W_{MATLAB} is the expected weight array, $W_{Feedback\ loop}$ is the array that calculated by the circuit. The error curve shows that the outputs of the circuit are lower than the expected values. Figure 5.8 shows test result of using the analog divider for memorizing, showing that the w values solved by the feedback loop can be kept to small errors. The large errors shown may be caused by noise during state changes.

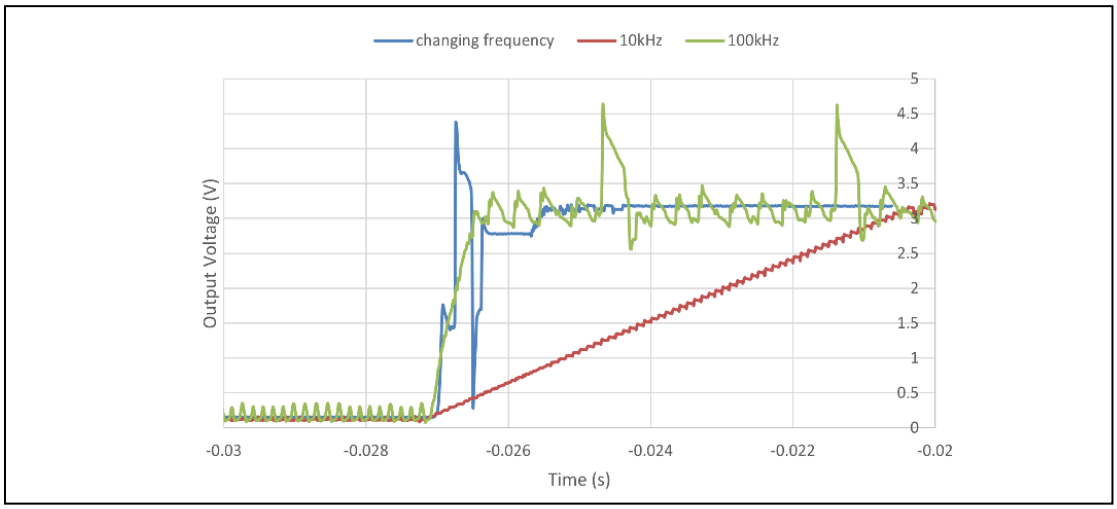


Figure 5.6 Detail of the three control strategies: Blue line: proportional control resistor adjustment strategy. Green line: 100kHz clock adjustment clock strategy. Red line: 10kHz clock adjustment clock strategy.

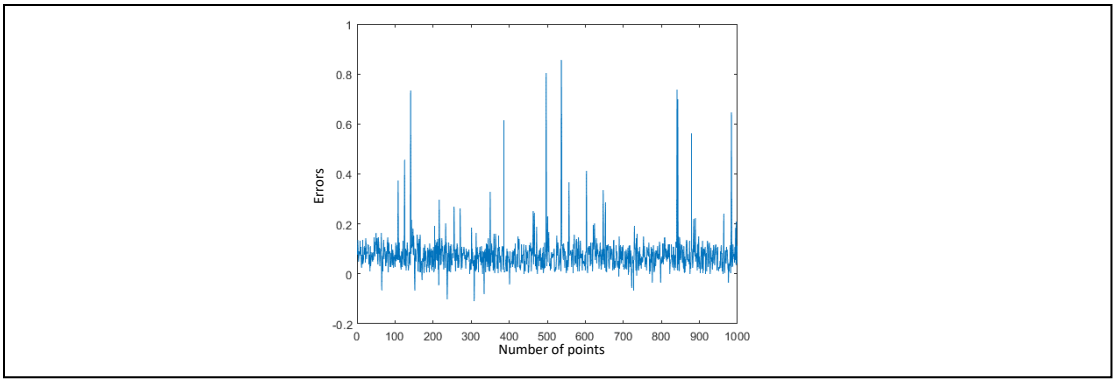


Figure 5.7 Learning results error curve.

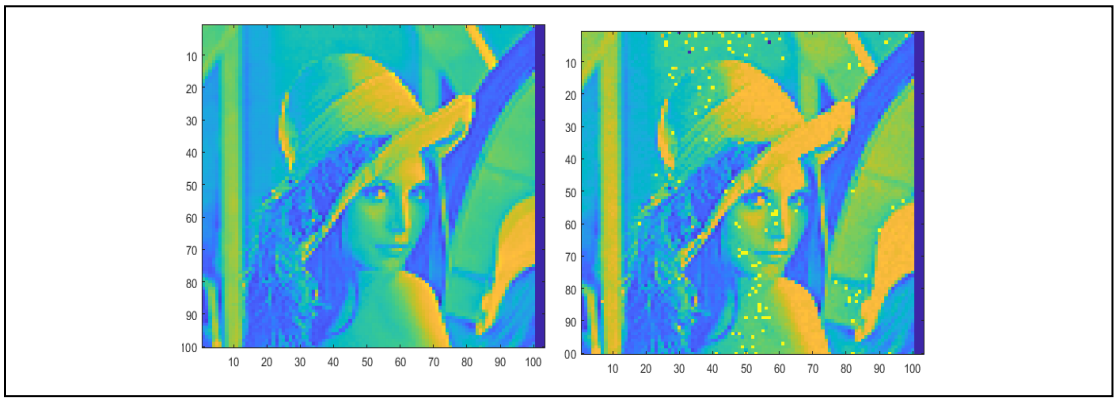


Figure 5.8 Provided image for learning (left) and the learnt image (right).

5.2. Add-Division Circuit for Training Convolutional Kernels

5.2.1. Introduction

This section shows a circuit for hardware implementing the convolutional kernel training algorithm which is mentioned in section 4.2.2. The circuit includes a 1*3 digital potentiometer array and 1*3 programmable resister-based feedback loops. It can calculate ‘add-division’ of the convolutional kernel training without using accumulators and multipliers.

5.2.2. Algorithm Design

The convolutional kernel training starts from a random 3*3 weight matrix and its convolution is shown by equation 5.8 and equation 5.9.

$$w_{m,n} = \frac{Temp_{m,n}}{\sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{Temp}_{i,j}} \qquad ,1 \le m \le 3, 1 \le n \le 3 \tag{5.8}$$

$$\begin{cases} y = \sum_{j=1}^{3}\sum_{i=1}^{3} x_{i,j} * w_{i,j} \\ \sum_{j=1}^{3}\sum_{i=1}^{3} w_{i,j} = 1 \end{cases} \tag{5.9}$$

Next, one element of the 3*3 weight matrix is increased by a step (eqn. 5.10), the weight matrix is renewed (eqn. 5.10), and calculates its 'next step convolution' y' (eqn. 5.11).

$$\begin{cases} w_{1,1} = \frac{Temp_{1,1}+step}{(\sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{Temp}_{i,j})+step} & ,m = 1, n = 1 \\ w_{m,n} = \frac{Temp_{m,n}}{(\sum_{j=1}^{3}\sum_{i=1}^{3} \mathrm{Temp}_{i,j})+step} & ,others \end{cases} \tag{5.10}$$

$$\begin{cases} y' = \sum_{j=1}^{3}\sum_{i=1}^{3} x_{i,j} * w_{i,j} \\ \sum_{j=1}^{3}\sum_{i=1}^{3} w_{i,j} = 1 \end{cases} \tag{5.11}$$

If the 'next step convolution' y' is higher than the 'current convolution' y, it means that the new convolutional kernel is closer to the expected convolutional kernel. Therefore, the element keeps increasing. If the 'next step convolution' y' is lower than the 'current convolution' y, it means that the new convolutional kernel is far away from to the expected convolutional kernel. Therefore, the element is decreased by subtracting the step (eqn. 5.12).

$$step = \begin{cases} step & ,y' > y \\ -step & ,y' < y \end{cases} \tag{5.12}$$

From the training algorithm implementations, we find that accumulators and

multipliers are used frequently. In order to implement the algorithm by circuit level hardware, we can consider voltage relationships of a serial connected resistor circuit (eqn. 5.13), assuming N resistors are serially connected.

$$V_{DD} = V_1 + V_2 + V_3 + \cdots + V_N \quad \textbf{(5.13)}$$

Where V_n are voltage difference of each resistor and V_{DD} is power supply. Dividing each of them by V_{DD} (eqn. 5.14)

$$1 = \frac{V_1}{V_{DD}} + \frac{V_2}{V_{DD}} + \cdots + \frac{V_N}{V_{DD}} \quad \textbf{(5.14)}$$

And considering the equation 5.11, sum of the weight matrix should be 1 (eqn. 5.15).

$$1 = w_1 + w_2 + \cdots w_N = \frac{t_1}{t_1+t_2+\cdots+t_N} + \frac{t_2}{t_1+t_2+\cdots+t_N} + \cdots + \frac{t_N}{t_1+t_2+\cdots+t_N} \quad \textbf{(5.15)}$$

Comparing the equation 5.14 with equation 5.15, an element of the convolution kernel can be expressed by ratio of two voltages (eqn. 5.16).

$$w_n = \frac{V_n}{V_{DD}} = \frac{t_n}{t_1+t_2+\cdots+t_N} \quad \textbf{(5.16)}$$

As mentioned above, V_n can be generated by serially connected resistors and the weight w_n can be memorized by the 'resistor-based feedback loop' as V_{ref}/V_{DD}. Then, the weights are memorized in the digital potentiometers. When feeding in a random voltage matrix, the output voltage is calculated by equation 5.17.

$$V_{out} = V_1 * w_1 + V_2 * w_2 + \cdots + V_N * w_N \quad \textbf{(5.17)}$$

As shown in equation 5.17, the weights w_n are generated by linear changing digital potentiometers. The weights would be re-assigned when one digital potentiometer is

adjusted, but the sum of weights is always 1. For the actual circuit design, a 1*3 digital potentiometer array is used with a 1*3 resistor-based feedback loop.

5.2.3. Algorithm Implementation

In the test, we used a 1*3 digital potentiometer array and a 1*3 feedback loop array. Therefore, the serially connected digital potentiometer array has equation 5.18.

$$5V = V_1 + V_2 + V_3 \tag{5.18}$$

Also, the 1*3 feedback loop array has equation 5.19,

$$1 = \frac{V_1}{5V} + \frac{V_2}{5V} + \frac{V_N}{5V} \tag{5.19}$$

Therefore, sums of the weight matrix that are memorized by the feedback loop matrix are always 1.

When the circuit is working, a MCU is used for controlling the resistor array and the switches. The circuit generates a normalized weight matrix by several steps. Potentiometers 1->6 are marked in figure 5.9.

Step 1: Adjusting the digital potentiometers 1~3 (eqn. 5.20).

$$\begin{cases} 5v = V_1 + V_2 + V_3 \\ V_1 = \frac{\mathrm{R_{Potentiometer1}}}{\mathrm{R_{Potentiometer1}}+\mathrm{R_{Potentiometer2}}+\mathrm{R_{Potentiometer3}}} \\ V_2 = \frac{\mathrm{R_{Potentiometer2}}}{\mathrm{R_{Potentiometer1}}+\mathrm{R_{Potentiometer2}}+\mathrm{R_{Potentiometer3}}} \\ V_3 = \frac{\mathrm{R_{Potentiometer3}}}{\mathrm{R_{Potentiometer1}}+\mathrm{R_{Potentiometer2}}+\mathrm{R_{Potentiometer3}}} \end{cases} \tag{5.20}$$

Where V_1, V_2, V_3 are voltage differences of potentiometers 1~3 and sum of V_1+V_2+V_3 is always 5V. Next, V_1, V_2, V_3 are sent to the feedback loops as target voltages.

Step 2: Enabling the digital potentiometers 4~6 and switching their inputs to 5V. Under the input signal 5V, each feedback loop starts following their reference voltages which are V1, V2, V3. Finally, the potentiometers 4~6 have stable resistances when the feedback loops arrive the target voltages. The ratios of the resistances of the potentiometers 4~6 with 100 kΩ are the normalized weights w1~w3 (eqn. 5.21).

$$\begin{cases} \mathrm{w}_1 = \frac{V_1}{5\mathrm{v}} = \frac{\mathrm{R}_{\mathrm{Potentiometer4}}}{100\mathrm{k}\Omega} \\ \mathrm{w}_2 = \frac{V_2}{5\mathrm{v}} = \frac{\mathrm{R}_{\mathrm{Potentiometer5}}}{100\mathrm{k}\Omega} \\ \mathrm{w}_3 = \frac{V_3}{5\mathrm{v}} = \frac{\mathrm{R}_{\mathrm{Potentiometer6}}}{100\mathrm{k}\Omega} \end{cases} \quad \textbf{(5.21)}$$

Where V_1, V_2, V_3 are the voltage differences of potentiometers 1~3 from step 1. $R_{potentiometer4}$~$R_{potentiometer6}$ are resistances generated by the feedback loops and recorded by the digital potentiometers.

From above, if we calculate w1+w2+w3, we can find (eqn. 5.22).

$$\mathrm{w}_1 + \mathrm{w}_2 + \mathrm{w}_3 = \frac{V_1}{5\mathrm{V}} + \frac{V_2}{5\mathrm{V}} + \frac{V_3}{5\mathrm{V}} = \frac{V_1+V_2+V_3}{5\mathrm{V}} = \frac{5\mathrm{V}}{5\mathrm{V}} = 1 \quad \textbf{(5.22)}$$

Step 3: Disabling the digital potentiometers 4~6 to stop the feedback self-adjustments locking the resistances of each potentiometer. Next is switching the input signals from 5V to the DAC array outputs V_{sig1} V_{sig2} V_{sig3} for convolution computation. Then, the outputs of potentiometers 4~6 are feed into an analog accumulator. Finally, the output of the analog accumulator is the convolution result (eqn. 5.23). Its schematic is in figure Appendix C.9 and its actual circuit implementation is in figure Appendix D.9.

$$V_{out} = V_{sig1} * w1 + V_{sig2} * w2 + V_{sig3} * w3 \quad \textbf{(5.23)}$$

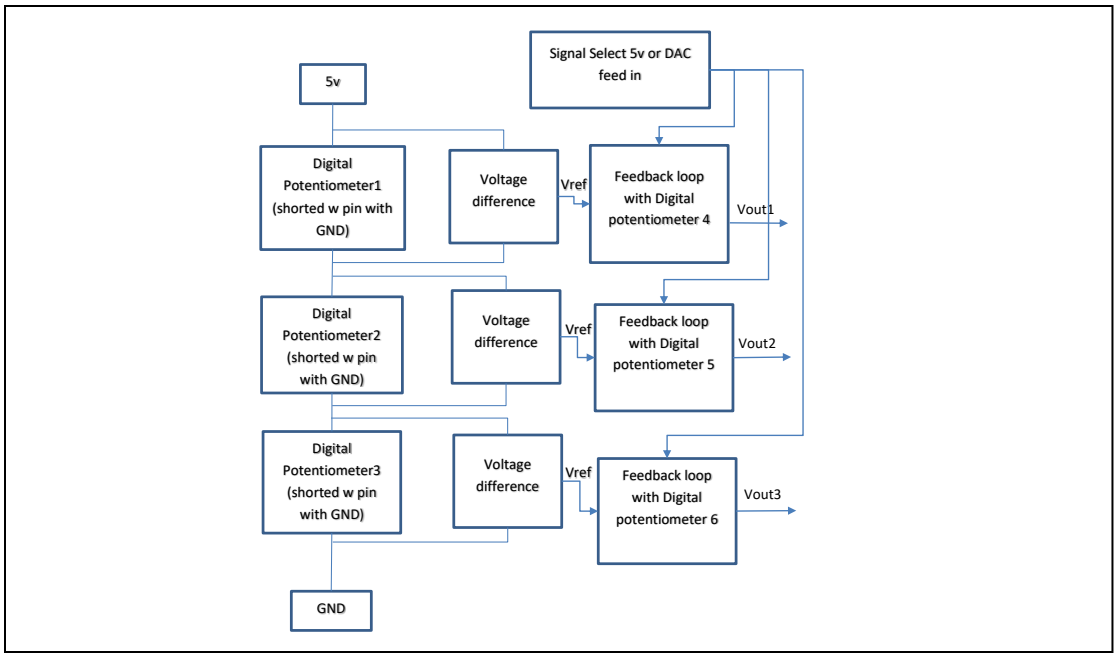


Figure 5.9 Block diagram of the add-division circuit.

5.2.4. Test Bench and Test Method

Table 5.2 Gradient descent test bench

Inputs	Measurement Device	Circuit being tested	Analysis Software
DAC array	Arduino Mega	Normalized weights generator	MATLAB

We connect the test bench as in table 5.2 and the system block diagram of figure 5.10. MATLAB controls the DAC matrix to generate required analog matrices through the DAC controller. During the test, the digital potentiometers 1, 2, 3 are changed linearly and the instrumentation amplifiers take voltage difference of each of them. Next, the voltage differences are fed into the feedback loop matrix as V_{ref} and the feedback loops generate a 1*3 weight matrix with a constant sum of 1. Finally, the circuit controller switches the

input signals to the DAC matrix and the multiply-accumulation signals are captured by the Arduino Mega (its schematic is in figure Appendix C.15).

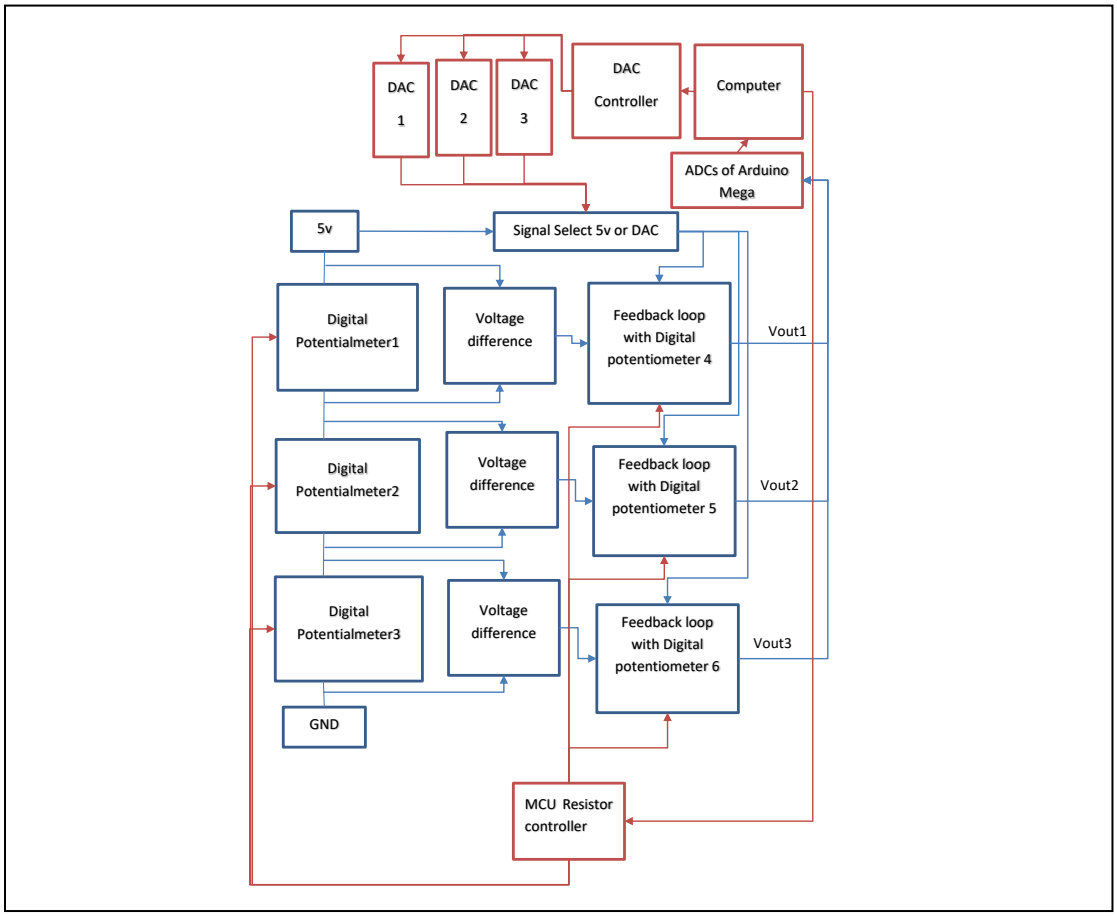


Figure 5.10 Block diagram of the test bench for the add-division circuit (Red blocks are testing devices).

In terms of the test, instead of just measuring convolution, we use the circuit for measuring gradients since gradients are more meaningful than convolution for convolutional kernels training. First, we measure the gradient at current point where R1=100kΩ， R2=100kΩ, and R3=100kΩ under input voltages 5V, 3.52V, and 1.37V (eqn 5.24).

$$\begin{cases} V_{out}(w_1, w_2, w_3) = 5V * w_1(R_1, R_2, R_3) + 3.52V * w_2(R_1, R_2, R_3) + 1.37V * w_3(R_1, R_2, R_3) \\ w_1(R_1, R_2, R_3) = \frac{R_1}{R_1+R_2+R_3} \\ w_2(R_1, R_2, R_3) = \frac{R_2}{R_1+R_2+R_3} \\ w_3(R_1, R_2, R_3) = \frac{R_3}{R_1+R_2+R_3} \end{cases} \quad \textbf{(5.24)}$$

By considering that gradient is found by the equation 5.25 in the software algorithm

$$k = \frac{f(x)-f(x+step)}{step} = \frac{y_{next}-y_{current}}{step} \quad \textbf{(5.25)}$$

We set 50kΩ as one step, then the gradient is calculated by the equation 5.26.

$$\begin{cases} k_{R1} = \frac{Vout(R1)-Vout(R1+50k\Omega)}{1k\Omega} \\ = \frac{\left(V1*\frac{R1}{R1+R2+R3}+V2*\frac{R2}{R1+R2+R3}++V3*\frac{R3}{R1+R2+R3}\right)-\left(V1*\frac{R1+50k\Omega}{R1+50k\Omega+R2+R3}+V2*\frac{R2}{R1+50k\Omega+R2+R3}++V3*\frac{R3}{R1+50k\Omega+R2+R3}\right)}{1k\Omega} \\ k_{R2} = \frac{Vout(R2)-Vout(R2+50k\Omega)}{1k\Omega} \\ = \frac{\left(V1*\frac{R1}{R1+R2+R3}+V2*\frac{R2}{R1+R2+R3}++V3*\frac{R3}{R1+R2+R3}\right)-\left(V1*\frac{R1}{R1+50k\Omega+R2+R3}+V2*\frac{R2+50k\Omega}{R1+50k\Omega+R2+R3}++V3*\frac{R3}{R1+50k\Omega+R2+R3}\right)}{1k\Omega} \\ k_{R3} = \frac{Vout(R3)-Vout(R3+50k\Omega)}{1k\Omega} \\ = \frac{\left(V1*\frac{R1}{R1+R2+R3}+V2*\frac{R2}{R1+R2+R3}++V3*\frac{R3}{R1+R2+R3}\right)-\left(V1*\frac{R1}{R1+R2+50k\Omega+R3}+V2*\frac{R2}{R1+50k\Omega+R2+R3}++V3*\frac{R3+50k\Omega}{R1+50k\Omega+R2+R3}\right)}{1k\Omega} \end{cases} \quad \textbf{(5.26)}$$

Where R1, R2, R3 represent resistances of the digital potentiometers 1, 2, 3 and V_{out} (R1), V_{out} (R2), V_{out} (R3) represent output voltages from the digital potentiometers 1, 2, 3. For simplifying the equations, we use R1, R2, and R3 to represent resistances of the digital potentiometers 1, 2, 3 for the following equations of this section.

During the tests, we keep input voltages at [5v, 3.52v, 1.37v] and change each digital potentiometer for measuring its outputs' gradient. Repeating the procedure of changing 1 step and measuring output voltage, moving back and measuring output for 100 times. Table 5.3 lists details of the test.

Table 5.3 Test method.

Initial conditions		1 Steps	Measuring outputs	Gradient	Repeat measurement
R1=100kΩ	W1=1/3	-50 kΩ	$V1*\frac{R1}{R1+R2+R3}+V2*\frac{R2}{R1+R2+R3}++V3*\frac{R3}{R1+R2+R3}$	Vdrop/step	100 times
R2=100kΩ	W2=1/3	-50 kΩ	$V1*\frac{R1}{R1+R2+R3}+V2*\frac{R2}{R1+R2+R3}++V3*\frac{R3}{R1+R2+R3}$	Vdrop/step	100 times
R3=100kΩ	W3=1/3	-50 kΩ	$V1*\frac{R1}{R1+R2+R3}+V2*\frac{R2}{R1+R2+R3}++V3*\frac{R3}{R1+R2+R3}$	Vdrop/step	100 times

Figure 5.11 illustrates the test method.

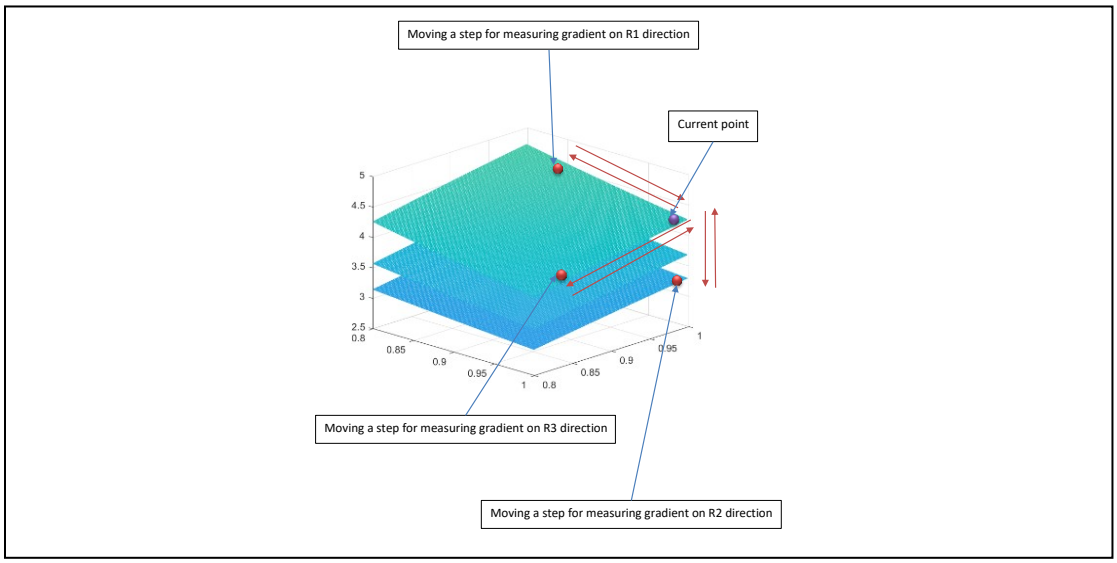


Figure 5.11 Test method illustration by computer simulation plot.

Next, based on the method for measuring gradient at current point we use the circuit for measuring gradients at specific points of the saddle surface of equation 6.22. Also, we use computer simulation to generate the same saddle surface and measuring gradients at the same points for comparisons (eqn. 5.27).

$$\begin{cases} \mathrm{z}\,(w_1,w_2,w_3)=5*w_1(x_1,x_2,x_3)+3.52*w_2(x_1,x_2,x_3)+1.37*w_3(x_1,x_2,x_3) \\ w_1(x_1,x_2,x_3)=\frac{x_1}{x_1+x_2+x_3} \\ w_2(x_1,x_2,x_3)=\frac{x_2}{x_1+x_2+x_3} \\ w_3(x_1,x_2,x_3)=\frac{x_3}{x_1+x_2+x_3} \end{cases} \quad \textbf{(5.27)}$$

Where x1, x2, x3 correspond to R1, R2, R3, z corresponds to the output voltage of the circuit. Figure 5.12 illustrates the test method. During the test, we set R1, R2, R3 of the circuit and x1, x2, x3 of the computer simulation simultaneously and record the gradients measurements. Table 5.4 shows the test method for the circuit.

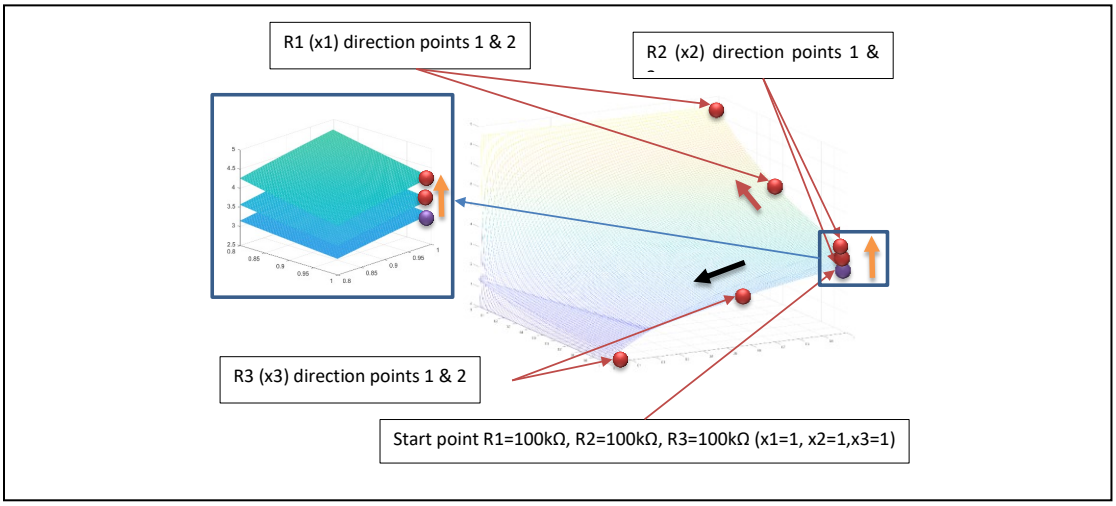


Figure 5.12 Test method illustration by computer simulated surface z=5*w1+3.52*w2+1.37*w3.

Table 5.4 Initial condition in the test.

Initial point						
	R1(Ω)	Actual W1	R2(Ω)	Actual W2	R3(Ω)	Actual W3
Start point	100k	0.33	100k	0.33	100k	0.33
Moving R1						
	R1(Ω)	Actual W1	R2(Ω)	Actual W2	R3(Ω)	Actual W3
Point 1 at R1 direction	50k	0.2	100k	0.4	100k	0.4
Point 2 at R1 direction	0k	0	100k	0.5	100k	0.5
Moving R2						
	R1(Ω)	Actual W1	R2(Ω)	Actual W2	R3(Ω)	Actual W3
Point 1 at R2 direction	100k	0.4	50k	0.2	100k	0.4
Point 2 at R2 direction	100k	0.5	0k	0	100k	0.5
Moving R3						
	R1(Ω)	Actual W1	R2(Ω)	Actual W2	R3(Ω)	Actual W3
Point 1 at R3 direction	100k	0.4	100k	0.4	50k	0.2
Point 2 at R3 direction	100k	0.5	100k	0.5	0k	0

Table 5.5 shows the test method for the computer simulation. Next, we change all

the digital potentiometers for decreasing gradients. From the test results of gradients measurements on single direction we can estimate a net-direction for decreasing its gradients (table 5.6). Table 5.7 shows test method for moving to net-direction. Instead of starting from [100kΩ, 100kΩ, 100kΩ], we start from [50kΩ, 50kΩ, 50kΩ] so that the resistance of three digital potentiometers can be increased or decreased. First we move the resistors to the middle point [50 kΩ, 50 kΩ, 50 kΩ]. Next, we increase or decrease the resistors. We also use the computer to simulate the changes for comparison (table 5.8).

Table 5.5 Initial condition of computer simulation.

Initial point						
	X1(R1)	Actual W1	X2(R2)	Actual W2	X3(R3)	Actual W3
Start point	1	0.33	1	0.33	1	0.33
Moving x1 (R1)						
	X1(R1)	Actual W1	X2(R2)	Actual W2	X3(R3)	Actual W3
Point 1 at R1 direction	0.5	0.2	1	0.4	1	0.4
Point 2 at R1 direction	0	0	1	0.5	1	0.5
Moving x2 (R2)						
	X1(R1)	Actual W1	X2(R2)	Actual W2	X3(R3)	Actual W3
Point 1 at R2 direction	1	0.4	0.5	0.2	1	0.4
Point 2 at R2 direction	1	0.5	0k	0	1	0.5
Moving x3 (R3)						
	X1(R1)	Actual W1	X2(R2)	Actual W2	X3(R3)	Actual W3
Point 1 at R3 direction	1	0.4	1	0.4	0.5	0.2
Point 2 at R3 direction	1	0.5	1	0.5	0	0

Table 5.6 Net-gradient with R1,R2,R3.

	R1	**R2**	**R3**
Decreasing net-gradient	Increase	Increase	Decrease

Table 5.7 Initial condition in the test.

Initial point						
	R1(Ω)	Actual W1	R2(Ω)	Actual W2	R3(Ω)	Actual W3
Start point	100k	0.33	100k	0.33	100k	0.33
Decreasing gradient						
	R1(Ω)	Actual W1	R2(Ω)	Actual W2	R3(Ω)	Actual W3
Point 1	50k	0.33	50k	0.33	50k	0.33
Point 2	100k	0.5	100k	0.5	0k	0

Table 5.8 Initial condition of computer simulation.

Initial point						
	X1(R1)	Actual W1	X2(R2)	Actual W2	X3(R3)	Actual W3
Start point	1	0.33	1	0.33	1	0.33
Decreasing gradient						
	X1(R1)	Actual W1	X2(R2)	Actual W2	X3(R3)	Actual W3
Point 1	0.5	0.33	0.5	0.33	0.5	0.33
Point 2	1	0.5	1	0.5	0	0

5.2.5. Test Results

Current point gradient measurement: Figure 5.13 shows curves of gradients by the circuit and by the computer simulation. We can find that measurements of the circuit are close to the computer simulations although they have large fluctuations.

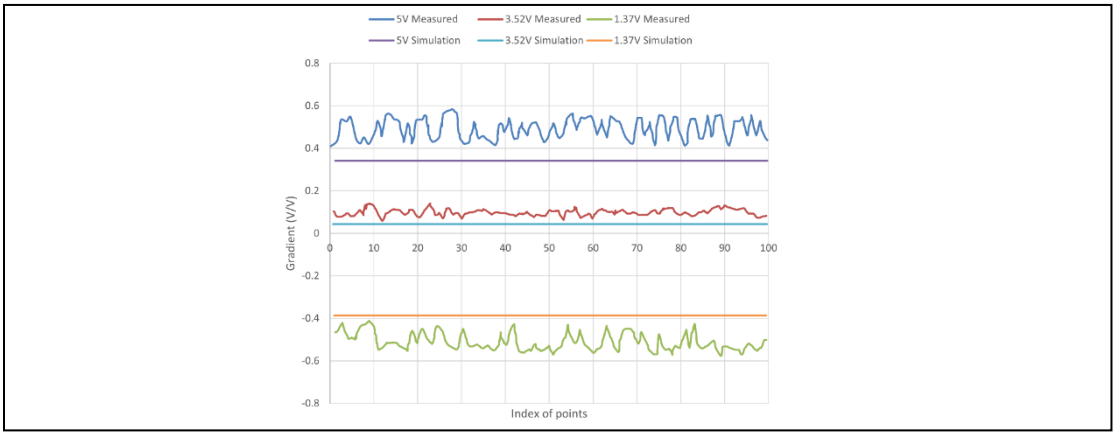


Figure 5.13 Gradient measurements of the test circuit and gradient measurements of computer simulation.

Gradient measurements on multiple points toward single direction: As shown in left images of figure 5.14, figure 5.15, and figure 5.16. The recorded data of '5V measurements', '3.52V measurements', and '1.37V measurements' represent the output voltage curves of the digital potentiometers which have input voltage 5V, 3.52V, and 1.37V respectively. The curves of '5V average curve', '3.52V average curve', and '1.37V average curve' represent the mean values of each 100 points of the measured data. We take the average curve from the measured data for comparing with computer simulation in the next step. The right images of figure 5.14, figure 5.15, and figure 5.16 show comparisons between the measured data and computer simulation. We can find that the outputs of the circuit are not accurate, but they have similar trends as the computer simulations.

Gradient measurements of Moving R1 (x1 in computer simulation):

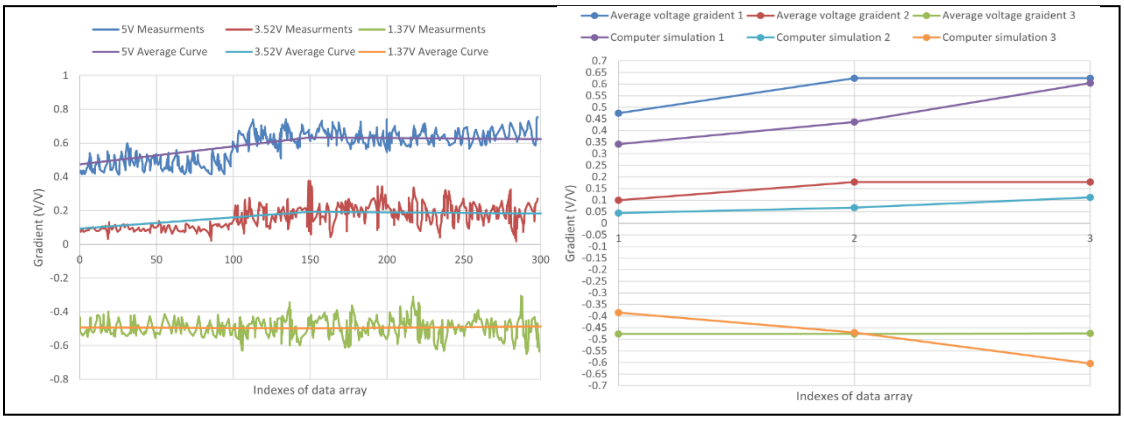


Figure 5.14 Moving R1 gradient measurements. The actual voltage gradient measurements and the average voltage gradients (left). Computer simulation (Right).

Gradient measurements of Moving R2 (x2 in computer simulation):

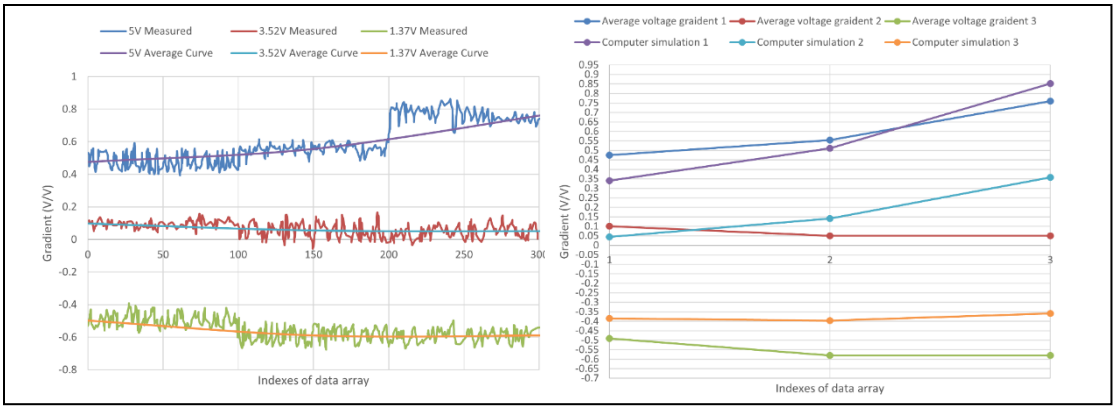


Figure 5.15 Moving R2 gradient measurements. The actual voltage gradient measurements and the average voltage gradients (left). Computer simulation (Right).

Gradient measurements of Moving R3 (x3 in computer simulation):

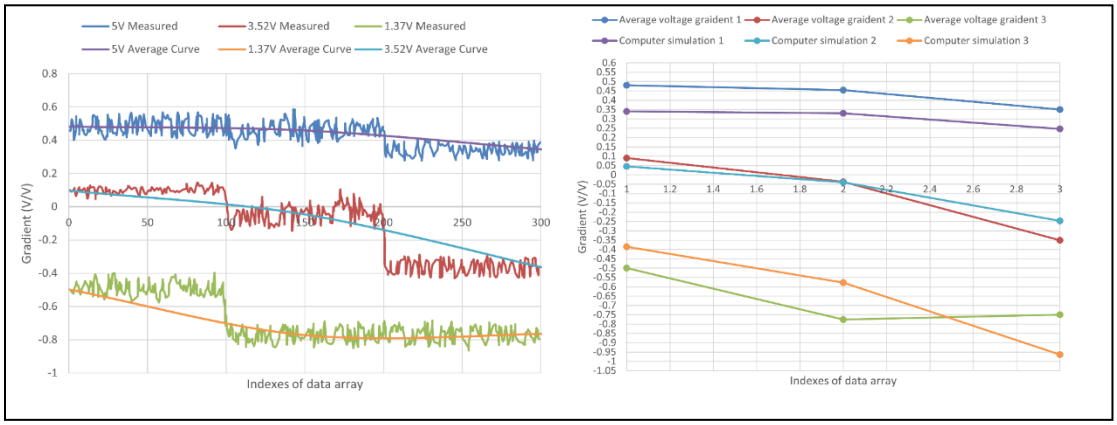


Figure 5.16 Moving R3 gradient measurements. The actual voltage gradient measurements and the average voltage gradients (left). Computer simulation (Right).

From the test results, all the gradient changing curves are monotonic increasing or decreasing curves. We can estimate a net-direction for decreasing the gradients (table 5.9). In the following test, we change R1, R2, R3 and x1, x2, x3 and measure the gradients.

Table 5.9 summary of relationship between gradient and R1,R2,R3

	Gradient changes
R1 direction	Increase
R2 direction	Increase
R3 direction	Decrease

As shown in left images of figure5.17. The recorded data of '5V measurements', '3.52V measurements', and '1.37V measurements' represent the output voltage curves of the digital potentiometers which have input voltage 5V, 3.52V, and 1.37V respectively. The curves of '5V average curve', '3.52V average curve', and '1.37V average curve' represent the mean values of each 100 points of the measured data. We take the average curve from the measured data for comparing with computer simulation in the next step. The right images of figure 5.17 show comparisons between the measured data and computer simulation. We find that the gradients decreased from the second point to the third point.

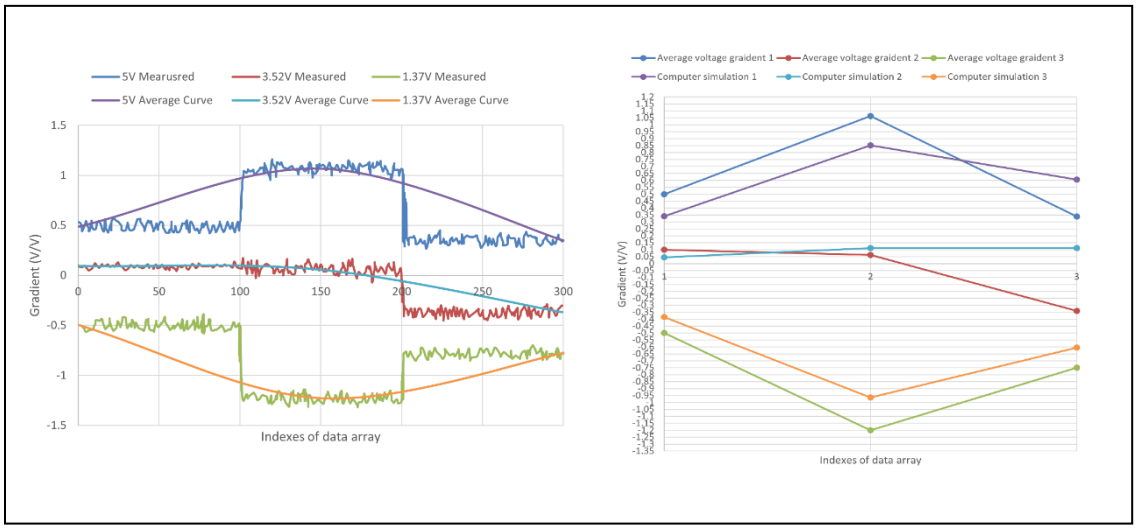


Figure 5.17 Moving R1, R2, R3 gradient measurements. The actual voltage gradient measurements and the average voltage gradients (left). Computer simulation (right).

5.3. Random Matrix Generator for Solving Diophantine Equations

5.3.1. Introduction

This test is to use the 'random matrix generator' to generate a plane x+y+z=1 for solving Diophantine equations. As we known, humans cannot solve Diophantine equations with form $y=a*x_1+b*x_2+c*x_3$ since there are infinite numbers of combinations of $[x_1, x_2, x_3]$. However, their solutions can be expressed on planes that are constructed by a basic plane x+y+z=1. This section shows a hardware implementing method for constructing the basic plane x+y+z=1 and use it for solving multiple unknown variables from multiple Diophantine equations.

5.3.2. Algorithm Design

As described in section 2.3.4, assuming a simple fully connected neural network that only has one output node y and three input nodes x_1, x_2, x_3 (eqn. 5.28).

$$y = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 \quad \textbf{(5.28)}$$

We assume the output y and the inputs x are provided, we want to solve $[w_1,w_2,w_3]$. The general method for solving multiple unknown variables from multiple Diophantine equations is to calculate them step by step. For demonstration, solving three equations with three unknowns. Supposing we already know 3 groups of inputs and outputs (eqn. 5.29).

$$\begin{cases} 5 = 10 * w_1 + 15 * w_2 + 20 * w_3 \\ 6 = 24 * w_1 + 18 * w_2 + 12 * w_3 \\ 7 = 14 * w_1 + 28 * w_2 + 21 * w_3 \end{cases} \quad \textbf{(5.29)}$$

The expected results are w_1=0.118, w_2=0.1, w_3=0.116 by solving w1~w3 manually. However, the solution also represents a cross point of three planes. Trying another way to solve the unknows by finding cross point of the three planes. Firstly, convert the equations to intercept forms (eqn. 5.30).

$$\begin{cases} 1 = 2 * w_1 + 3 * w_2 + 4 * w_3 \\ 1 = 4 * w_1 + 3 * w_2 + 2 * w_3 \\ 1 = 2 * w_1 + 4 * w_2 + 3 * w_3 \end{cases} \quad \textbf{(5.30)}$$

The three planes have interceptions as given in table 5.10.

Table 5.10 Interception of the equations.

	W1	**W2**	**W3**
Eq1	½	1/3	¼
Eq2	¼	1/3	½
Eq3	1/2	1/4	1/3

Next, we use a computer to generate large numbers of random points to construct a fundamental plane x+y+z=1 in 3D space. Applying the equations for random points generation (eqn. 5.31).

$$\begin{cases} x + y + z = 1 \\ \mathrm{x} = \dfrac{\text{Random number1}}{\text{Random number1+Random number2+Random number3}} \\ \mathrm{y} = \dfrac{\text{Random number2}}{\text{Random number1+Random number2+Random number3}} \\ \mathrm{z} = \dfrac{\text{Random number3}}{\text{Random number1+Random number2+Random number3}} \end{cases} \quad \textbf{(5.31)}$$

Comparing equation 5.30 with equation 5.31, we give [x,y,z] of x+y+z=1 some parameters so that the three planes of equation 5.30 can be constructed by the plane x+y+z=1. Generating three planes by x+y+z=1 (table 5.11).

Table 5.11 Converting 1=a+b+c to interceptions.

	W1	W2	W3
Eq1	1/2*x	1/3*y	1/4*z
Eq2	1/4*x	1/3*y	1/2*z
Eq3	1/2*x	1/4*z	1/3*z

Next, we plot the three planes in a 3D space and find the cross point in the MATLAB.

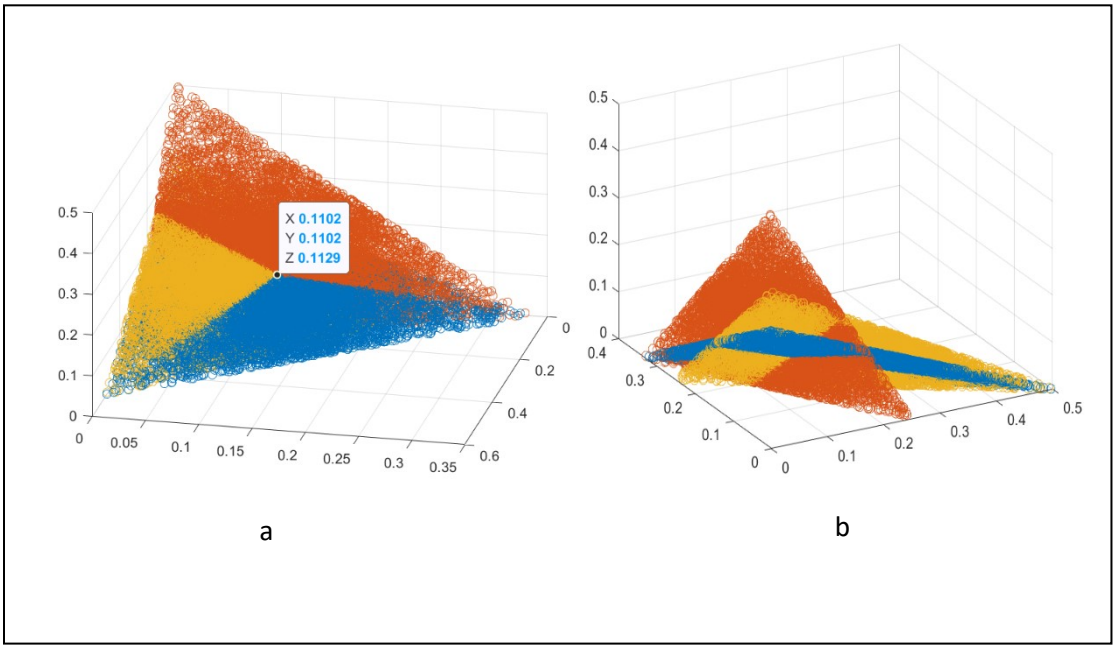


Figure 5.18 Solving weight matrix from three planes a) the cross point of the three planes represents solution of the three Diophantine equations b) observing the three planes from another angle.

As shown in figure 5.18, some points are at the crossing of the three planes. The solution can be estimated by selecting a point. This solution has low accuracy (table 5.12), but this method can be applied on the circuit of section 5.8.2, which generates random points by circuit noise.

Table 5.12 Compare expected solution with result.

	x	y	z
Expect solution	0.118	0.1	0.116
Solved by exhaustivity method	0.1102	0.1102	0.1129

It is possible to solve a Diophantine equation by a resistor-based feedback loop. For example, solving w_1, w_2, w_3 of a Diophantine equation with known inputs x_1, x_2, x_3 and known output y (eqn. 5.32),

$$y = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 \quad \textbf{(5.32)}$$

We can consider the algorithm simulation above, the infinite numbers of solutions [w1, w2, w3] are generated by computers sequentially and the random numbers are based on 'pseudorandom algorithms'. Next, we design a 'hardware algorithm' for implementing the basic plane x+y+z=1 and using it for solving Diophantine equations. The random numbers can be generated by noise of circuits and the matrix can be generated by a feedback loop. The following paragraphs show details of the relative algorithms: **Solving Diophantine equation x+y+z=1 by a feedback loop** and **a noise-driven random generator**.

Solving Diophantine equation x+y+z=1 by a feedback loop: we can assume that we have a feedback loop with a 1*3 digital potentiometer array. The circuit has output voltage relationships that are solutions to the equation 5.33.

$$\frac{V_{out(1)}}{V_{feedback}} + \frac{V_{out(2)}}{V_{feedback}} + \frac{V_{out(3)}}{V_{feedback}} = 1 \quad \textbf{(5.33)}$$

Where V_{out1}, V_{out2}, V_{out3} are random voltages of the digital potentiometer array, and $V_{feedback}$ is always 5V. The ratio of $V_{out}/V_{feedback}$ is saved in digital potentiometers by their

resistance. Next, as previously described, any Diophantine equation can be converted to its intercept form as equation 5.34.

$$\frac{x_1 * w_1}{y} + \frac{x_2 * w_2}{y} + \frac{x_3 * w_3}{y} = 1 \quad \textbf{(5.34)}$$

Comparing equation 5.7 and 5.8 and doing a conversion so that the unknown matrices [w1,w2,w3] are represented by output matrices [$V_{out(1)}/V_{feedback}$, $V_{out(2)}/V_{feedback}$, $V_{out(3)}/V_{feedback}$] yield equation 5.35.

$$w_n = \frac{V_{out(n)}}{V_{feedback}} * \frac{y}{x_n} \quad \textbf{(5.35)}$$

Where y and x_n are known, $V_{out(n)}$ is original output voltage of the circuit, $V_{feedback}$ is fixed 5V. Therefore, if measuring the output voltages, all matrices [$V_{out(1)}/V_{feedback}$, $V_{out(2)}/V_{feedback}$, $V_{out(3)}/V_{feedback}$] are available for the solution.

Noise-driven random number generator: first, if we recall the proportional control algorithm test result of section 5.1.5, the comparator generates increase/decrease commands rapidly at the stable stage under 100kHz clock (figure 5.19). Next, we consider that the jitter of clock generators is caused by noise of the circuits. The three digital potentiometers would be changed individually and randomly if we feed in three square waves (same frequency) with jitter for the three digital potentiometers respectively. Figure 5.20 illustrates the three clocks with jitter on each. The increasing and decreasing commands are assigned to the three digital potentiometers randomly because of the jitter, but the summed voltage is locked by the feedback loop.

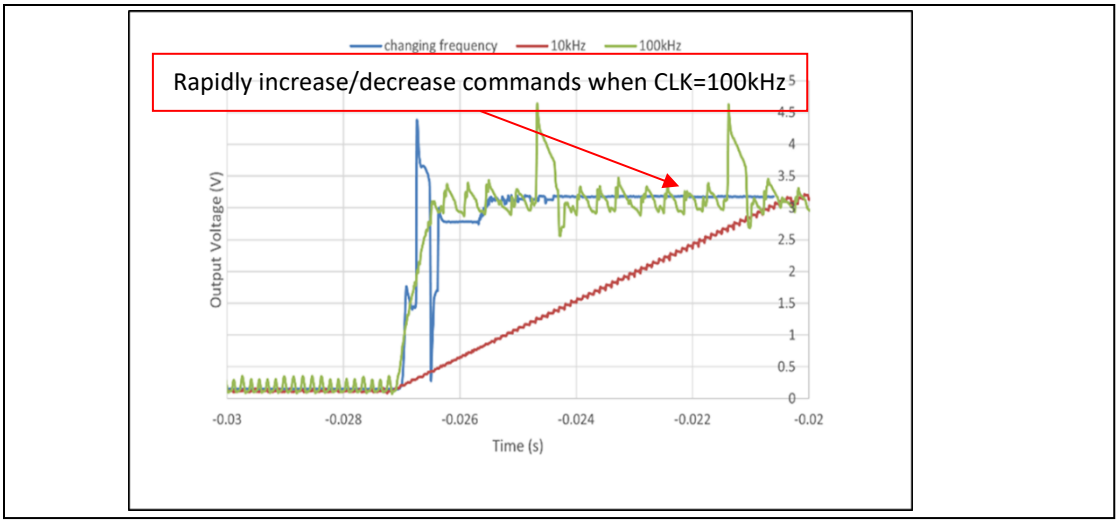


Figure 5.19 The self-adjustment curve of computer simulation (left) and the measured actual self-adjustment curve (right).

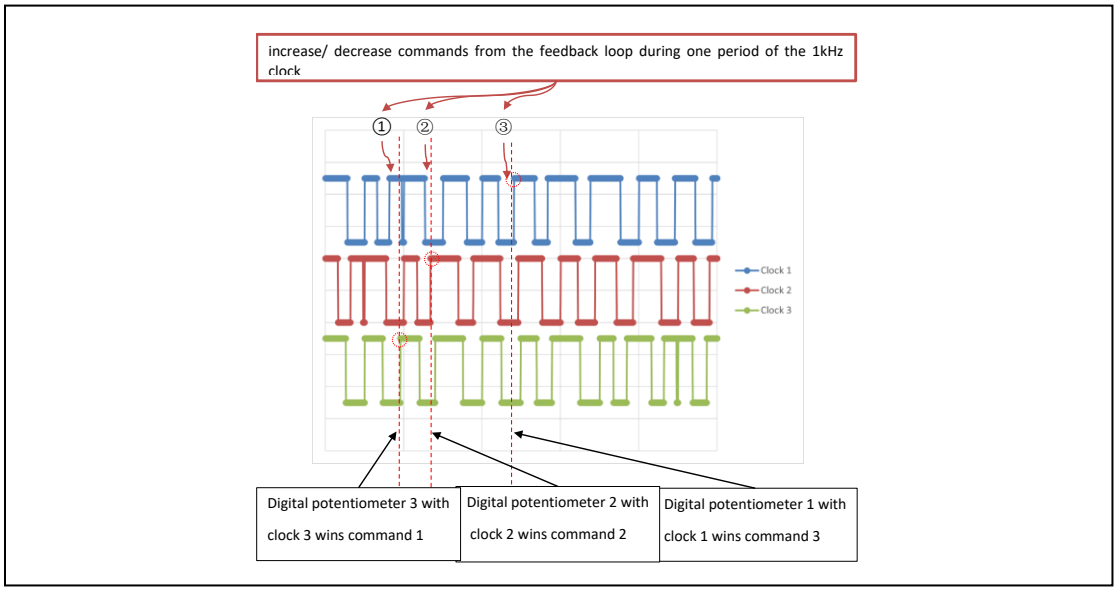


Figure 5.20 Principle of the random matrix generator.

5.3.3. Algorithm Implementation

Figure 5.21 shows a block diagram of the circuit design. The hardware implementation is based on a feedback loop with a multiply-accumulation unit. Also, each digital potentiometer has their individual clock generator. The three independent clocks

are important for generating random outputs since each clock generator is affected by jitter at random time. The random phenomenon disappears when the three digital potentiometers share the same clock generator, which shows that the circuit is driven by noise (its schematic is in figure Appendix C.8 and its actual circuit implementation is in figure D.8).

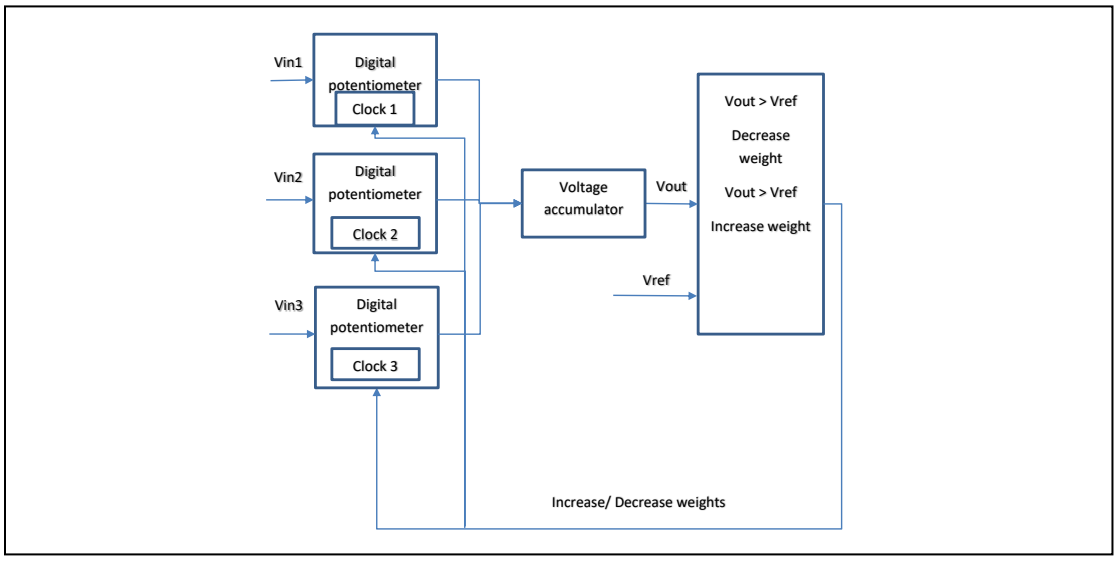


Figure 5.21 Block diagram of the true random matrix generator.

After the circuit is powered, the feedback loop would arrive at its balanced stage that V_{out} is equal to V_{ref}. At this time, the comparator starts to generate increasing/decreasing commands rapidly and one of the digital potentiometers would capture a command at random time. For explanation, we assume that one the digital potentiometers has captured an increasing command and V_{ref} is 5V. After this, sum of outputs of the digital potentiometers would be higher than 5V. The comparator then generates a decreasing command for keeping the feedback voltage at 5V. Next, one of the digital potentiometers would capture this decreasing command and sum of outputs

of the digital potentiometers would be lower than 5V. Then, the loop goes to the start state again. Therefore, the three digital potentiometers change randomly but their summed voltages are 5V constantly.

5.3.4. Test Bench and Test Method

Table 5.13 Test bench.

Inputs	Measurement Devices	Circuit being tested	Software
DAC array	Arduino Mega	Random matrix generator	MATLAB

Table 5.13 shows a test bench for the circuit and figure 5.22 shows the block diagram of the testbench. MATLAB controls the three DACs through the DAC controller for generating input voltage matrices. The feedback voltage of the loop is set to fixed 5V. The Arduino Mega is used for measuring output voltages of the three digital potentiometers and the feedback voltage.

There is also a clock generator controller for controlling the clock generators. During the test we set the 3 clock generators to fixed 100kHz square waves and enable the potentiometers feedback loops. Next, we set the DAC matrix outputs [5V, 5V, 5V] through MATLAB. Finally, we record the outputs of the three potentiometers and save the results in MATLAB (its schematic is in figure Appendix C.16).

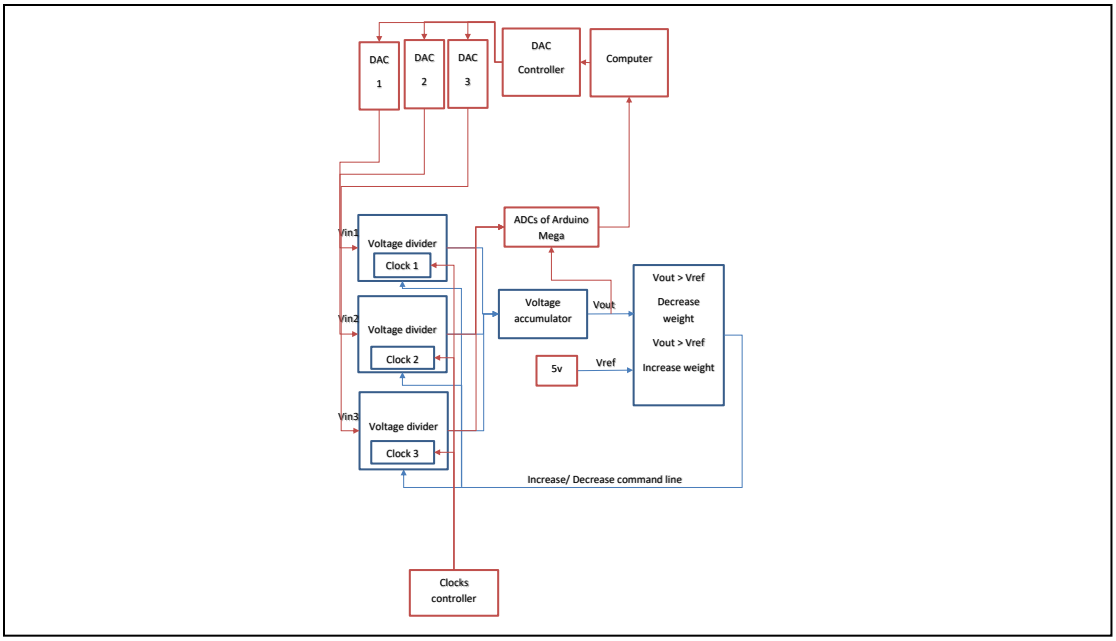


Figure 5.22 Block diagram of the test bench for the 'true random matrix generator' (Red blocks are testing devices).

Solving single Diophantine equation: the measured data require conversions from voltage values to numbers. For the test, V_{ref} is 5V and 5V= V_1+ V_2+V_3, we can write the conversion as equation 5.36.

$$\begin{cases} x = \frac{V_1}{5} \\ y = \frac{V_2}{5} \\ z = \frac{V_3}{5} \end{cases} \quad \textbf{(5.36)}$$

Where V_1, V_2, V_3 are output voltages of the digital potentiometers.

Solving multiple Diophantine equation: test of this section is based on computer simulation since it requires devices for locking multiple outputs which are difficult to implement. As we known, three unknown variables can be solved from three Diophantine equations. Assuming inputs and outputs of a neural network are already provided (eqn. 5.37) but [w_1, w_2, w_3] is unknown.

$$\begin{cases} 5 = 5 * w_1 + 5 * w_2 + 5 * w_3 \\ 5 = 3.9 * w_1 + 3 * w_2 + 5 * w_3 \\ 5 = 1.96 * w_1 + 3.9 * w_2 + 5 * w_3 \end{cases} \quad \textbf{(5.37)}$$

We use the same method for converting them to intercept equations (eqn. 5.38)

$$\begin{cases} 1 = w_1 + w_2 + w_3 \\ 1 = \frac{3.9}{5} * w_1 + \frac{3}{5} * w_2 + w_3 \\ 1 = \frac{1.96}{5} * w_1 + \frac{3.9}{5} * w_2 + w_3 \end{cases} \quad \textbf{(5.38)}$$

Next, we construct the target planes by using the plane x+y+z=1 which is generated by the circuit (table 5.14). It shows the conversions for the three Diophantine equations. We use MATLAB to simulate the conversion results and plotting the planes of [w_1, w_2, w_3].

Table 5.14 Converting 1=x+y+z planes to the target planes.

	W1	**W2**	**W3**
Eq.1	V_{out1}	V_{out2}	V_{out3}
Eq.2	V_{out1}*5/3.9	V_{out2}*5/3	V_{out3}
Eq.3	V_{out1}*5/1.96	V_{out2}*5/3.9	V_{out3}

5.3.5. Test Results

Solving single Diophantine equation: Figure 5.23 shows the results of the tests. Figure 5.23 (a) shows that all the outputs are random in the 2D space. Figure 5.23 (b) shows the feedback voltage of the feedback loop is kept at 3V. Figure 5.23 (c) shows that the circuit generates a plane 1=x+y+z controlled by the feedback loop. Also, we pick some points from the measurements for verifying that the circuit can generate x+y+z=1 plane (Table 5.15). It shows that sums of the outputs are always close to 1.

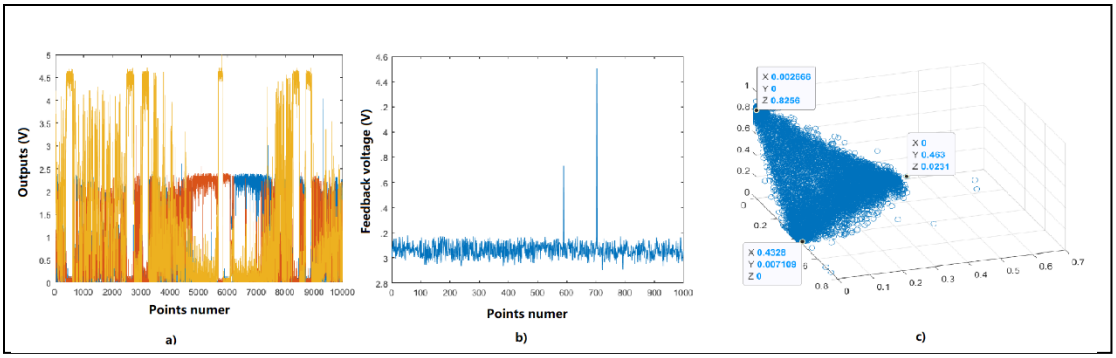


Figure 5.23 Test results of the 'true random matrix generator. a) outputs of the three digital potentiometers on 2D plots. b) The feedback voltage of the loop. c) Plotting [Vout1, Vout2, Vout3] on a 3D plane 1=x+y+z.

Table 5.15 Some random points of outputs

	x	y	z	sum
Random point 1	0.40	0.36	0.03	0.78
Random point 2	0.12	0.03	1.44	1.59
Random point 3	0.01	0.73	0	0.75
Random point 4	0.65	0.10	0.12	0.87
Random point 5	0.23	0.62	0	0.85
Random point 6	0.05	0.76	0.01	0.82
Random point 7	0.18	0.32	0.63	1.12
Random point 8	0.31	0.03	0.58	0.92
Random point 9	0.22	0.17	0.81	1.21
Random point 10	0.47	0.14	0.13	0.74

Solving multiple Diophantine equations: Figure 5.23 shows the test result which contains three planes of the equation 5.38. Solution for the weight matrix [w_1, w_2, w_3] is at a cross point of the three planes. We use MATLAB to find points which are close to each other from the three planes. Figure 5.24 shows the target points which are in the expected region. Picking a point for verifying the solution

$$\begin{cases} 0.7931 = \frac{5}{3} * 0.03216 + \frac{5}{3} * 0.0494 + \frac{5}{3} * 0.3943 \\ 0.7484 = \frac{3.9}{3} * 0.03216 + \frac{3}{3} * 0.0494 + \frac{5}{3} * 0.3943 \\ 0.7424 = \frac{1.96}{3} * 0.03216 + \frac{3.9}{3} * 0.0494 + \frac{5}{3} * 0.3943 \end{cases} \quad \textbf{(5.39)}$$

We find that results of the three Diophantine equations are close (eqn. 5.39). Obviously, this method has low accuracy, but it shows a new method for solving neural networks by random numbers.

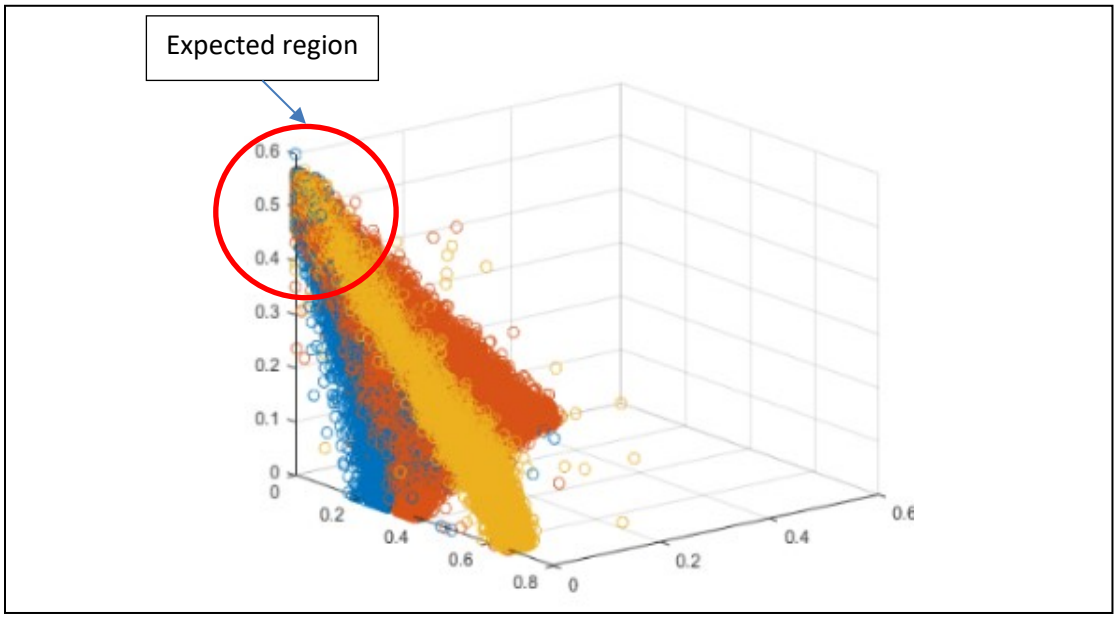


Figure 5.24 Results of converting x+y+z=1 plane to the three Diophantine equations.

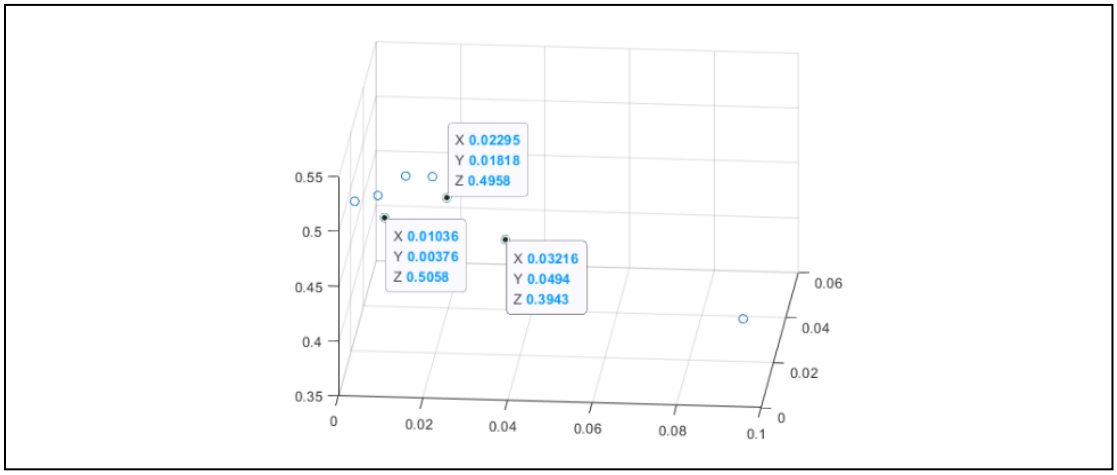


Figure 5.25 Cross points of the three planes.

5.4. Conclusion and Analysis

In Chapter 5 we have shown programmable resistor-based circuit designs for backward neural network architecture implementations or hardware implemented neural network solvers. The first design is a programmable resistor-based feedback loop that is the most basic component of feedback loops. In the second design, we use a 1*3 digital potentiometer array and a 1*3 programmable resistor-based feedback loops to implement the convolutional kernel training algorithm on circuit-level. In the third design, we use a 'random matrix generator' that is based on a noise-driven multiply-accumulation feedback loop for solving Diophantine equations. The following provides some more detailed analysis.

Programmable resistor-based feedback loops: As discussed, the algorithm design of section 5.1.2 shows that the feedback loop can solve V_{ref}/V_{in} by an analog computing method and save the results as resistances by the same programmable resistor. The capability can be called 'analog computing-in-memory'. Also, proportional controlled clock strategy is used for attenuating fluctuations for getting higher accuracy, which is different from conventional fixed frequency clocks of digital devices. In the tests, the circuit not only shows computing characteristics as shown in figure 5.7, but also shows memorizing characteristics as shown in figure 5.8. However, from the test results of figure 5.7, the outputs of the circuit were generally lower than the expected values. The memorized image of figure 5.8 shows some noisy points that are caused by instabilities of the circuit.

'Add-division' circuit: this circuit shows an example of implementing a practical algorithm by programmable resistor-based linear and non-linear architectures. Figure 5.13 shows test results of gradient measurements on single point by the circuit. Figure 5.14, 5.15, 5.16 are gradient measurements on multiple points toward single direction by the circuit. Figure 5.17 shows a result of moving the point toward net-direction for decreasing its gradients. As shown in the above test results, although the outputs of the circuit are inaccurate when compared with the computer simulations, they have similar trends as the expected values. Therefore, it is possible to use the circuit for convolutional kernel training applications under restricted circumstances such as low power consumption usage.

'Random matrix generator' circuit: the circuit is based on an analog multiply-accumulation unit of chapter 4 and a feedback loop. The circuit generates random matrices by noise-driven clocks, but their outputs are locked by the feedback loop. The feedback loop constructs the x+y+z=1 plane by 'shaping' the random points. Figure 5.23 shows test results of constructing the x+y+z=1 plane by the circuit. As shown in figure 5.23 (a) the outputs are random, but in figure 5.23 (c) the random points are restricted on the same plane. From table 5.15, we can see some solutions for Diophantine equation x+y+z=1. After this, we use the basic plane for solving Diophantine equation groups. Figures 5.24 and 5.25 show estimated solutions for a Diophantine equation group. We find that although the solutions are inaccurate, the circuit can one-step solve multiple unknown values.

The test results also show some unexpected results. As shown in figure 5.23 (a), the actual measurements of the three outputs are not as random as expected. We plot the three outputs separately as shown in figure 5.26. Two of the digital potentiometers have similar changing tendency, but the other one has a quite different changing tendency. This phenomenon may be caused by individual differences of ICs such as input/output resistances and frequency responses. Small differences of wire length of circuits, ICs' parameters, and circumstances can affect the final outputs in these random systems.

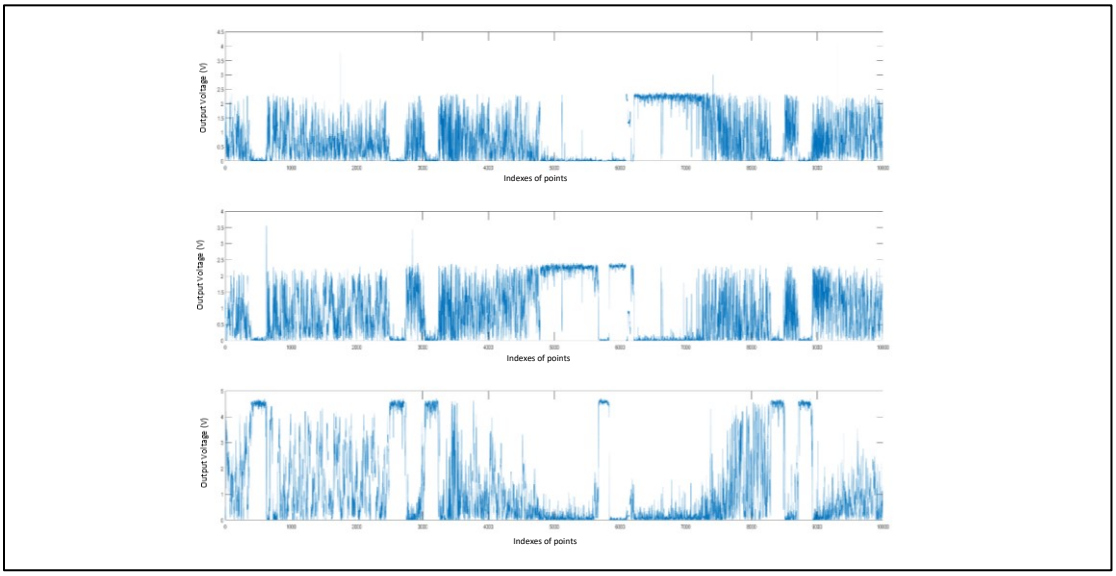


Figure 5.26 Details of the 2D plot.

Also, we find that the feedback voltage is 3V, which is lower than the expected voltage 5V, from figure 5.23 (b). After connecting a voltage divider with ratio 2/3 into the loop for increasing the feedback voltage, the actual feedback voltage is still lower than 5V. Figure 5.27 shows a comparison between the actual outputs and the computer

simulation. We find that although the points are limited by the feedback loop in a plane, they are not on the plane 1=x+y+z accurately. The error is caused by the lower feedback voltage, as mentioned above.

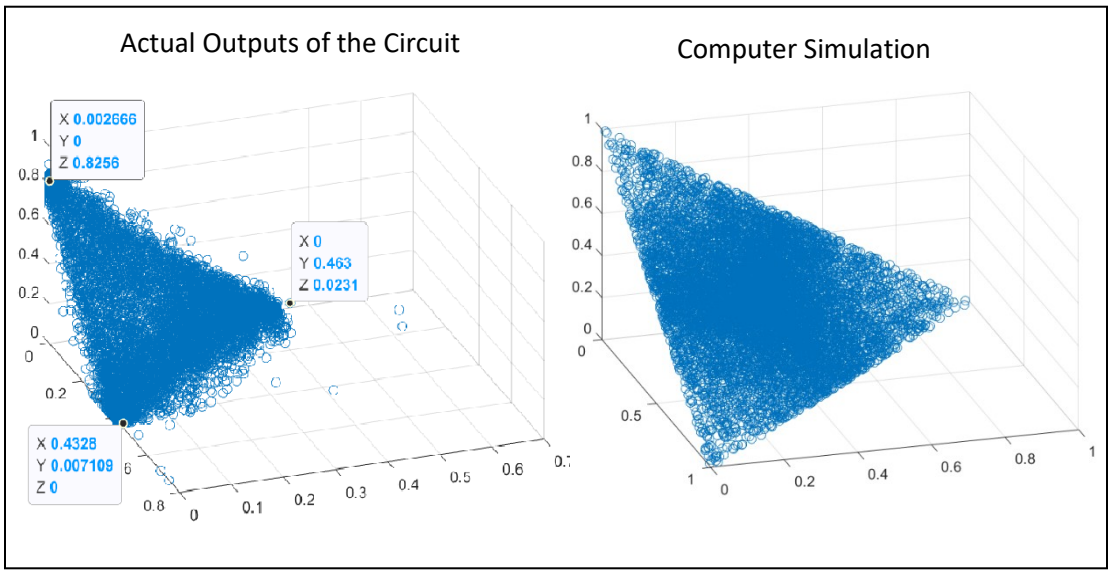


Figure 5.27 Comparing the actual outputs with computer simulation.

Chapter 6

Improvements

6.1. Signal Amplifier for Image Processing

6.1.1. Test Introduction

The test implemented in this section is to intensify image processing capability of the system. For high resolution images, processing speed of the system and resolution of outputs become important. We replace the Arduino Mega with Raspberry Pi 3B with a MCP3008 to increase capture speed of the system. Meanwhile, we add the 'SQNR intensify circuit' to the output of the system for increasing the resolution of the images (its schematic is in figure Appendix C.10).

6.1.2. Test Bench and Test Method

The Raspberry Pi +MCP3008 ADC are used for signal capturing. The test bench is shown in table 6.1 (its actual circuit implementation is in figure Appendix D.11).

Table 6.1 Raspberry Pi + MCP 3008 test bench.

Device for tests	Measurement devices	Measurement software
1. 1*Multiply-accumulation system 2. ADC's SQNR intensify circuit	Raspberry Pi +MCP3008	MATLAB

First, we test the system processing speed by measuring an analog signal for one time and using the 'tic-toc' function for speed measuring. Next, we apply the same image filter for the system without the SQNR intensify circuit followed by the system with the SQNR intensify circuit.

6.1.3. Test Result

Below are signal capturing speeds comparison (table 6.2). Also, image resolution comparison (figure 6.1) of adding SQNR intensify circuit. Figure 6.1 (a) and (c) are outputs without the intensify circuit which has a swap range of 1V and figure 6.1 (b) and (d) are outputs with the intensify circuit which has a swap range of 4.5V.

Table 6.2 Measuring speed comparison.

Measurement device	Time cost	Time cost for 404*388 image
ADCs on Arduino Mega+Signal monitoring board	0.027698s	36 h
Raspberry Pi 3B +MCP3008	0.000013s	4h

Figure 6.1 a) 'Laplacian operator' filter processing result without analog amplify b) 'Laplacian operator' filter processing result with analog amplify c) 'Sobel' filter processing result without analog amplify d)) 'Sobel' filter processing result with analog amplify.

As shown in figure 6.1, the image has less noise and clear edges than the images processed by the system without the op-amp amplifier. Therefore, the image quality is increased by the op-amp amplifier.

6.1.4. Conclusion

As shown in table 6.2, the measurement device Raspberry Pi 3B with MCP3008 is much faster than the ADCs of ATmega2560. The reasons are: firstly, the Raspberry Pi communicates with the computer through ethernet cable which is faster than the UART bus. Secondly, the Cortex A53 processor has lower response delay. Thirdly, the MCP3008 chips have higher performances than the integrated ADCs of the ATmega 2560. Next, we compare resolution of the output image. Under the same image processing filter, the SQNR intensify circuit increases resolution of the image by maximizing the swapping ranges of signals.

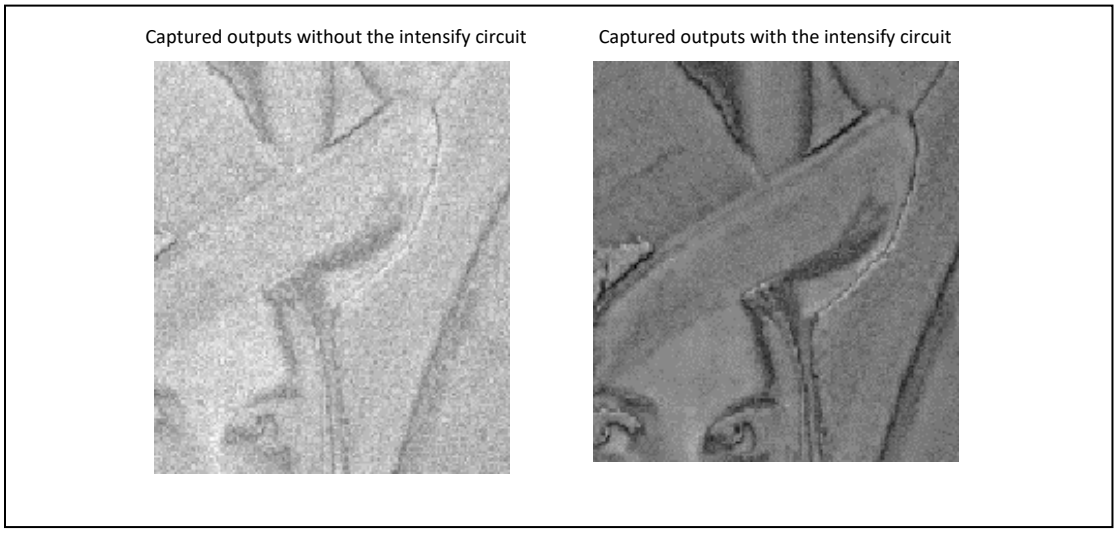


Figure 6.2 Resolution comparison of processed images.

The comparison of figure 6.2 shows that the circuit can increase the image resolution by maximizing swapping ranges of the signals.

6.2. System Expansion

6.2.1. Test Introduction

Computing and artificial intelligence use of the system requires more processing units for memorizing more convolution kernels. The system is a 'computing-in-memory' system that can be expanded by adding addressing in the control node. In this test, we add another system to construct a 6*6 convolution system to enable memorizing more complicated patterns for recognition.

6.2.2. Test Bench and Test Method

The test bench is connected as in table 6.3 and figure 6.3. Instead of an address range of 1~18, the system now has address 1~36 for the DAC array and the resistors array. The result of convolution is read by the MATLAB through the Arduino Mega development boards.

Table 6.3 Test and measuring devices.

Device for tests	Measurement devices	Measurement software
2*Multiply-accumulation system	Arduino Mega	MATLAB

The two systems were connected in parallel. We combine the analog multiply-

accumulation cores of the two systems and then implement tests for the large core. The 36 multiply units of two systems can construct convolutional kernels with different sizes as the table 6.4 shows. For demonstration, we tried to save number patterns to the 6*6 convolution kernel. First, we create number patterns 1->9 as matrices in MATLAB. Next, MATLAB programs the 36 digital potentiometer through the firmware. Finally, we read the patterns' contents through setting the DAC arrays to all 5v and recording the output voltage arrays through the Arduino Mega (its actual circuit implementation is in figure Appendix D.12).

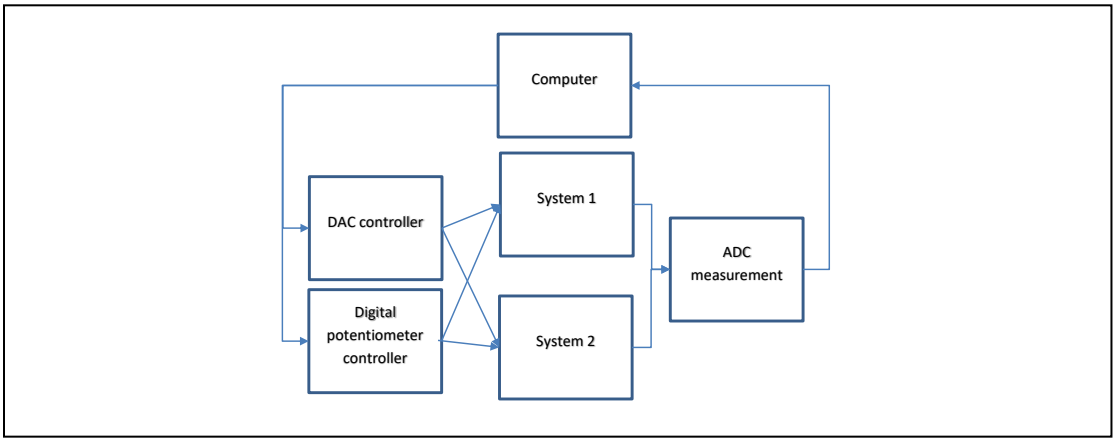


Figure 6.3 System diagram of expanding a 2*3*3 convolution kernel system to a 6*6 convolution kernel system.

Table 6.4 Possible convolution kernels that can be implemented by the 6*6 convolutional kernels

Single convolutional kernel	3*3
	4*4
	5*5
	6*6
Multiple convolutional kernel	4*3*3
	9*2*2
Different sizes convolutional kernel	2*2+4*4

6.2.3. Test Results

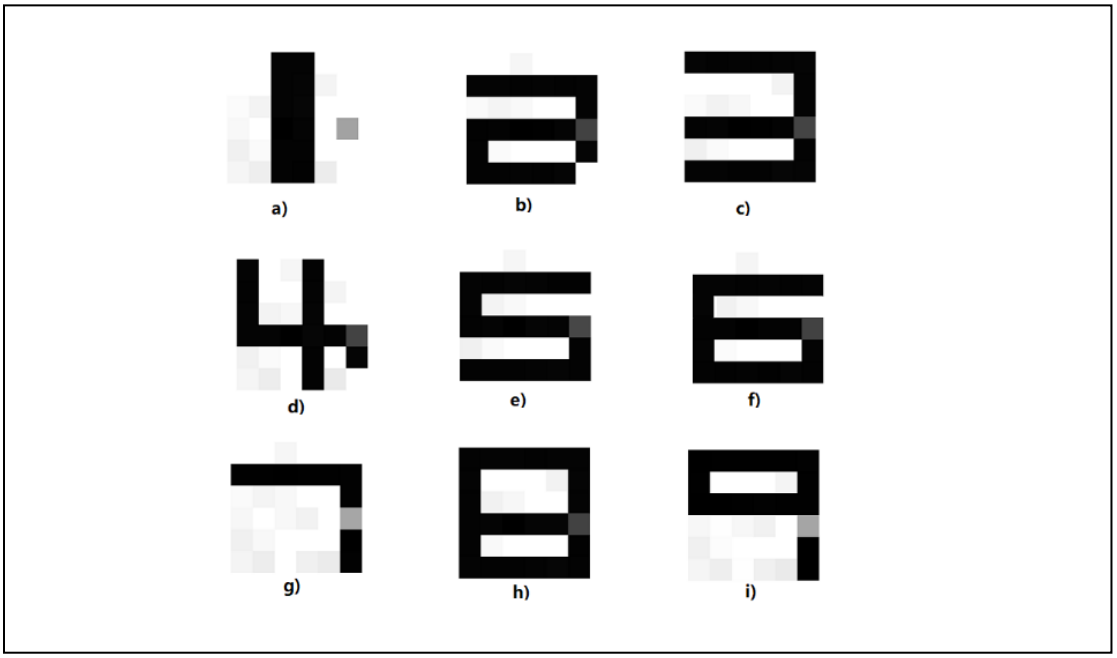


Figure 6.4 Patterns in 6*6 matrix array a) ‘1’ b) ‘2’ c) ‘3’ d) ‘4’ e) ‘5’ f) ‘6’ g) ‘7’ h) ‘8’ i) ‘9’.

Figure 6.4 shows that a larger multiply-accumulation core with 36 units can be constructed by two multiply-accumulation systems. The 6*6 convolutional kernel can memorize more details than a single 3*3 convolutional kernel.

6.2.4. Conclusion

In this section, the test shows expanding the capability of the system by parallel connection of two systems with an extra address line. Theoretically, the system can be expanded infinitely if the addresses are sufficient, and the signals are not attenuated. A larger digital potentiometer matrix is constructed by four 3*3 matrices, and patterns of numbers 1->9 can be memorized in it.

6.3. Parallel Processing in the Frequency Domain

6.3.1. Test Introduction

This idea comes from a research paper "Scalable massively parallel computing using continuous-time data representation in nanoscale crossbar array" from 'Nature nanotechnology' [51]. Multiple signals can overlap each other after modulation in the frequency domain. One voltage divider of the analog multiply-accumulation system can process infinite numbers of signals simultaneously under ideal conditions. The theory of section 2 shows that an arbitrary signal can be expanded to a series of waves with different frequencies.

We can recall the relative equations (eqn 6.1, eqn 6.2, and eqn 6.3),

$$\mathrm{x(n)} = \mathrm{A} * \mathrm{e}^{\mathrm{jwn}} \qquad -\infty < \mathrm{n} < \infty \tag{6.1}$$

$$y(n) = \sum_{k=-\infty}^{+\infty} h(k) * A * e^{jw(n-k)} = A * [\sum_{k=-\infty}^{+\infty} h(k) * e^{-jwk}\ ] * e^{jwn} \tag{6.2}$$

$$H(e^{jw}) = \sum_{k=-\infty}^{+\infty} h(k) * e^{-jwk} \tag{6.3}$$

Vice versa, an arbitrary signal can be constructed by a series of known signals in the frequency domain. For example, assuming a, b, c, d are known parameters with frequency w_0, $2*w_0$, $3*w_0$, $4*w_0$, equation 6.4 shows their frequency responses.

$$\begin{cases} H(e^{j0}) = 0 \\ H(e^{jw_o}) = a * \delta(w - w_o) \\ H(e^{j*2*w_o}) = b * \delta(w - 2 * w_o) \\ H(e^{j*3*w_o}) = c * \delta(w - 3 * w_o) \\ H(e^{j*4*w_o}) = d * \delta(w - 4 * w_o) \end{cases} \tag{6.4}$$

Where H() represent frequency response, δ is impulse function. Sum of the above signals is equation 6.5.

$$H(e^{jw}) = \sum_{k=1}^{4} h(k) * e^{-jwk}$$

$$= a * \delta(w - w_o) + b * \delta(w - 2 * w_o) + c * \delta(w - 3 * w_o) + d * \delta(w - 4 * w_o) \quad \textbf{(6.5)}$$

By comparing equation 6.5 with equation 6.3. h(k) can be calculated,

$$\begin{cases} h(1) = a * e^{jw_o} \\ h(2) = b * e^{j2w_o} \\ h(3) = c * e^{j3w_o} \\ h(4) = d * e^{j4w_o} \end{cases} \quad \textbf{(6.6)}$$

According to the h(n), their real part y(n) is equation 6.7,

$$y(n) = a * \cos(w_o * n) + b * \cos(2 * w_o * n) + c * \cos(3 * w_o * n) + d * \cos(4 * w_o * n) \quad \textbf{(6.7)}$$

Next, we assume the signal y(n) passed through a voltage divider with ratio w. The signal becomes equation 6.8,

$$y'(n) = y(n) * w$$

$$= w * a * \cos(w_o * n) + w * b * \cos(2 * w_o * n) + w * c * \cos(3 * w_o * n) + w * d * \cos(4 * w_o * n)$$

(6.8)

From equation 6.8, a group of new h(n) can be defined.

$$\begin{cases} h'(1) = w * a * e^{jw_o} \\ h'(2) = w * b * e^{j2w_o} \\ h'(3) = w * c * e^{j3w_o} \\ h'(4) = w * d * e^{j4w_o} \end{cases} \quad \textbf{(6.9)}$$

Based on equation 6.9, y'(n) in the frequency domain is equation 6.10,

$$H(e^{jw}) = \sum_{k=0}^{3} h(k) * e^{-jwk}$$

$$= w * a * \delta(w - w_o) + w * b * \delta(w - 2 * w_o) + w * c * \delta(w - 3 * w_o) + w * d * \delta(w - 4 * w_o) \quad \textbf{(6.10)}$$

The above equations show the principles of parallel processing by signal overlapping. First, we modulate required signals and overlap them in the frequency domain. Secondly, the signal is processed in the time domain. Finally, we capture processed signals' amplitudes at corresponding frequencies. We apply frequency overlapping method for a convolutional layer. Convolutional operations for the four sub-matrices (red, orange, green, blue) are based on the same convolutional kernel (figure 6.5).

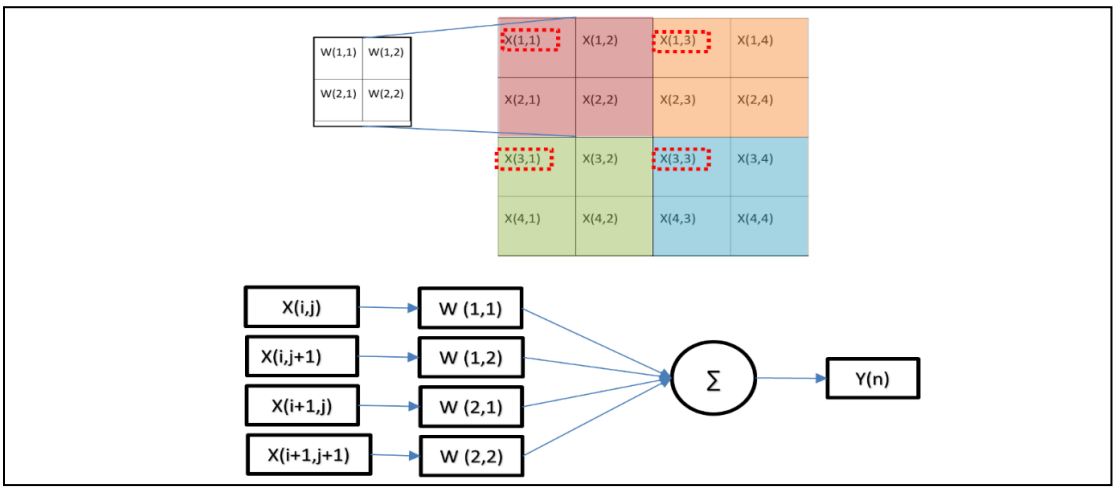


Figure 6.5 Doing convolution for each 2*2 sub-matrices of the input.

We pick convolution operations which relate to w(1,1) for demonstration. Note that x(1,1), x(1,3), x(3,1), x(3,3) are multiplied by w(1,1). Therefore, the four elements are modulated by different frequencies and passing through w(1,1). Then, results x(1,1)* w(1,1), x(1,3) * w(1,1), x(3,1) * w(1,1), x(3,3) * w(1,1) are obtained in the frequency domain (figure 6.6).

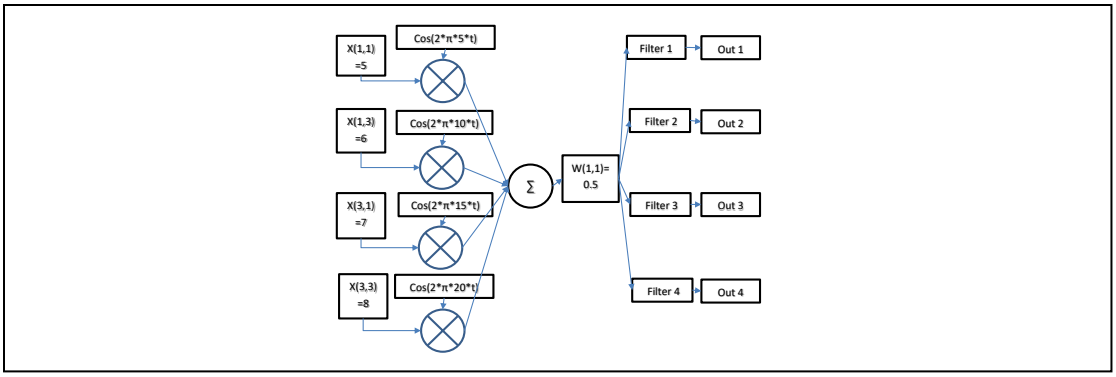


Figure 6.6 Overlapping all inputs which relates W(1,1) by modulation.

6.3.2. Test Method

The test in this section is based on computer simulation and ideal hardware conditions (not considering IIP2, IIP3, and circuit's noise). We will assume x(1,1)=5, x(1,3)=6, x(3,1)=7, x(3,3)=8, w(1,1)=0.5. The simulations and plotting results are done in both the time-domain and frequency domain. Plots are produced of the input signals and output signals in the time domain.

6.3.3. Test Result

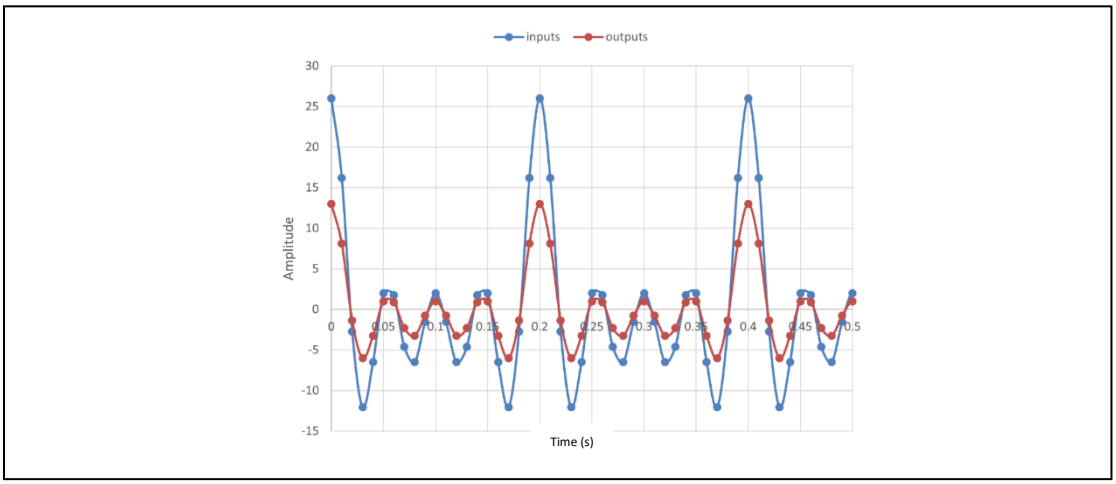


Figure 6.7 The overlapped signal in the time domain.

Figure 6.7 shows plots of the input signals in the time domain. Also, the outputs are checked by plotting the input signals and output signals in the frequency domain.

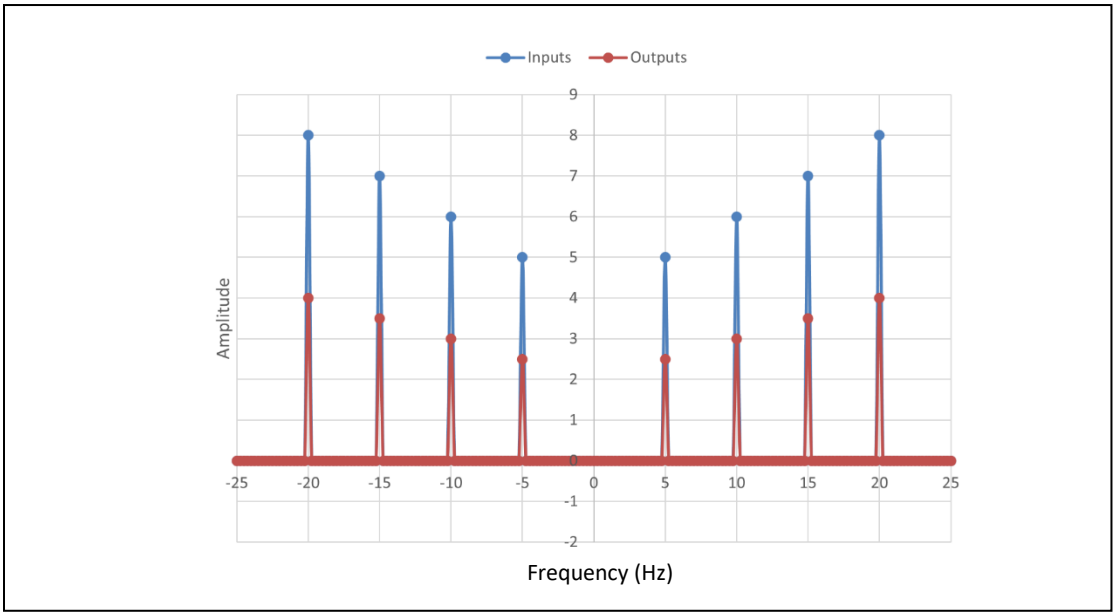


Figure 6.8 The overlapped signal in the frequency domain.

Figures 6.8 shows that the four signals are processed by the voltage divider simultaneously in the time domain. The figure of the frequency domain shows that amplitudes of inputs are two times that of the outputs. Next, we apply 3 orders Butterworth filters for separating the signals, band width=2* π . Figure 6.9 shows filtering results. We can find that the signals' amplitudes are affected by the filters. For getting more accurate outputs, the filters should be high-orders and high-linearity.

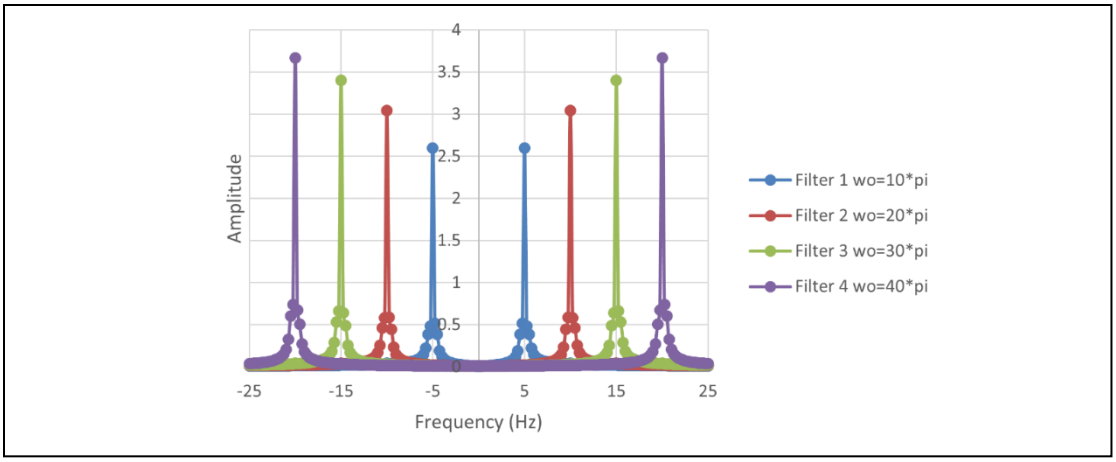


Figure 6.9 Separating output signals by filters.

6.3.4. Conclusion

This section shows a method for parallel convolution processing by overlapping signals in the frequency domain, processing by voltage dividers in the time domain, and separating outputs in the frequency domain. Under ideal conditions, the number of overlapped signals can be infinite. In practice, the numbers of overlapped signals are limited by devices' linearity, noise, and filter performance.

Chapter 7

Discussions

7.1. Analog Multiply-Accumulation Computing Units for Feedforward Architecture

7.1.1. DSP Algorithm Implementations

From the above tests, specific algorithms are possible to be implemented by analog components at the circuit level. Convolution operation is the major algorithm being investigated in the tests of this research. Instead of conventical logic-level accumulators and multipliers, we use the computations of analog accumulators and multipliers are based on Kirchhoff laws and Ohm's laws which have multiplication phenomenon on resistive voltage dividers and addition phenomenon on current accumulations. Tests of section 4.1 shows a 1*4 resistor array based multiply-accumulation array for computing 1D and 2D convolution. The four digital potentiometers can pre-store parameters of the digital filter and processing images through digital-analog and analog-digital conversions. The images of test results in section 4.10 and 4.11 show that it is possible to design an analog circuit-level device for implementing DSP filter algorithms directly.

7.1.2. Current Algorithms of Computer Science Implementation

In section 4.2, we tried to expand the 1*4 multiply-accumulation array to a 2*3*3

multiply-accumulation matrix so that it is compatible with current algorithms of computer science, such as 3*3 image filters of computer vision and artificial intelligence. The system also has self-monitoring module and firmware module for creating an interface between computers and the analog systems. In the tests of section 4.2.2, two sharpening image digital filters Sobel and Laplacian operator filter, and two smoothing filters Mean and Median are transplanted to the analog multiply-accumulation system. The test results show that the system has capability for implementing algorithms of computer science.

Also, we used the system for implementing a simple convolutional neural network which only has a pooling layer, a convolutional layer, and a fully-connected layer. As shown in Table 4.7, the system can implement a convolutional neural network with helps of MATLAB. As shown in the algorithm implementation of section 4.2.3, insufficient multiplying units restrict capability of the system for implementing larger convolutional kernels. Therefore, in section 6.2 we show an example of system expansion that increases the maximum sizes of convolution kernels from 3*6 to 6*6 by the parallel connection of two systems.

7.1.3. DSP, ISP, and CNN Hardware Implementations

Similarities and differences of calculations of DSP, ISP, and CNN were discussed in Table 2.1. The tests of sections 4.1 and 4.2 show the connections of their essences: from perspective of DSP, matrices are filters with specific impulse responses, from perspective of ISP, matrices are also patterns that represent memorized images, from perspective of

CNN, matrices are also pure numbers that they can be processed by mathematical methods.

Although matrices have their own definitions in the different fields, they have similar principles. For example, matrices of DSP algorithms are digital filters, but they are also 1-dimensional patterns of ISP. If we think about section 4.1.2, the matrix [1, -1, 1, -1] can detect edges because its corner frequencies are $\pm\pi/2$, 0, but [1, -1] can be treated as a 1-dimensional pattern of edges. Also, image filters of ISP are patterns, but they can be treated as digital filters with multiple frequency components. If we think about the 'zeros-convolutional kernels conversions' of table 4.1 in section 4.1.2, a specific convolutional kernel represents a filter with specific frequency response. In section 4.2.2, we show larger matrices can express signal filters with more complicated frequency responses. Moreover, convolutional kernels are learnt by training algorithms for increasing correlations, but the learnt matrices are patterns of the fed in images. If we think about figure 4.25, the learnt convolutional kernels are just down-sampled input images.

Therefore, it is possible to use an analog multiply-accumulation system for different digital algorithm implementations.

7.2. Analog Programmable Resistor-Based Feedback loops for Backward Architecture Implementations

7.2.1. Feedback Loops and Computing-in-Memory Units

In sections 5.1, 5.2, and 5.3, a programmable resistor-based feedback loop, a 'Add-division' computing unit, and a 'Random matrix generator' are designed and tested. The three circuits are based on the same architecture: the feedback loop. The test results of figures 5.7 and 5.8 show both computing ability and memorizing capabilities of them. In term of computing capability, the feedback loop can solve w=x/y and save the result in the digital potentiometer with acceptable accuracy. In terms of memorizing capability test, the feedback loop can follow and save voltage matrix of an image. From the test results, we can find that the programmable resistor-based feedback loops can execute calculation and memory on the same device. Therefore, they also can be defined as computing-in-memory devices. Nowadays, most computing systems are based on large scale integrated digital circuits. Algorithm executions require multiple layers from operating system to computing/memory units to logic devices to transistors. Also, buses between computing units and memories restrict their performance. Instead of the architecture with buses and multiple layers, the programmable resistor-based feedback loops have both computing and memorizing functions: the feedback loops adjust the digital potentiometers according to a group of known input and output under self-training mode. The digital potentiometers become multipliers for feed in voltages for the implementing mode.

Based on the characteristics of computing-in-memory, more algorithms can be implemented by combining multiple feedback loops and programmable resistors. The

‘Add-division’ computing unit and the ‘Random matrix generator’ are two examples whose architecture can be named by ‘small feedback loop matrix’ and ‘large feedback loop matrix’. The ‘small feedback loop matrix’ has several independent single resistor feedback loops and the ‘large feedback loop matrix’ has a multiply-accumulation unit in single feedback loop. Test results of the ‘Add-division circuit’ in section 5.2.5 shows that algorithms with divisions can be implemented by the programmable resistor-based feedback loops. Test results of the ‘Random matrix generator’ in section 5.3.5 shows ‘high dimensional feedback loop’. If we think about the calculation of the single resistor feedback loop w=y/x, calculation of this circuit is $[w1 \quad w2 \quad w3] = [y1 \quad y2 \quad y3]./$ $[[x11, x12, x13], \quad [x21, x22, x23], \quad [x31, x32, x33]]$ at 3-dimensional space, which can be called ‘3-dimentional feedback loop’.

7.2.2. Exceeding Classical Computing Systems

Conventional computing systems are based on fixed digital signals, but they ignore random signals that are useful for artificial intelligence. The author of this thesis got an idea during studying for ‘radio frequency system’ class at Carleton University. This was making a computing system based on noise since noise can always be distributed in space uniformly. It can be used for true random number generations which is useful for self-learning of computers. The ‘Random matrix generator’ is designed based on this idea. As shown in section 5.2.5, the output points are ‘controllable random’ which show random characteristics at 2D space but on the same surface at 3D space. Application of this circuit

is to generate some random matrices for computers' self-explorations.

Furthermore, the circuit design of 5.2.5 shows an example of implementing algorithms without actual computing units. As we know, computing units are defined by humans, but brains rely on connections of neurons. This shows that architecture of the resistor-based circuits may closer to the actual brains and feedback loops may exist in connections of actual neurons.

7.3. Programmable Resistor Circuits for AI Accelerations

7.3.1. Computing Density vs. Accuracy

The architecture of programmable resistor-based circuits that mentioned in chapter 4 and chapter 5 can be integrated by using memristors as in figure 1.5. Comparing with the complicated architectures of digital multipliers, memristor matrices are much simpler. However, as shown in section 4.2.6, analog computing shows low accuracy in large matrix calculations, such as the 3*3 image filters of computer vision. Therefore, it is important to consider their target working circumstances. Theoretically, smaller devices mean higher density of integrations. Under volume restrictions, a chip can integrate more memristor multiply units than digital multiply units. Also, as descriptions of the introduction chapter, for near-sensor signal processing implementations, memristor arrays show much higher speed than digital signal processing since all signals are analog and all computing units work in parallel.

7.3.2. Software Defined Hardware and System Expansion

The multiply-accumulation system shows 'software defined hardware' features. MATLAB presets the digital potentiometer matrices through the firmware, but the matrices process images independently. Therefore, the resistor matrices can be programmed as requirements for different usages. Also, data streams are also controlled by MATLAB through the DAC matrix. For example, the DAC matrix converts each 1*4 pixels from images in section 4.1, but it converts each 3*3 pixels from images in section 4.2. Therefore, the computing architecture becomes more flexible with the help of software. Sizes of fed in images and convolutional kernels are not limited.

The test of section 6.2.2 shows a test for system expansion that adding more units to the system through assigning more addresses. Larger convolutional kernels can be constructed by connecting two systems in parallel. Comparing with CPU interconnections of Von Neumann architecture, all the units have the same priority.

7.3.3. Digital-Analog Complementary Signal Processing

The circuit design and tests of section 6.1 shows a digital-analog complementary signal processing method. On the one hand, applying digital signal processing algorithms on resistor matrices leads more flexible filters and simpler hardware. On the other hand, the ADCs' signal noise ratio is improved by the analog amplifier. Therefore, the method of 'analog-digital implementation' makes it possible to take advantages of both sides.

Chapter 8

Conclusion

This thesis talks about two paths of analog hardware algorithm implementations for accelerating algorithm execution speeds. In terms of feedforward architecture, a 1*4 multiply-accumulation array was designed for implementing algorithms of DSP filters. Also, a 2*3*3 multiply-accumulation matrix system was designed for implementing algorithms of computer science through transplant. In terms of backward architecture, a programmable resistor-based feedback loop was designed for computing-in-memory applications. Also, a 'add-division' circuit with 'small feedback loop matrix' was designed for implementing the convolutional kernels training algorithm. Furthermore, a 'random matrix generator' circuit with 'large feedback loop matrix' was designed for implementing the algorithm of Diophantine equation solver.

From the testing we find that digital assisted analog system can accelerate specific linear or non-linear algorithms. Especially, as the artificial intelligence and image processing algorithms are more and more common in people's life, analog accelerating systems will show higher advantages on power consumption, efficiency, and size in mobile devices and edge computing devices. Further, high density integrated memristor based artificial intelligence chips may replace GPUs to be the major accelerating chip for AI.

References

[1] Jaeger, R.C., & Blalock, T.N. “Microelectronic Circuit Design”. New York: McGraw-Hill company, Inc., pp. 226–233. (1997).

[2] Blanchet, G., & Dupouy, B. “Computer Architecture”. London: John Wiley & Sons, Inc., P15. (2012).

[3] Zhu, Q., Li, B., & Yang, D., et al. “A flexible ultrasensitive optoelectronic sensor array for neuromorphic vision systems”. Nature communications. **12**, 1798. (2021).

[4] Liu, Z., Tang, J., Gao, B., et al. “Neural signal analysis with memristor arrays towards high-efficiency brain-machine interfaces”. Nature communications, **11**, 4234. (2020).

[5] Yao, P., Wu, H., Gao, B., et al. “Fully hardware-implemented memristor convolutional neural network”. Nature. **577**, 641-646. (2020).

[6] Sun, Z., Pedretti, G., Bricalli, A., & Lelmini, D. “One-Step regression and classification with cross-point resistive memory arrays”. Science Advances. **6** (5), 2378. (2020).

[7] Cai, F., Correll, J.M., Lee, S.H., Lim, Y., Bothra, V., Zhang, Z., Flynn, M.P., & Lu, W. “A fully integrated reprogrammable memristor-CMOS system for efficient multiply-accumulate operations”. Nature electronics. **2**,290-299. (2019).

[8] Sosutha, S., & Mohana, D. “Heterogeneous parallel computing using CUDA for chemical process”. Procedia Computer Science **47**, 237-246. (2015).

[9] Jouppi, N.P., Young, C.Y., Patil, N., et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit,” in International Symposium on Computer Architecture (ISCA), Toronto, June 2017.

[10] Yang, F., Li, J., Long, Y., et al. “Wafer-scale heterostructured piezoelectric bio-organic thin films”. Science. **373** (6552), 337-342. (2021).

[11] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., pp28. (2006).

[12] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., pp5. (2006).

[13] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., pp12. (2006).

[14] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., pp14. (2006).

[15] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., pp21. (2006).

[16] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p23. (2006).

[17] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”, Upper Saddle

River, NJ: Prentice Hall, Inc., P89. (2006).

[18] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p118. (2006).

[19] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”, Upper Saddle River, NJ: Prentice Hall, Inc., p121. (2006).

[20] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p301. (2006).

[21] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p88. (2006).

[22] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p661. (2006).

[23] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p660. (2006).

[24] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p327. (2006).

[25] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p252. (2006).

[26] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p110. (2006).

[27] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p329. (2006).

[28] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p330-331. (2006).

[29] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p756. (2006).

[30] Proakis, J.G., & Manolakis, D.G. “Digital Signal Processing 4th edition”. Upper Saddle River, NJ: Prentice Hall, Inc., p664. (2006).

[31] Gonzalez, R.C., & Woods, R.E. “Digital Image Processing Third Edition”. Upper Saddle River, NJ: Prentice Hal, Inc., P69. (2002).

[32] Gonzalez, R.C., & Woods, R.E. “Digital Image Processing Third Edition”. Upper Saddle River, NJ: Prentice Hal, Inc., P271. (2002).

[33] Saha, S. “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way”. Towards Data Science [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[34] Venkatesan, R., & Li, B. “Convolutional Neural Networks in Visual Computing: A

Concise Guide", New York: CRC Press Taylor & Francis Group, Inc., pp97-99. (2017).

[35] Ramsundar, B., & Zadeh, R.B. "TensorFlow for Deep Learning". Sebastopol, CA: O'Reilly Media, Inc., P121. (2018).

[36] Valueva, M.V., Nagornov, N.N., Lyakhov, P.A., Valuev, G.V., & Chervyakov, N.I. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". Science Direct. 177, p-234. (2020).

[37] Valueva, M.V., Nagornov, N.N., Lyakhov, P.A., Valuev, G.V., & Chervyakov, N.I. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". Science Direct. 177, p-235. (2020).

[38] Venkatesan, R., & Li, B. "Convolutional Neural Networks in Visual Computing: A Concise Guide", New York: CRC Press Taylor & Francis Group, Inc., pp90. (2017).

[39] "CNN| Introduction to pooling layer". Geaks for Geaks. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

[40] Ramsundar, B., & Zadeh, R.B. "TensorFlow for Deep Learning". Sebastopol, CA: O'Reilly Media, Inc., P82. (2018).

[41] Proakis, J.G., & Manolakis, D.G. "Digital Signal Processing 4th edition". Upper Saddle River, NJ: Prentice Hall, Inc., p66. (2006).

[42] Proakis, J.G., & Manolakis, D.G. "Digital Signal Processing 4th edition", Upper Saddle River, NJ: Prentice Hall, Inc., pp57. (2006).

[43] Sedra, A.S., Smith, K.C. "Microelectronic Circuits". New York: Oxford University press,

Inc., pp925-927. (2004).

[44] “X9C104 Datasheet”, All Datasheet [Online]. Available: https://pdf1.alldatasheet.com/datasheet.pdf/view/609491/INTERSIL/X9C104.html

[45] Sedra, A.S., Smith, K.C. “Microelectronic Circuits”. New York: Oxford University press, Inc., pp75-88. (2004).

[46] Vordos, N., Gkika, D.A., & Bandekas, D.V. “Wheatstone Bridge and Bioengineering”. Engineering Science and Technology Review. **13** (5), 4-6. (2020).

[47] Gonzalez, R.C., & Woods, R.E. “Digital Image Processing Third Edition”. Upper Saddle River, NJ: Prentice Hal, Inc., P182. (2002).

[48] Gonzalez, R.C., & Woods, R.E. “Digital Image Processing Third Edition”. Upper Saddle River, NJ: Prentice Hal, Inc., P174. (2002).

[49] Ramsundar, B., & Zadeh,R.B. “TensorFlow for Deep Learning”. Sebastopol, CA: O'Reilly Media, Inc., Pp50-53. (2018).

[50] Ang, K.H., & Chong, G. “PID Control System Analysis, Design, and Technology”. IEEE Transactions on control systems technology. **13** (4), 559-575. (2005).

[51] Wang, C., Liang, S., Chen, Y., et al. “Scalable massively parallel computing using continuous-time data representation in nanoscale crossbar array”, Nature nanotechnology. **16**, 1079-1085. (2021).

Appendix A

ICs and MCUs Used in Circuit Implementations

Chips used in tests

Chips name	Function
DAC0832	8-bits DAC IC
X9c104	Digital potentiometer
AD620	Instrumental amplifier
74LS154	4-16 decoder
74HC4051	Analog MUX
OP07	Single op-amp IC
LM324	Four op-amp IC
LM339	Four Comparator IC
AD9833	Waveform generator
LM7905	-5V regulator IC
LM7805	+5V regulator IC
MCP23017	I2C GPIO expander
MCP3008	SPI 10-bits ADC

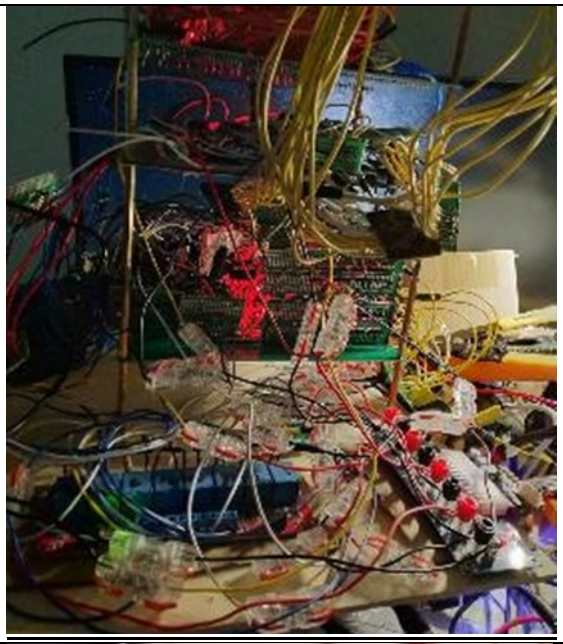
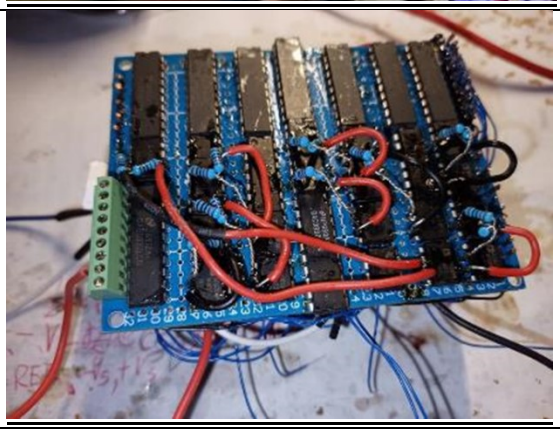
MCU development boards used for test bench:

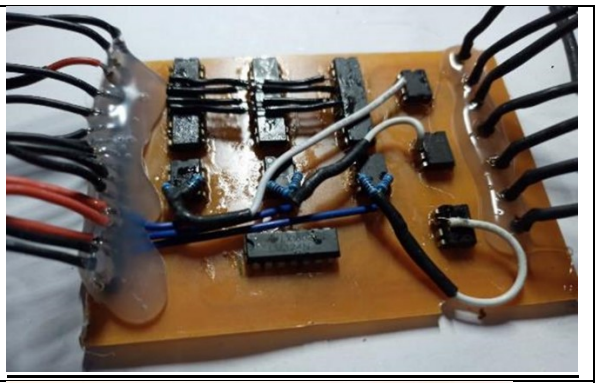
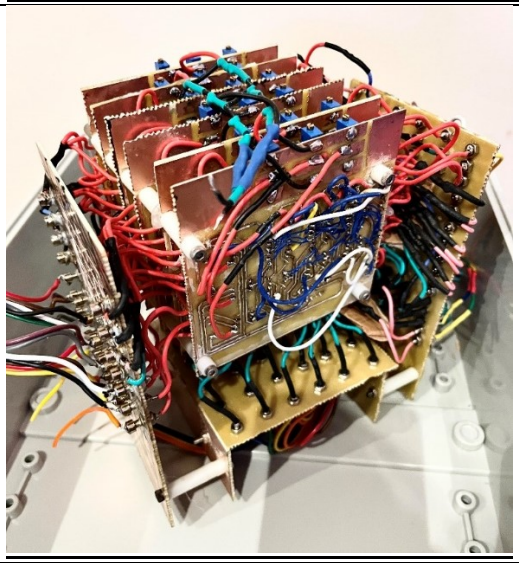
Chips name	MCU Category
STM32F407	Cortex M4
STM32F103	Cortex M3
ATmega2560	AVR
ATmega328	AVR
BCM2837	Cortex A53

Appendix B

Past Versions of the Analog Multiply-Accumulation Core

Past versions:

	Pictures
First version (2020/5~2020/8)	
Second version (2020/9~2020/10)	

Third version (2020/10~2020/12)	
Forth version (2021/1~2021/12)	
Analog multiply-accumulation systems	

Appendix C

Schematics of Circuits

One section of the analog multiply-accumulation core.

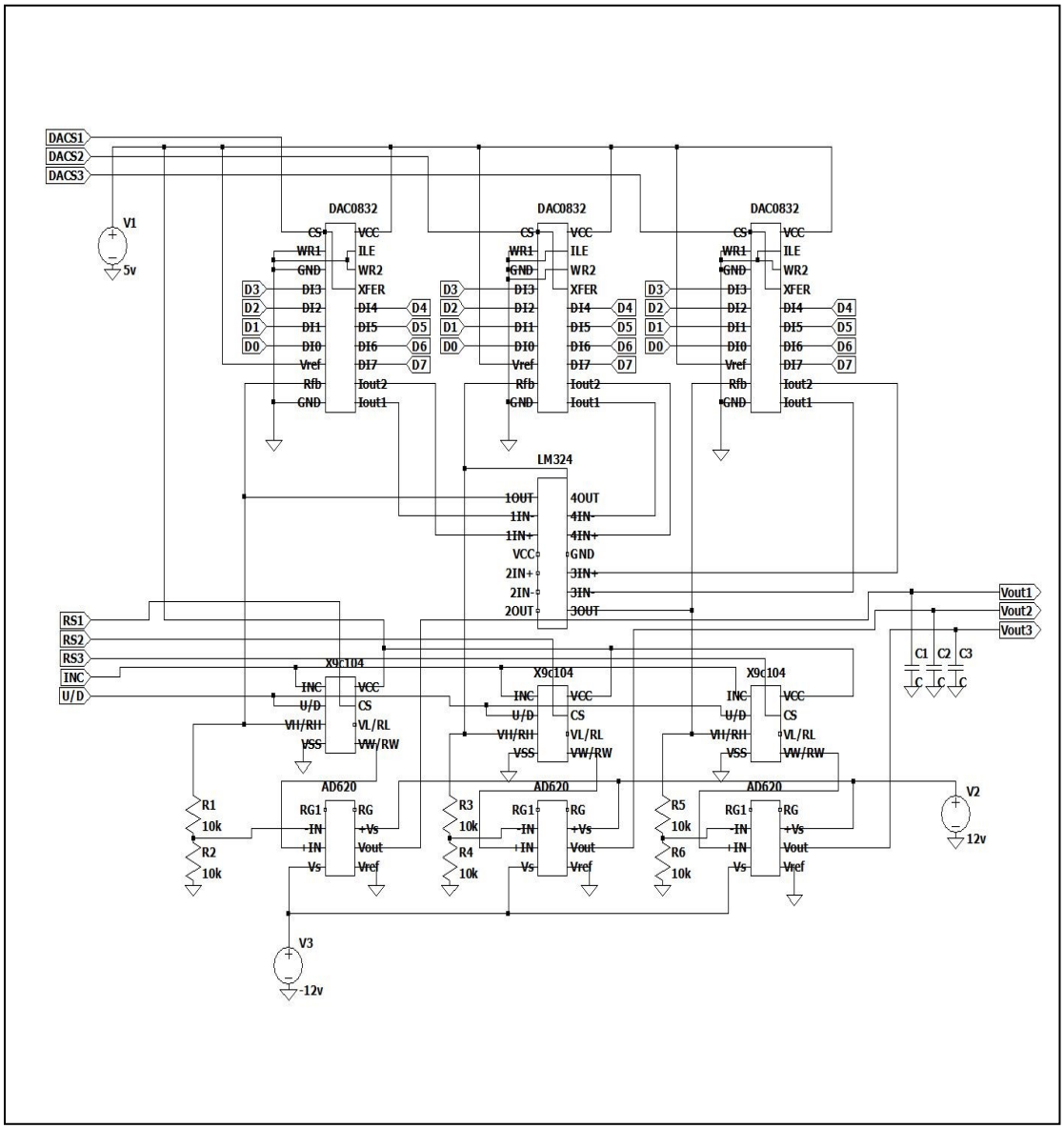


Figure Appendix C.1 Schematic of the analog multiply-accumulation circuit.

A 3*3 analog multiply-accumulation core.

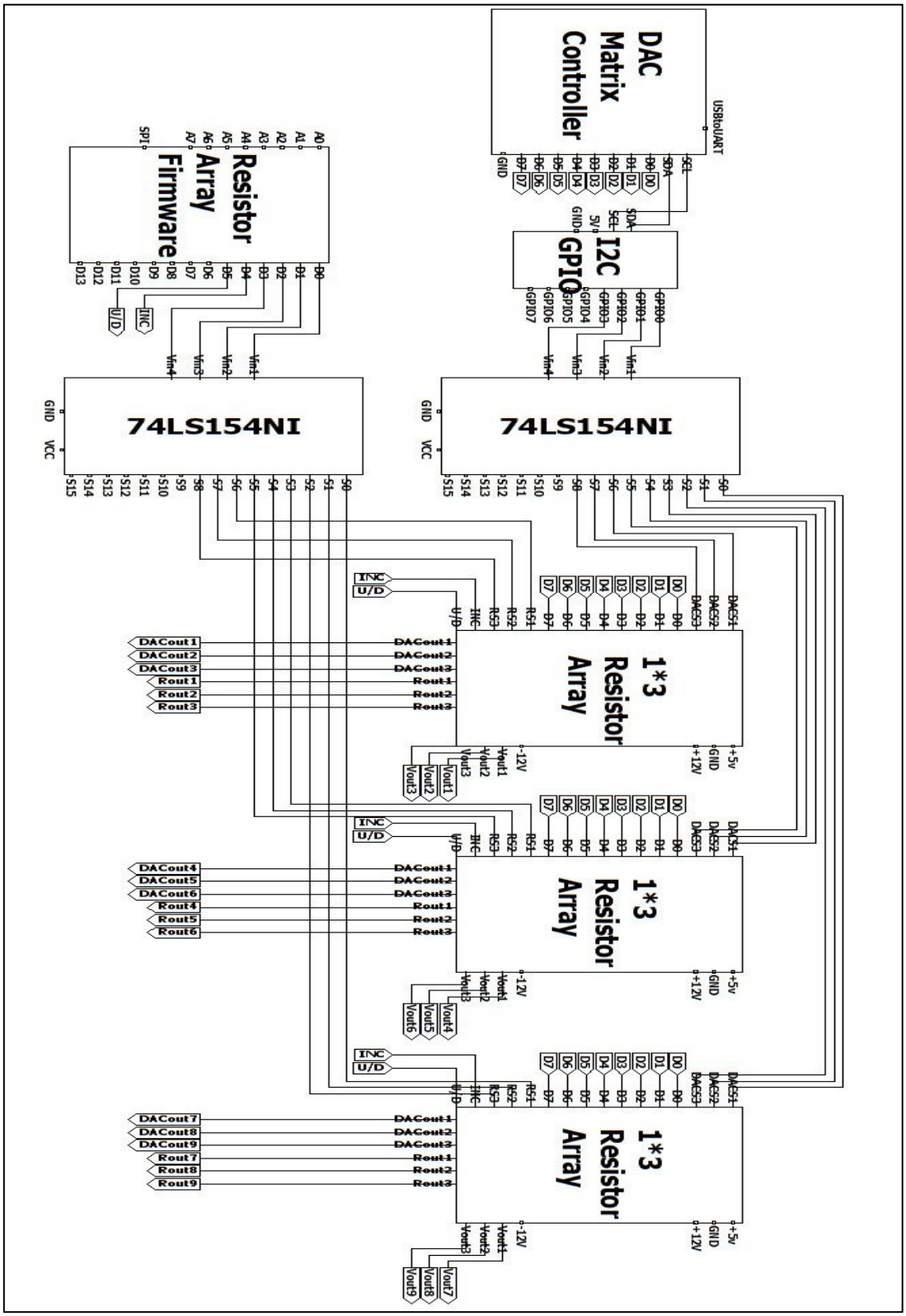


Figure Appendix C.2 Schematic of a 3*3 multiply-accumulation core.

An analog accumulator.

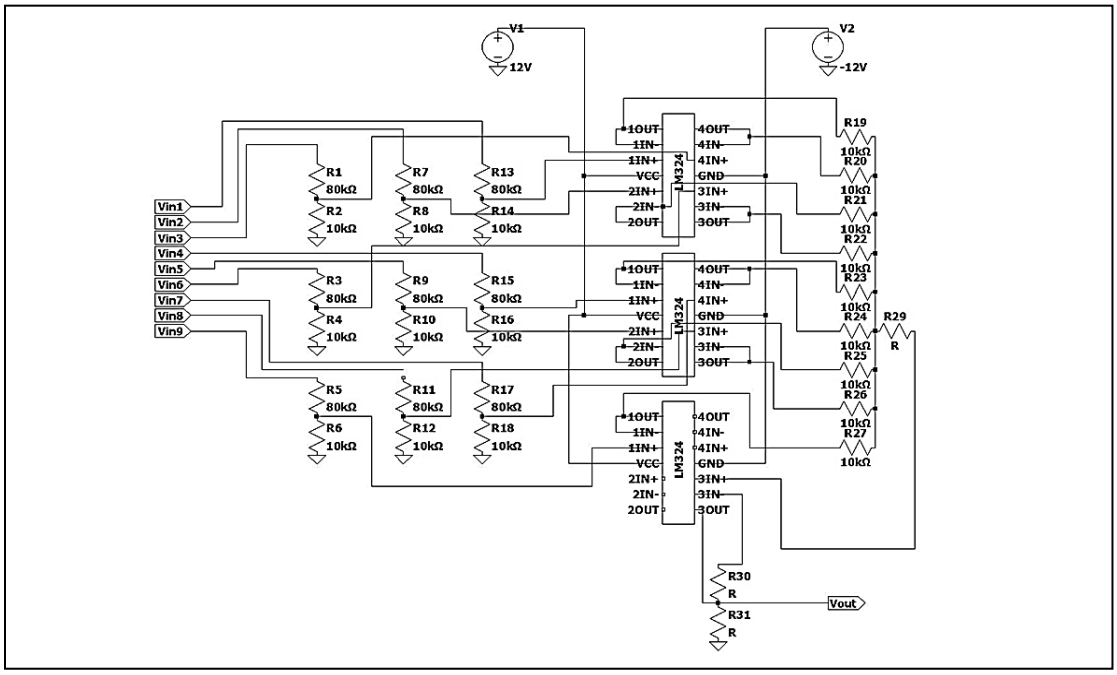


Figure Appendix C.3 Schematic of the analog accumulator.

An analog pooling unit.

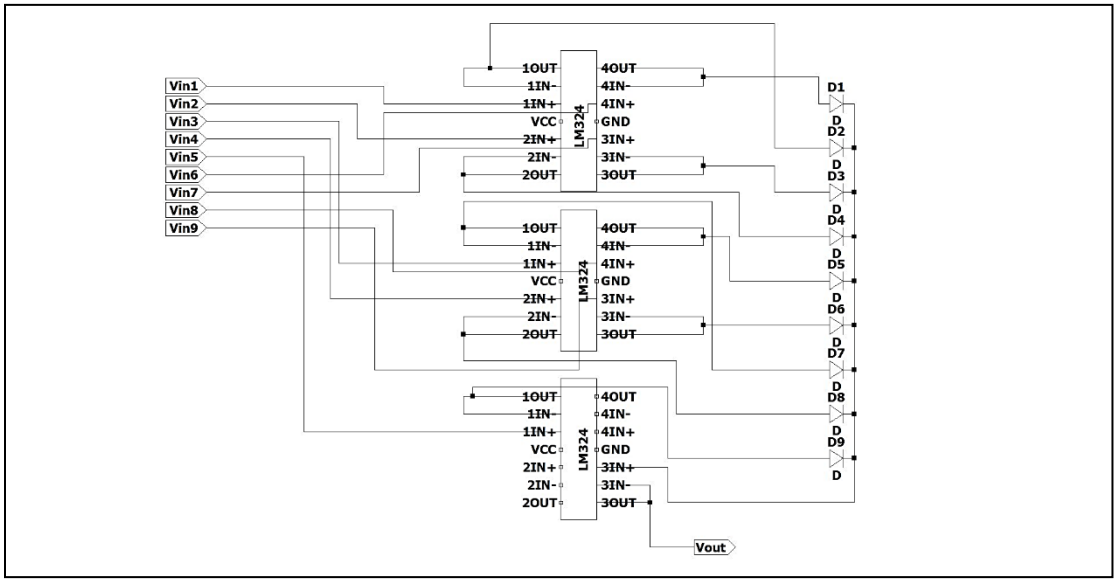


Figure Appendix C.4 Schematic of the pooling circuit.

Signal monitoring circuit.

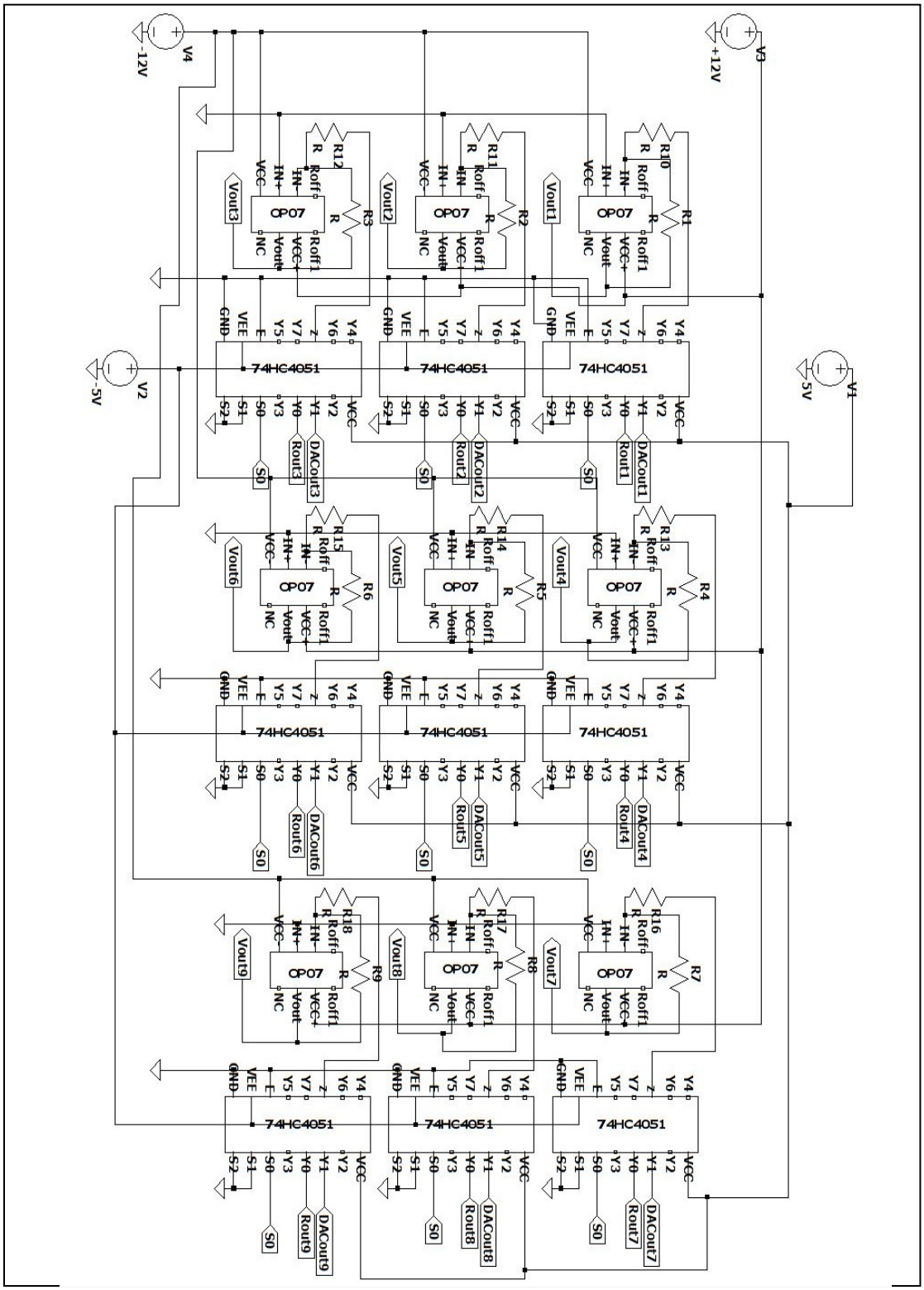


Figure Appendix C.5 Schematic of the signal monitoring circuit.

A firmware for digital potentiometer matrices.

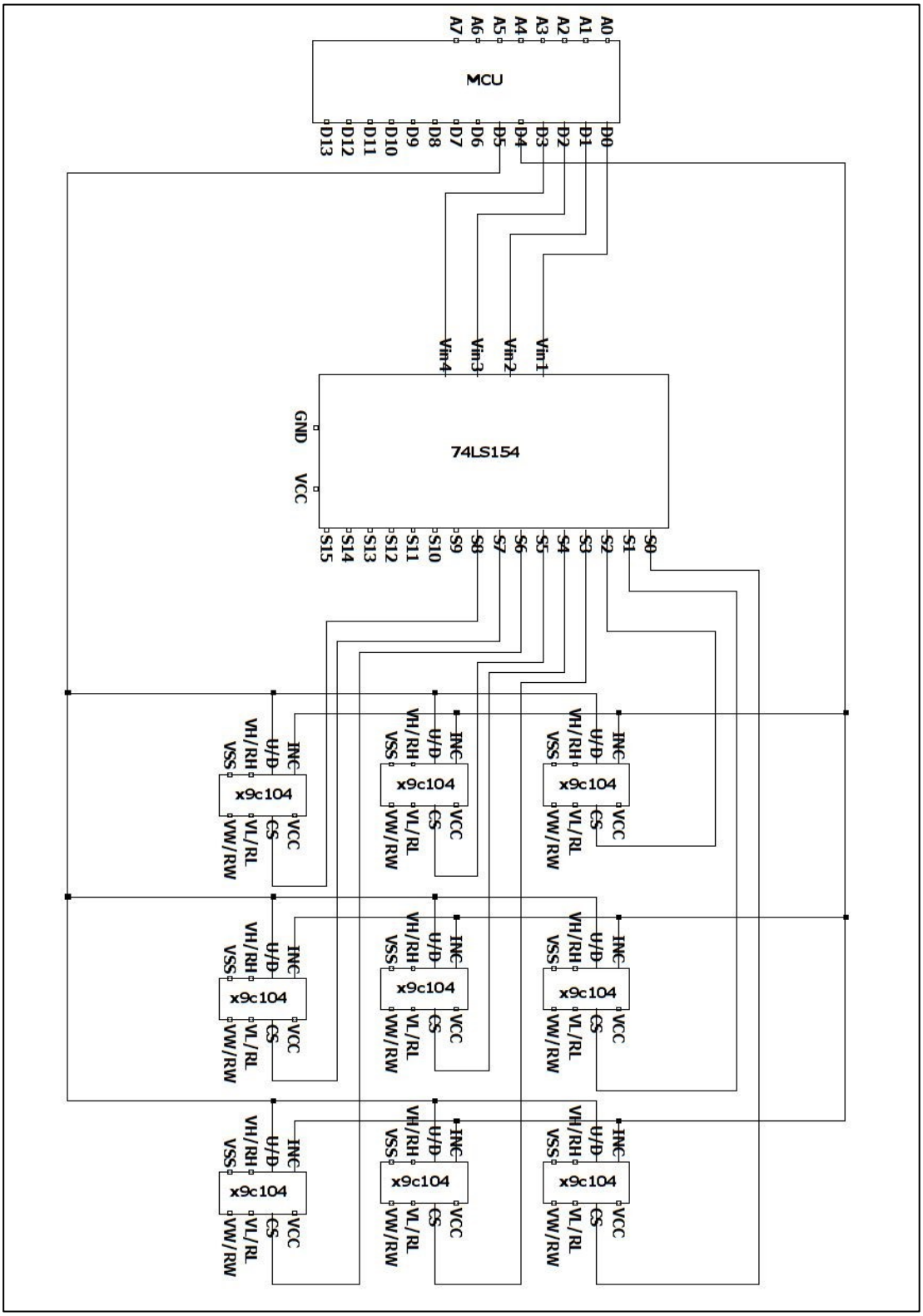


Figure Appendix C.6 Schematic of a MCU resistor array controller.

Programmable resistor-based feedback loop.

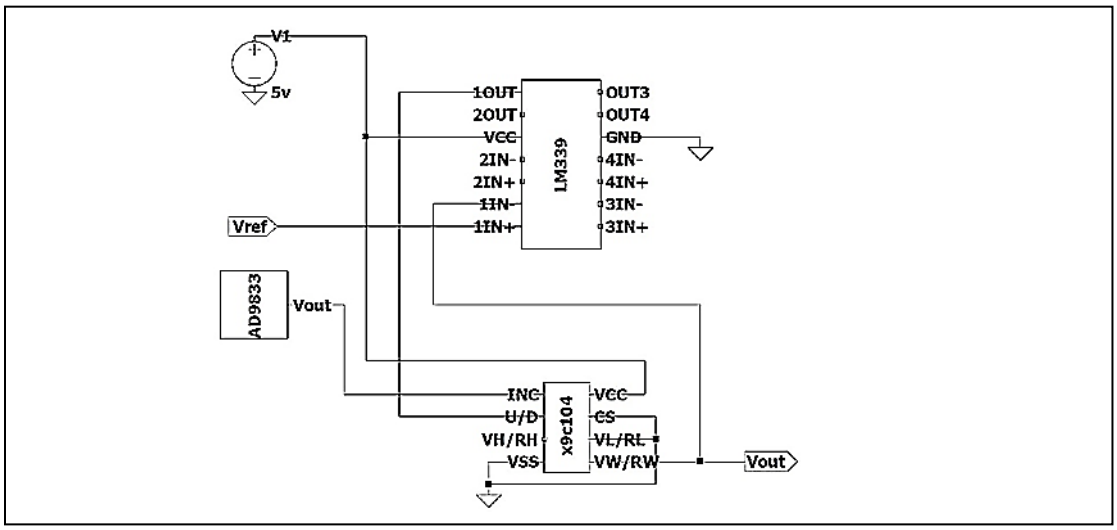


Figure Appendix C.7 Schematic of the programmable resistor-based feedback loop.

Random matrix generator.

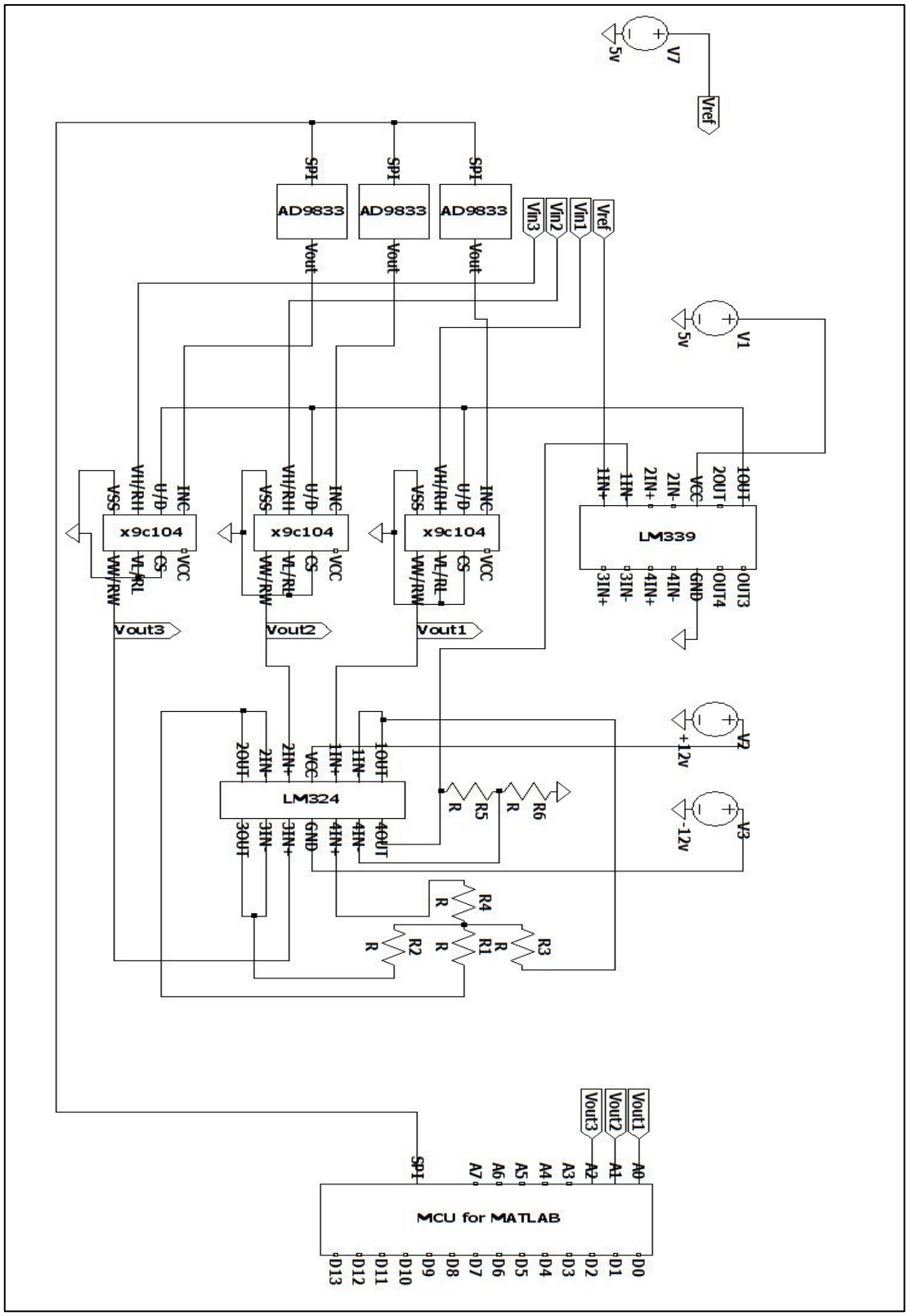


Figure Appendix C.8 Schematic of the random matrix generator.

Add-division circuit.

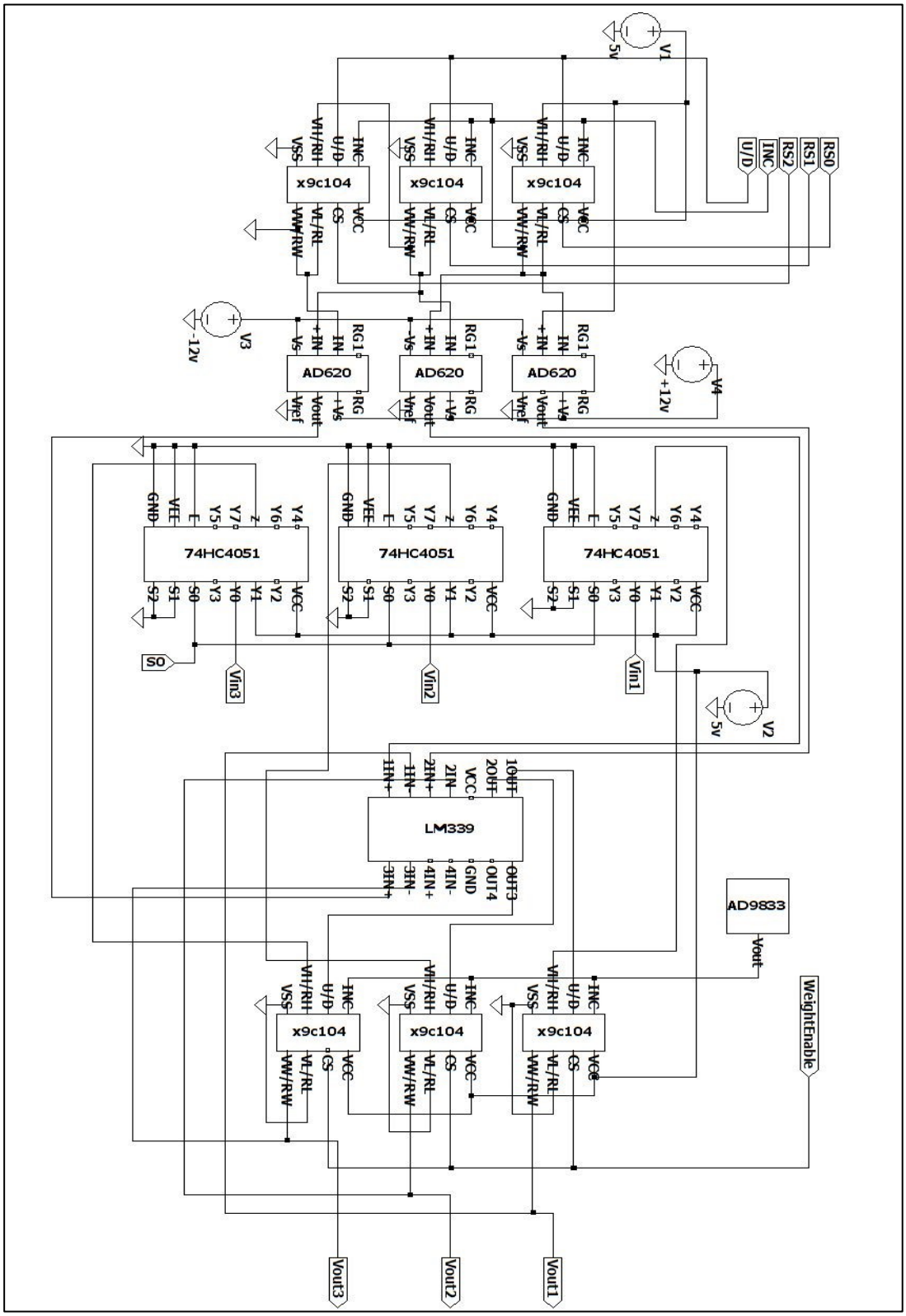


Figure Appendix C.9 Schematic diagram of the add-division circuit.

SQNR optimized circuit.

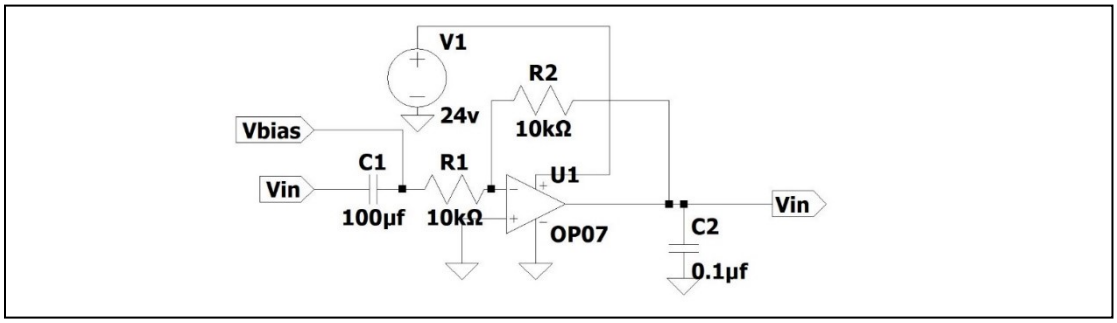


Figure Appendix C.10 Schematic of the Op-amp amplifier with coupling capacitors for increasing SQNR.

System block diagram of the analog multiply-accumulation system.

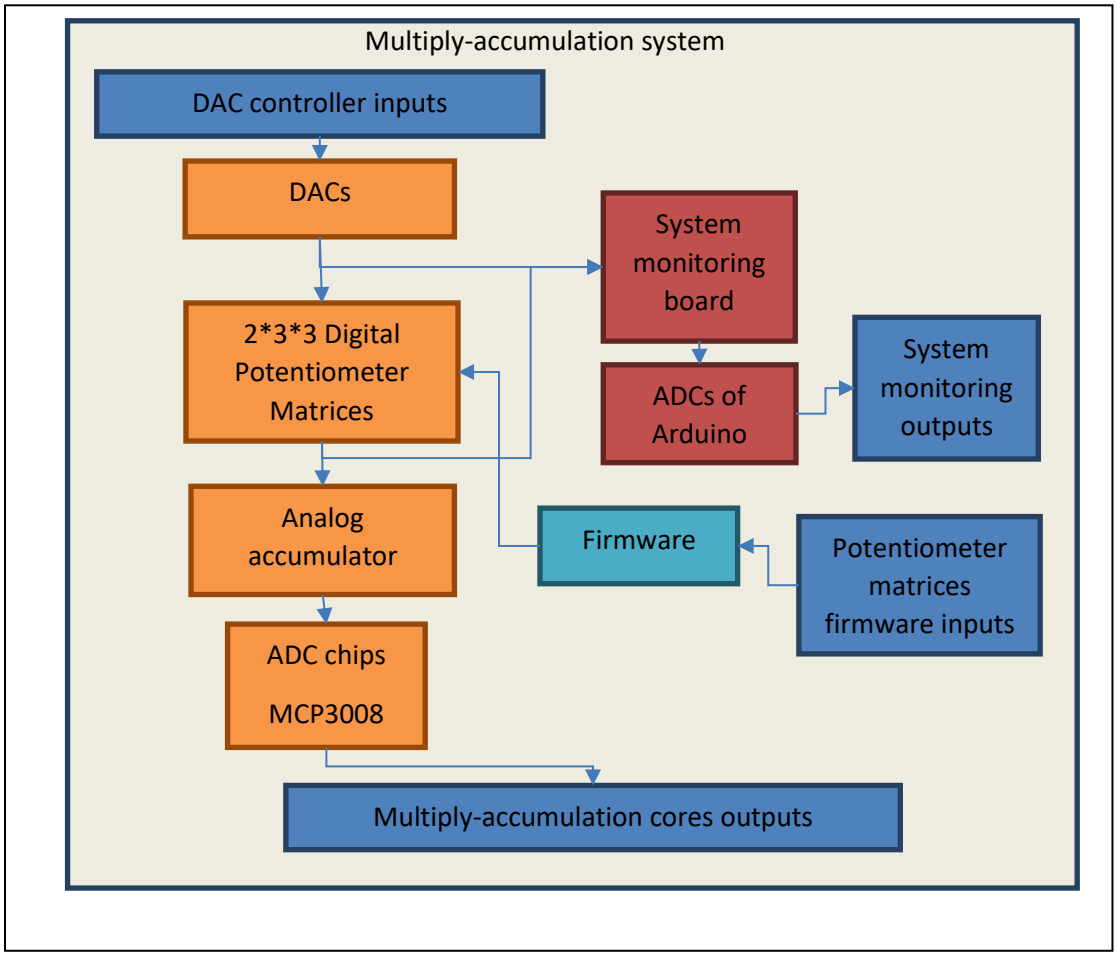


Figure Appendix C.11 Block diagram of a full multiply-accumulation system a) detail of a multiply-accumulation system b) two multiply-accumulation systems with controlling and data collecting devices.

System block diagram of a dual-system.

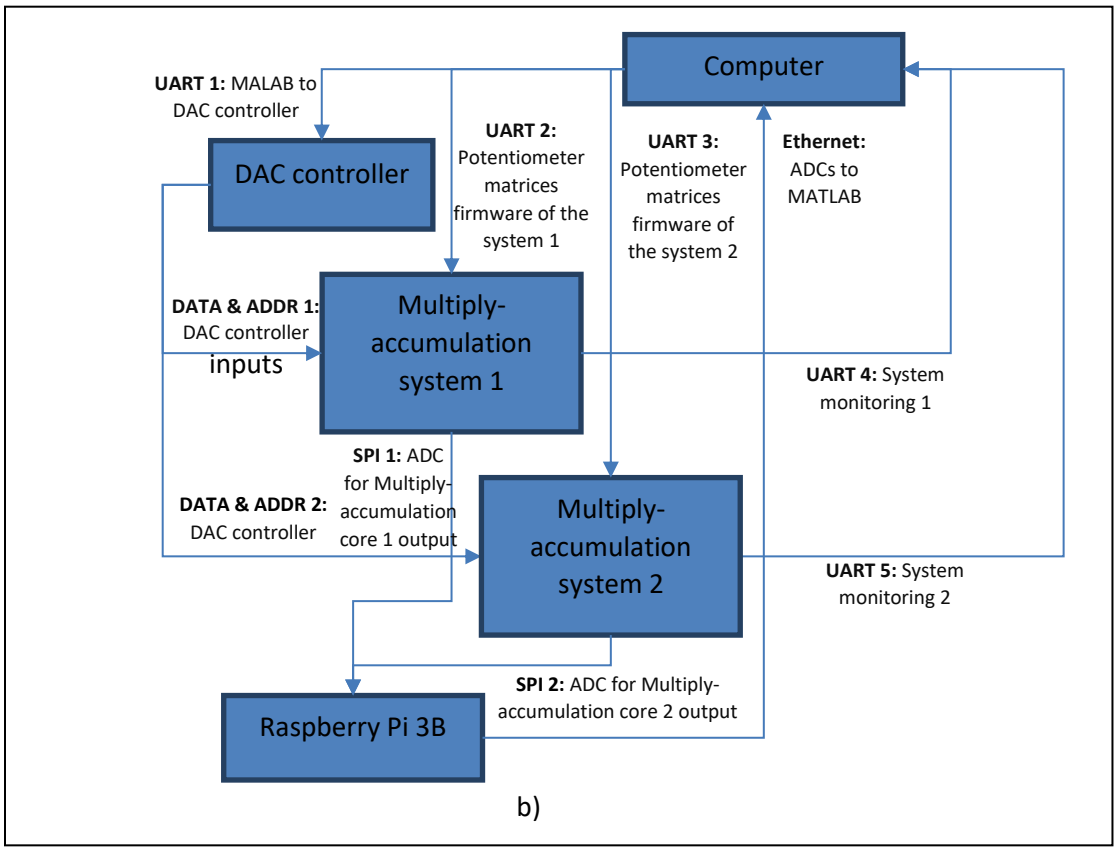


Figure Appendix C.12 Block diagram of a full multiply-accumulation system a) detail of a multiply-accumulation system b) two multiply-accumulation systems with controlling and data collecting devices.

Schematic of the test benches.

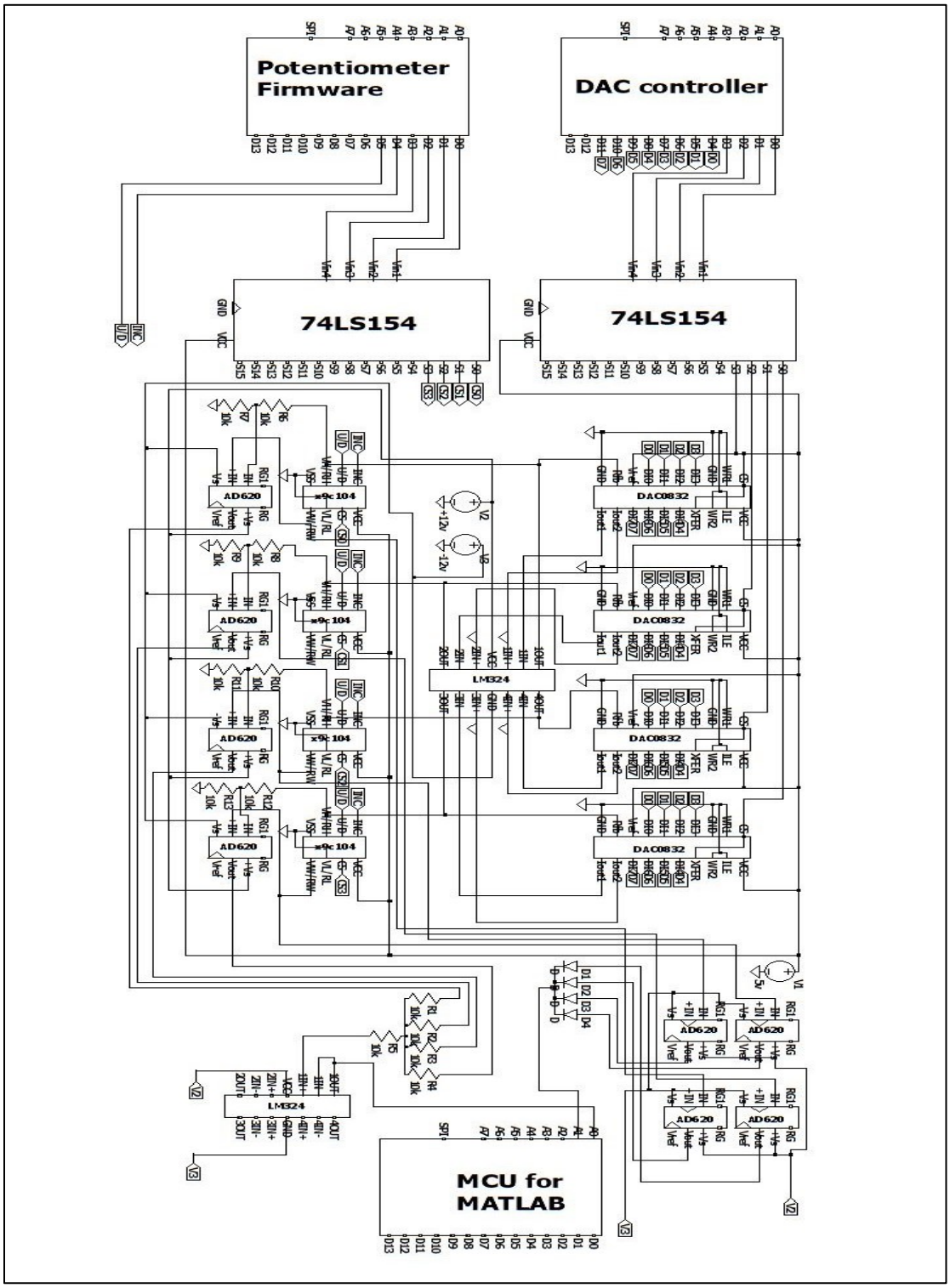


Figure Appendix C.13 Schematic of the test bench for testing analog multiply-accumulation circuits for convolution operations.

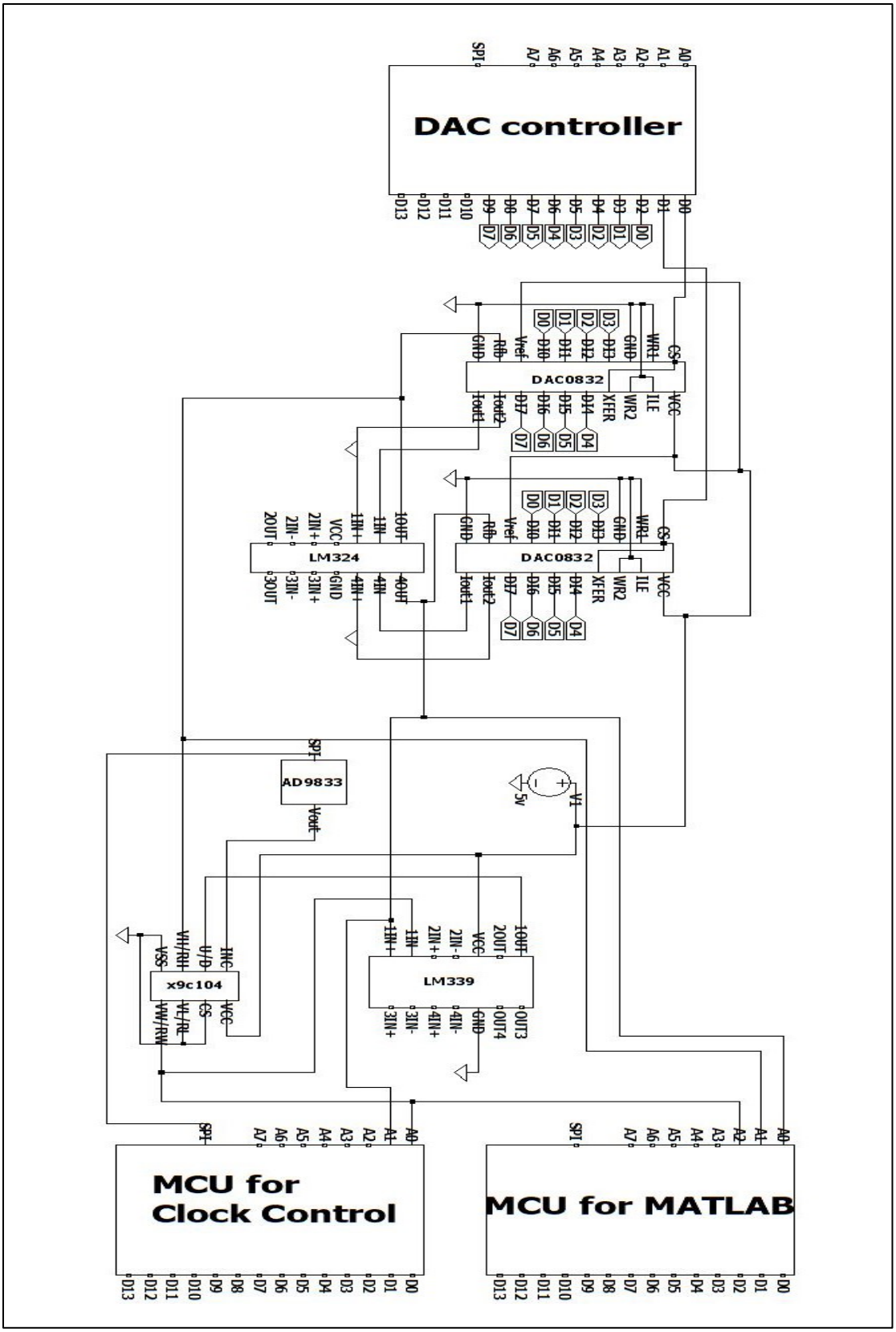


Figure Appendix C.14 Schematic of the testbench for testing the resistor-based feedback loop circuit.

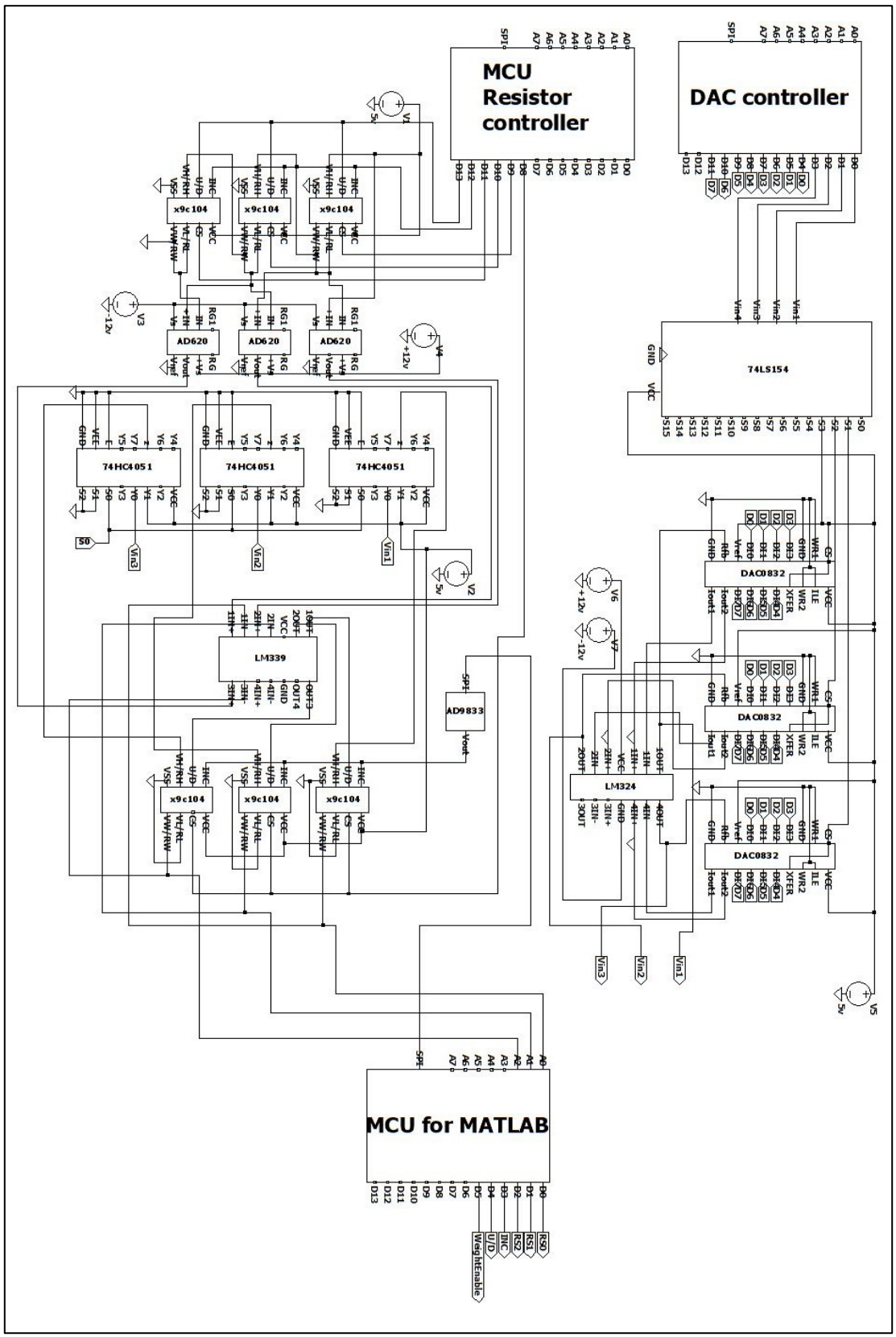


Figure Appendix C.15 Schematic of the test bench for the add-division circuit.

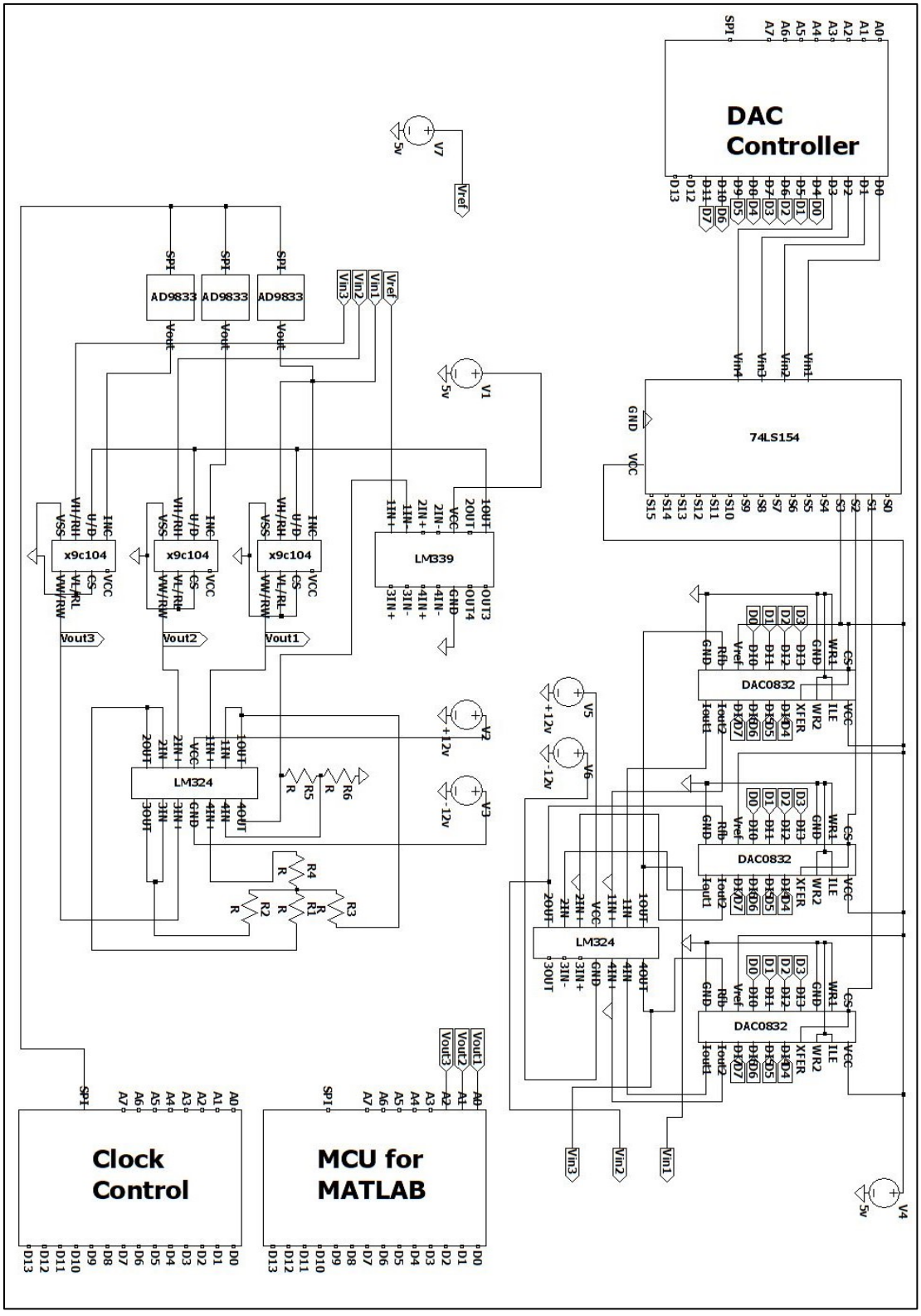


Figure Appendix C.16 Schematic of the test bench for the 'random matrix generator'.

Appendix D

Actual Circuit Implementations

Figure Appendix D.1 Actual DAC matrix controller in the tests.

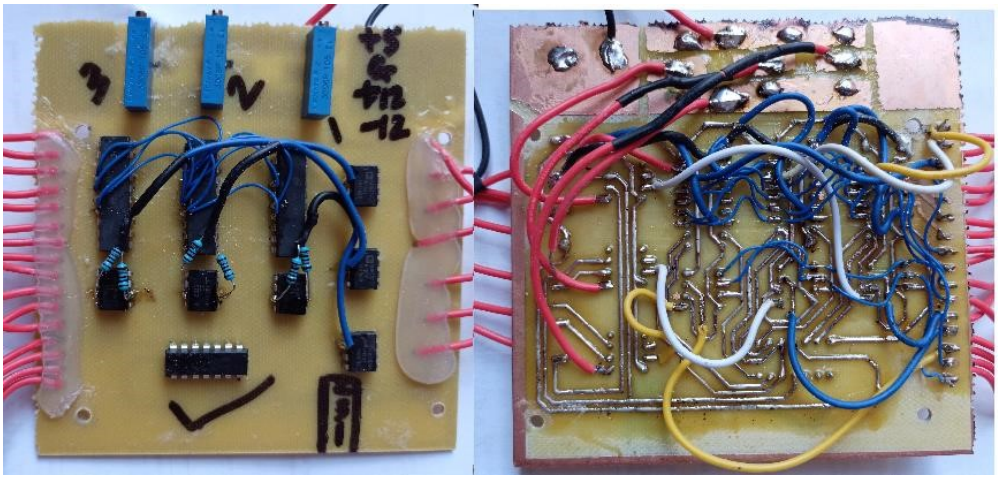


Figure Appendix D.2 Actual PCB with ICs.

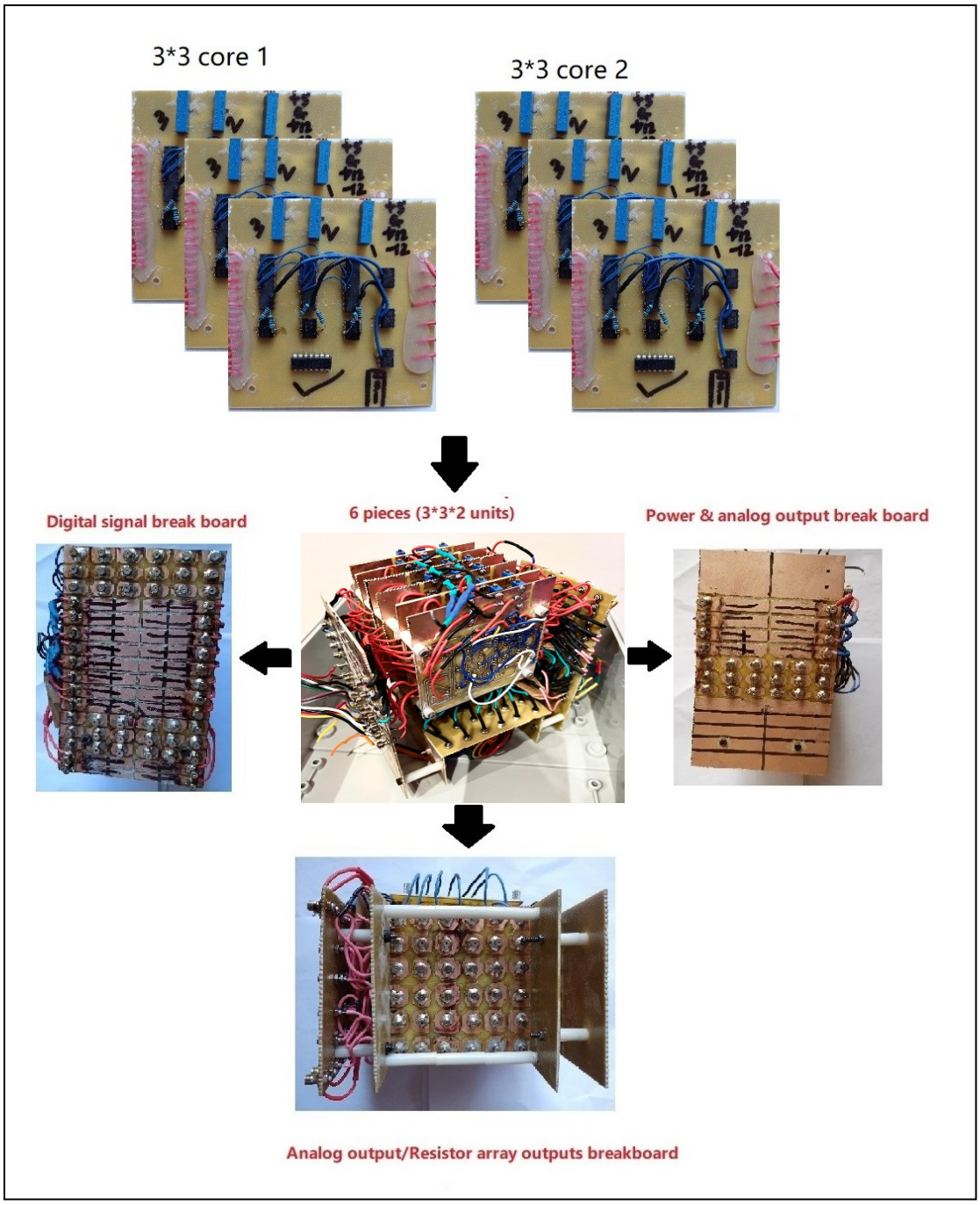


Figure Appendix D.3 An actual 2*3*3 multiply-accumulation core.

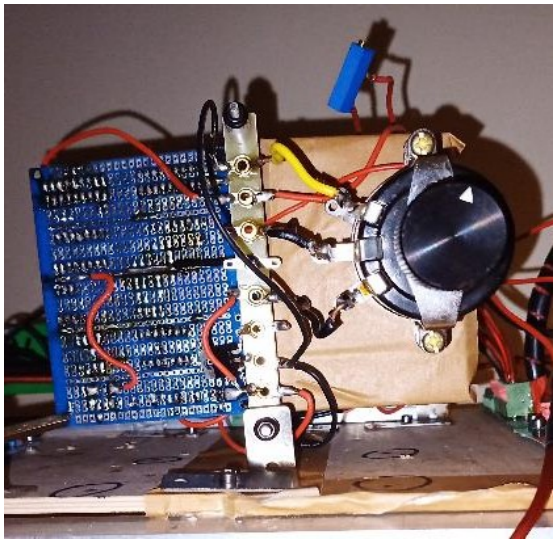


Figure Appendix D.4 Actual circuit of the analog accumulator.

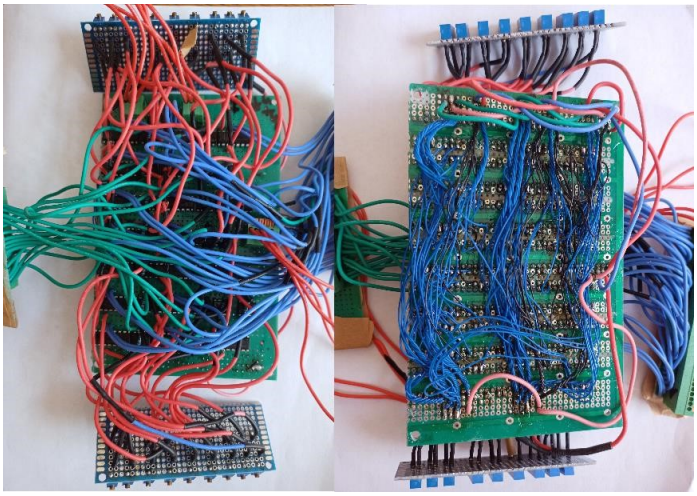


Figure Appendix D.5 Top and bottom of the actual signal monitoring circuit.

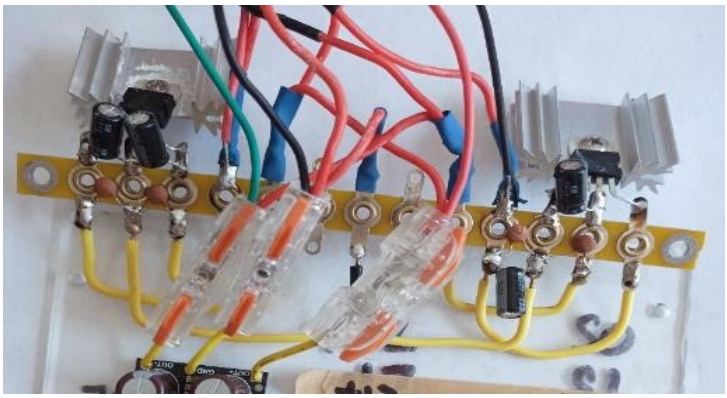


Figure Appendix D.6 Regulator for converting +12v and -12v to +5v and -5v.

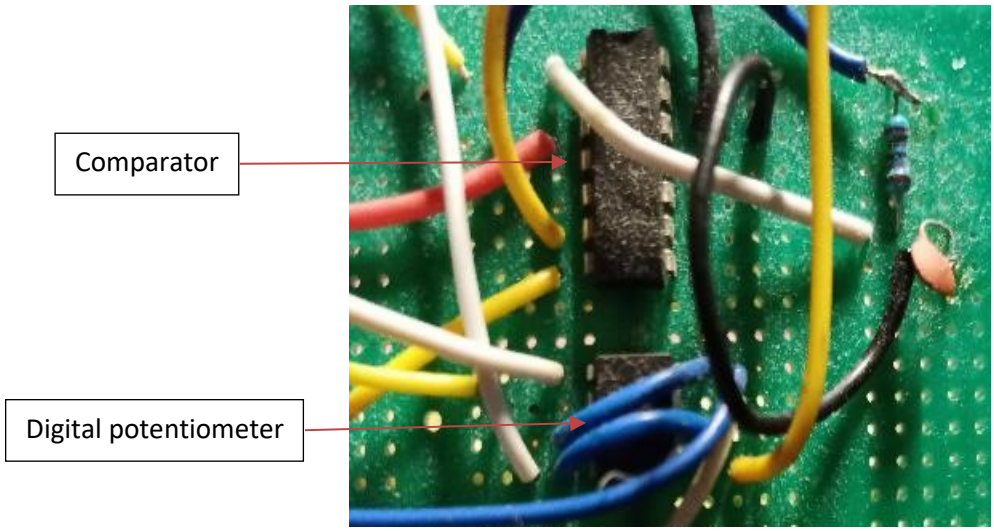


Figure Appendix D.7 Actual circuit of the resistor-based feedback loop.

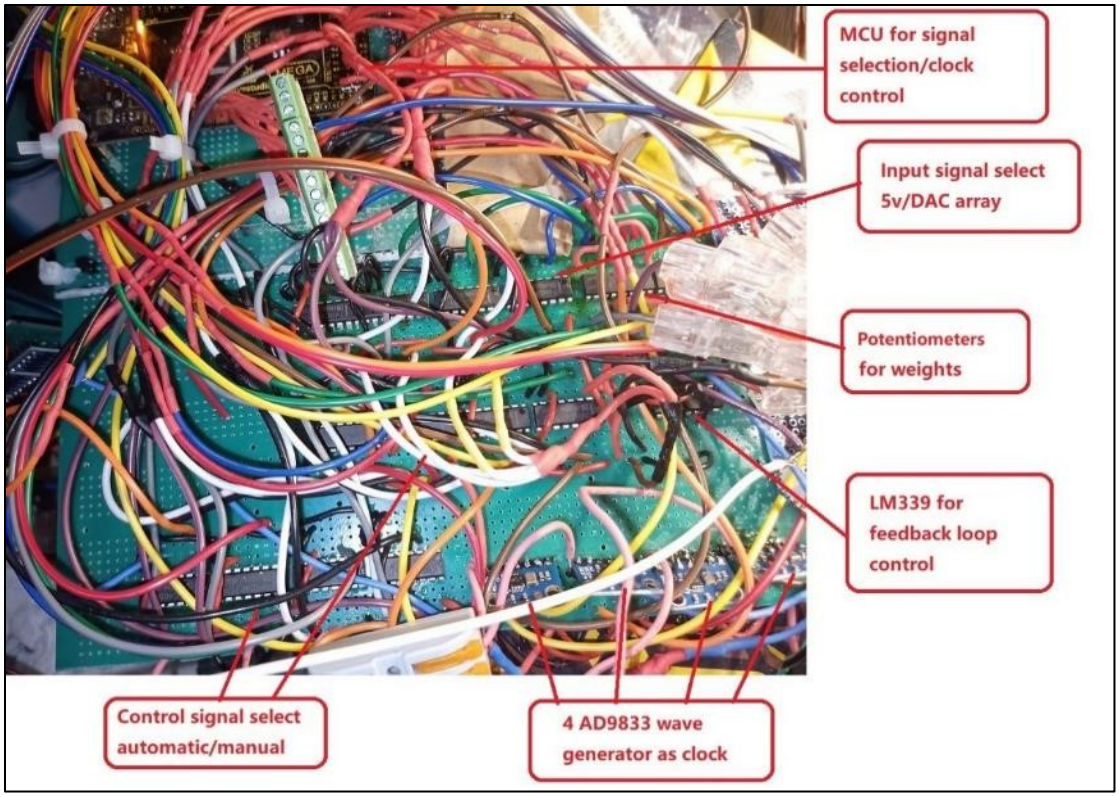


Figure Appendix D.8 Actual circuit of the true random matrix generator.

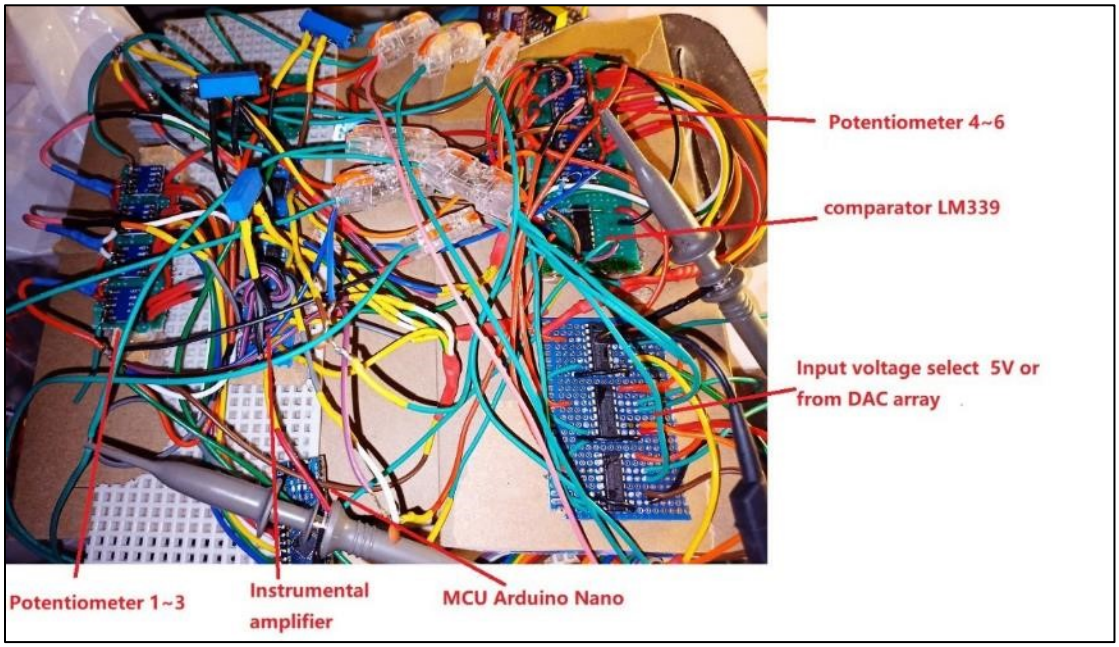


Figure Appendix D.9 Actual circuit of the add-division circuit.

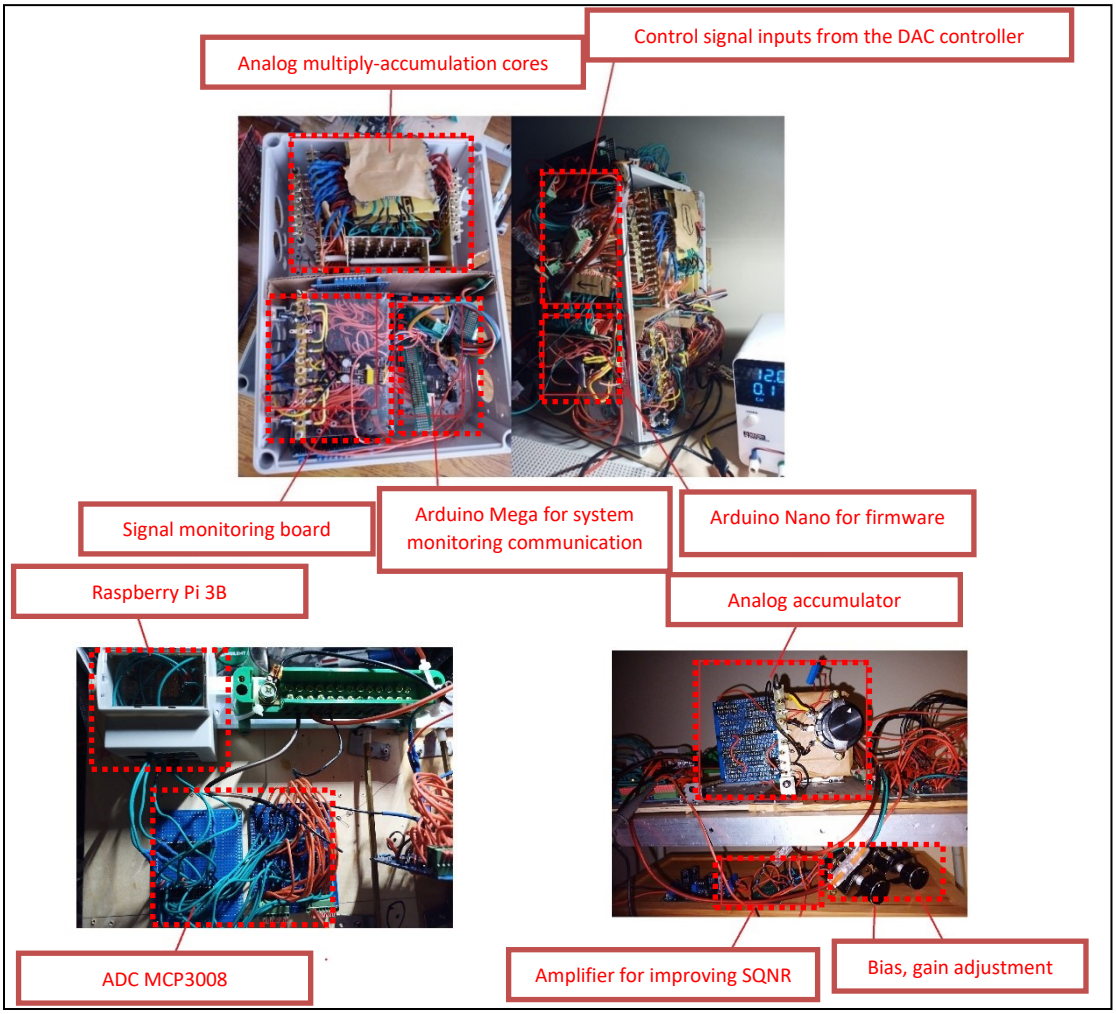


Figure Appendix D.10 Actual circuit of a multiply-accumulation system.

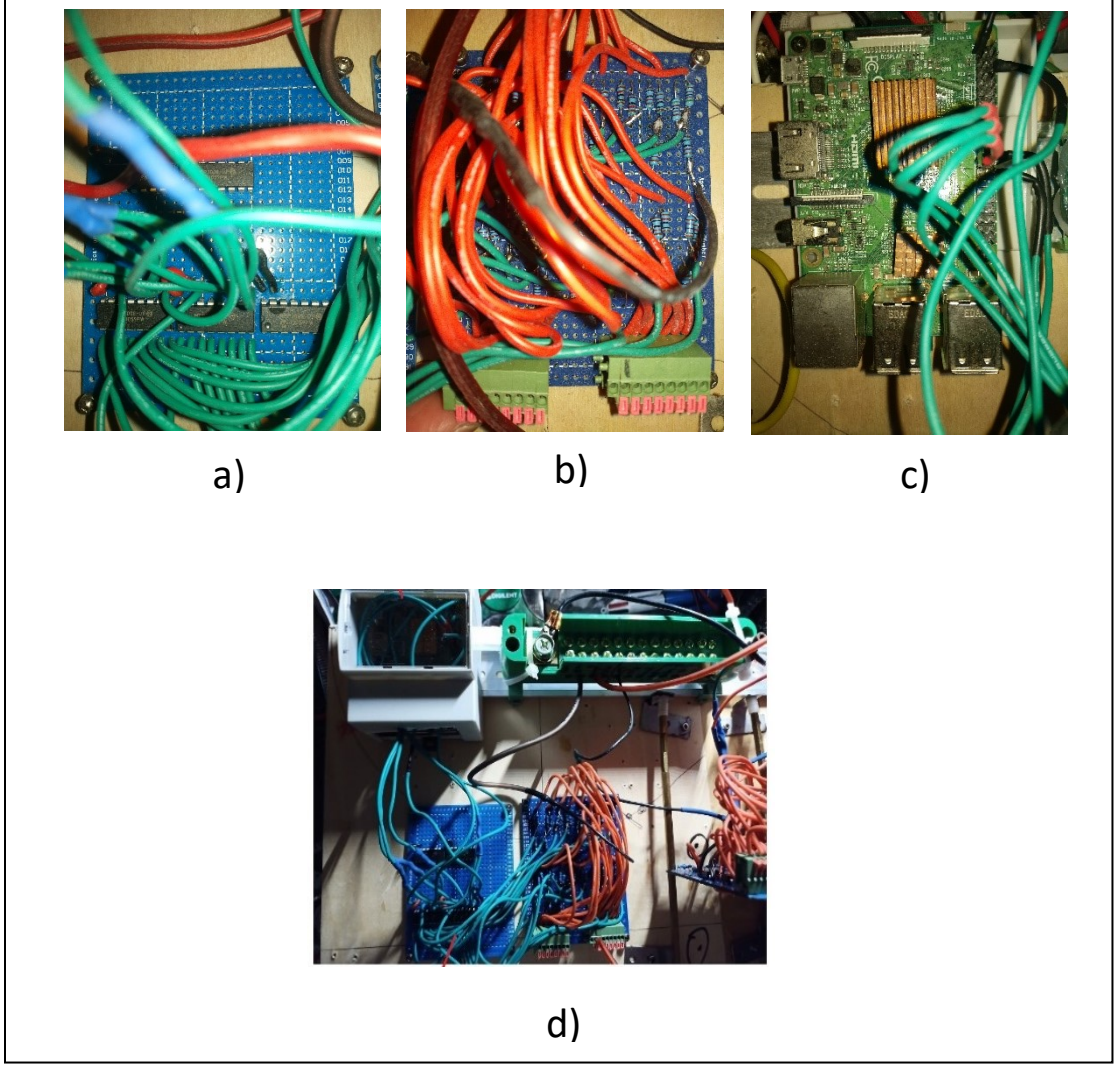


Figure Appendix D.11 Raspberry pi 3B with MCP3008 for measurements a) one of the 5 MCP3008 is used for the test b) a 5V to 3.3V resistors-based voltage converter c) a Raspberry pi 3B with connection of MCP3008 through SPI GPIO pins d) a full Raspberry Pi based measuring system.

The actual dual multiply-accumulation system.

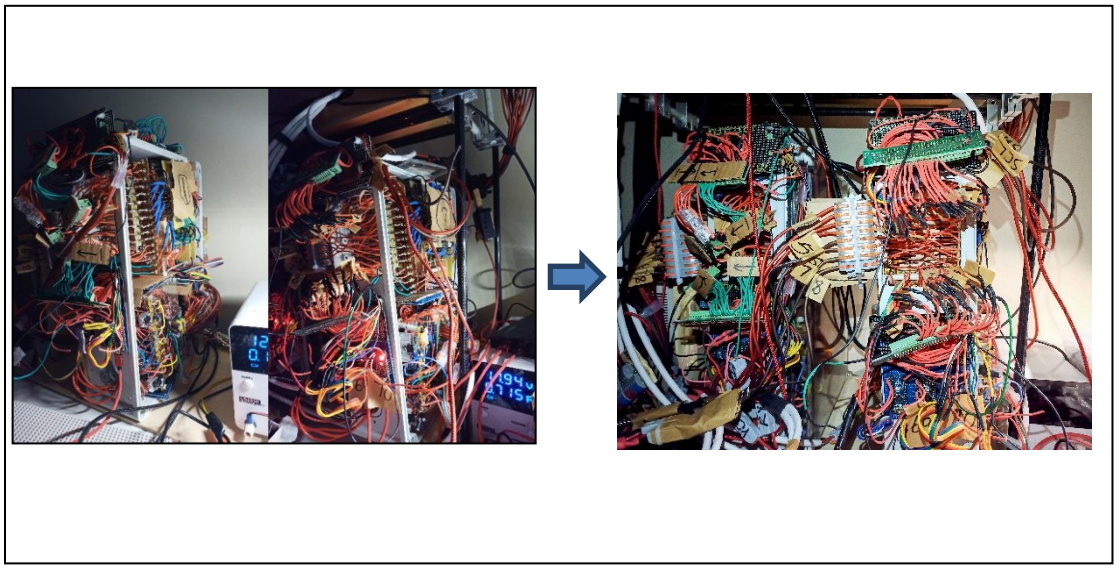


Figure Appendix D.12 Left:The No.1 and No.2 system. Right: the expanded No.1+No.2 system.

Appendix E

Required Math Theory and Simulations

1. Pattern Recognition

1.1. Recognition by Correlation

Cross-correlation is used for determining similarities of two matrices. In this section, we use a computer simulation for testing recognitions by cross-correlation. Cross-correlation equation (eqn. E.1),

$$\mathrm{r_{yx}}(l) = \sum_{n=-\infty}^{+\infty} \mathrm{y}(n) * \mathrm{x}(n-l) \qquad \textbf{(E.1)}$$

Where x and y are two different matrices and l is a shifting bias. Based on this algorithm, we generate a random wave based on a sine wave with length 500. Next, we pick a segment of the wave with length 50. Then, we calculate cross-correlation by shifting the segment to the left and right. Repeating the process for two times.

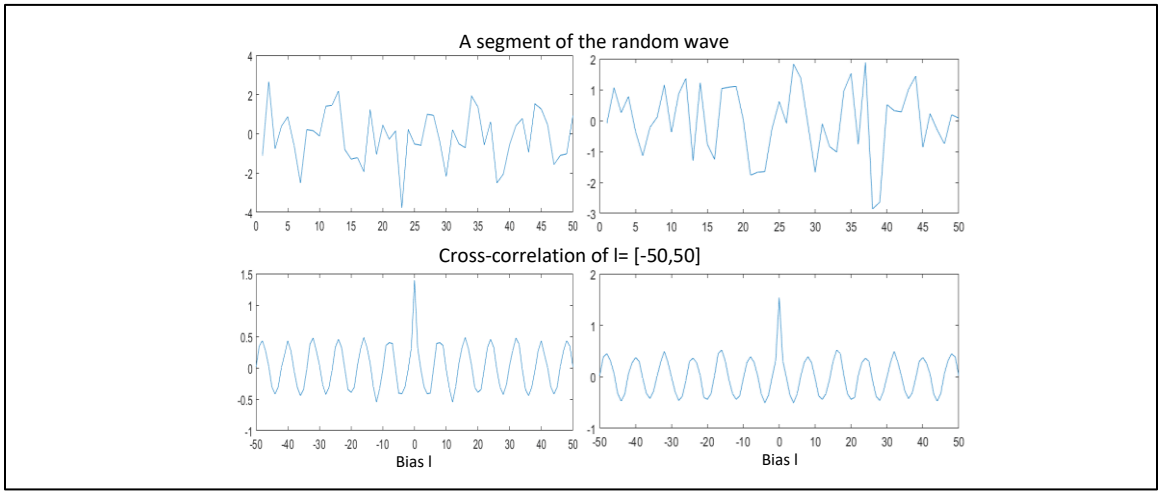


Figure Appendix E.1 Results of the two correlation tests.

Figure Appendix E.1 shows the cross-correlations achieve their peaks when their shifting bias are 0 where the two waves for cross-correlation calculations are the same. Therefore, cross-correlation can be a mathematical tool for image recognition.

1.2. Simple Pattern Recognition

Patterns are recognized by doing convolution operations for input matrices and convolution kernels. The results are cross-correlations that shows similarities of two matrices. During the computer simulation, randomly placing ten 3*3 squares in a 100*100 image. Using a 3*3 convolution kernel to find it. Similar as the simple neural network, the convolution result y(m,n) is based on equation E.2.

$$y\,(m,n) = \sum_{j=0}^{2} \sum_{i=0}^{2} x_{m+i,n+j} * w_{i,j} + b \qquad \textbf{(E.2)}$$

Where y(m,n) is a point of the convolution result, $x_{m+i,n+j}$ is a element of 3*3 matrix from the input image, $w_{i,j}$ are the weights of the convolution kernel, b=0. As shown in figure Appendix E.2, squares patterns of the input image are marked in the output image.

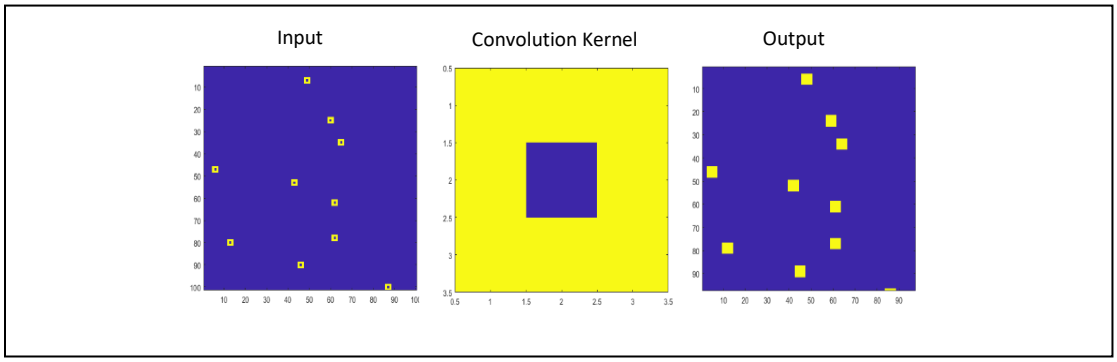


Figure Appendix E.2 Recognizing square pattern from an input image.

1.3. Pattern Recognition With Noise

Under nonideal conditions, noise of images leads difficulties for image recognitions. This section demonstrates recognizing expected patterns under noise by differentiating squares from triangles. In the computer simulation, the convolution kernel has output from both triangle and square, but the unwanted results are lower from the triangle. Therefore, applying b=-6 which is the highest number of the convolution result from triangle. Then we apply an activation function y=1 (y>0) and y=0 (y<=0) as equation E.3,

$$\begin{cases} \mathrm{y}\,(m,n) = \mathrm{f}\left(\sum_{j=0}^{2}\sum_{i=0}^{2} \mathrm{x}_{m+i,n+j} * \mathrm{w}_{i,j} - 6\right) \\ \mathrm{f}(\mathrm{y}\,(m,n)) = \begin{cases} 1\ (\mathrm{y} > 0) \\ 0\ (\mathrm{y} \le 0) \end{cases} \end{cases} \quad \textbf{(E.3)}$$

Where y(m,n) is a point of the convolution result, $x_{m+i,n+j}$ is a element of 3*3 matrix from the input image, $w_{i,j}$ are the weights of the convolution kernel, f() is a function for making a decision.

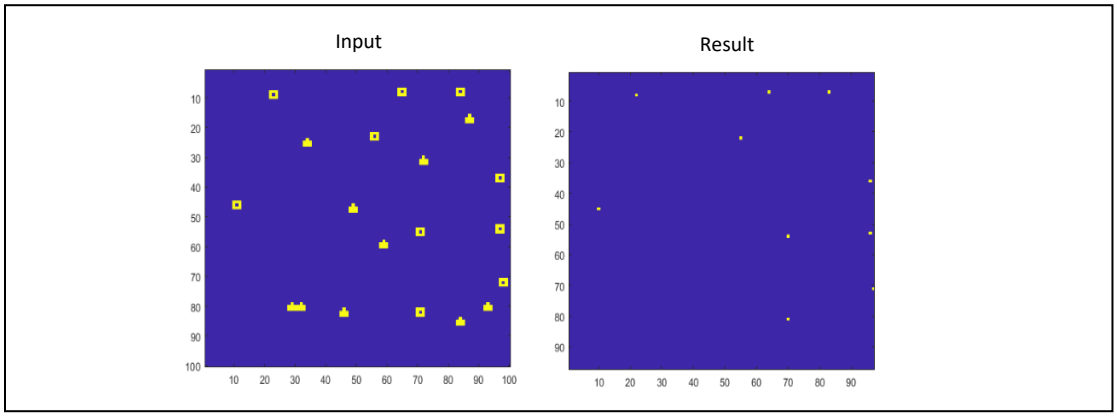


Figure Appendix E.3 Recognizing square from an image under noise.

Figure Appendix E.3 shows that the square patterns are marked but the triangles are not marked. Therefore, applying 'bias' and 'activation functions' gives threshold for the outputs.

1.4. Multiple Convolutional Kernel Recognition

Complicated patterns can be divided into several basic patterns for recognition. Next, combining the recognition results for making final decisions. During the computer simulation, we generate an image that contains the target patterns and some noise patterns. Also, setting the two convolution kernels to square and triangle. Figure Appendix E.4 shows a process for complicated pattern recognition. The final recognition result is a combination of the square recognition result and triangle recognition result.

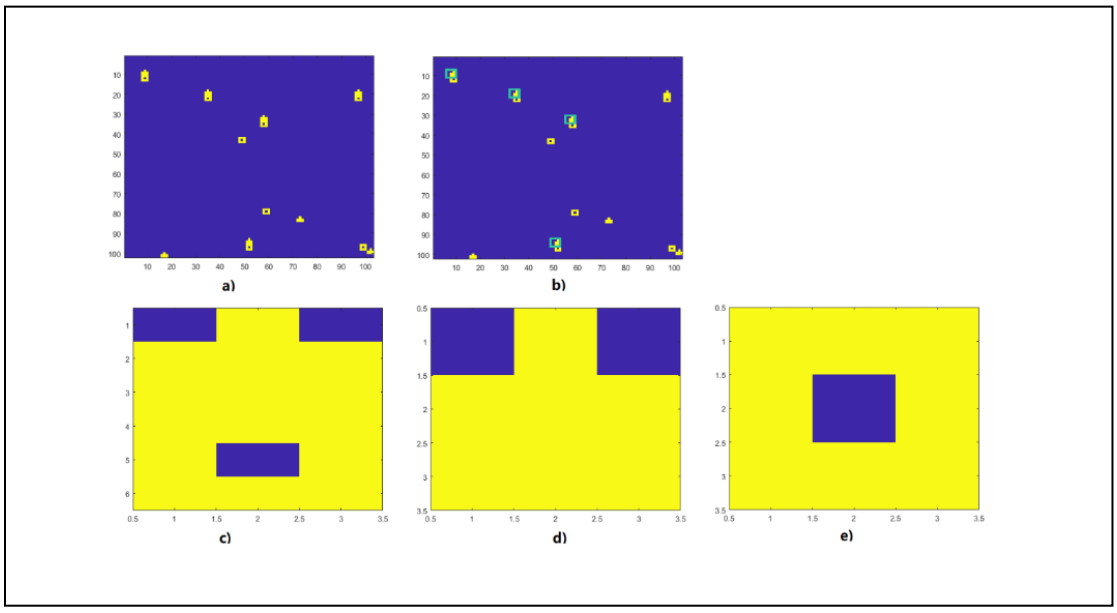


Figure Appendix E.4 a) Input image which contain triangle+square, triangle, square b) only triangle+square patterns are detected in the output image c) triangle+square d) triangle pattern used in the detection e) square pattern used in the detection.

1.5. Gradient Decent

1.5.1. Gradient Descent for One Unknown Weight

'Gradient Descent' is a mathematical tool for finding extremum of functions. In order to utilize it for finding solutions of neural networks, we need to create an equation with an independent variable and error which is 'loss function'. For demonstration, finding w from an input 0.5 and output 0.3 (eqn. E.4).

$$0.3 = 0.5 * w \tag{E.4}$$

The unknown variable w can be found by the equation: trying random w values until error is 0 (eqn. E.5).

$$y = (\text{error})^2 = [0.5 * (w + \text{step}) - 0.3]^2 \tag{E.5}$$

We plot the equation in the MATLAB and find the expect w value manually.

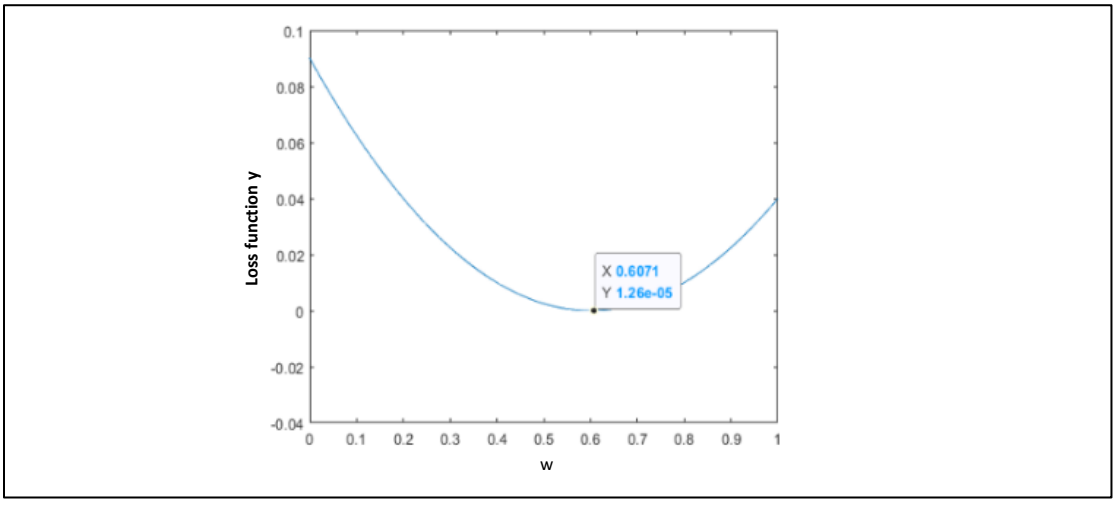


Figure Appendix E.5 Expect w found by manual.

Figure Appendix E.5 shows that when w≈0.6071 the loss function has the lowest

value. Computers can find the same point by Gradient descent algorithm. The flow chart diagram in figure Appendix E.6 shows the algorithm: Starting from a random w, finding its current gradient, finding gradient of the next point, going to the direction that gradient is decreasing.

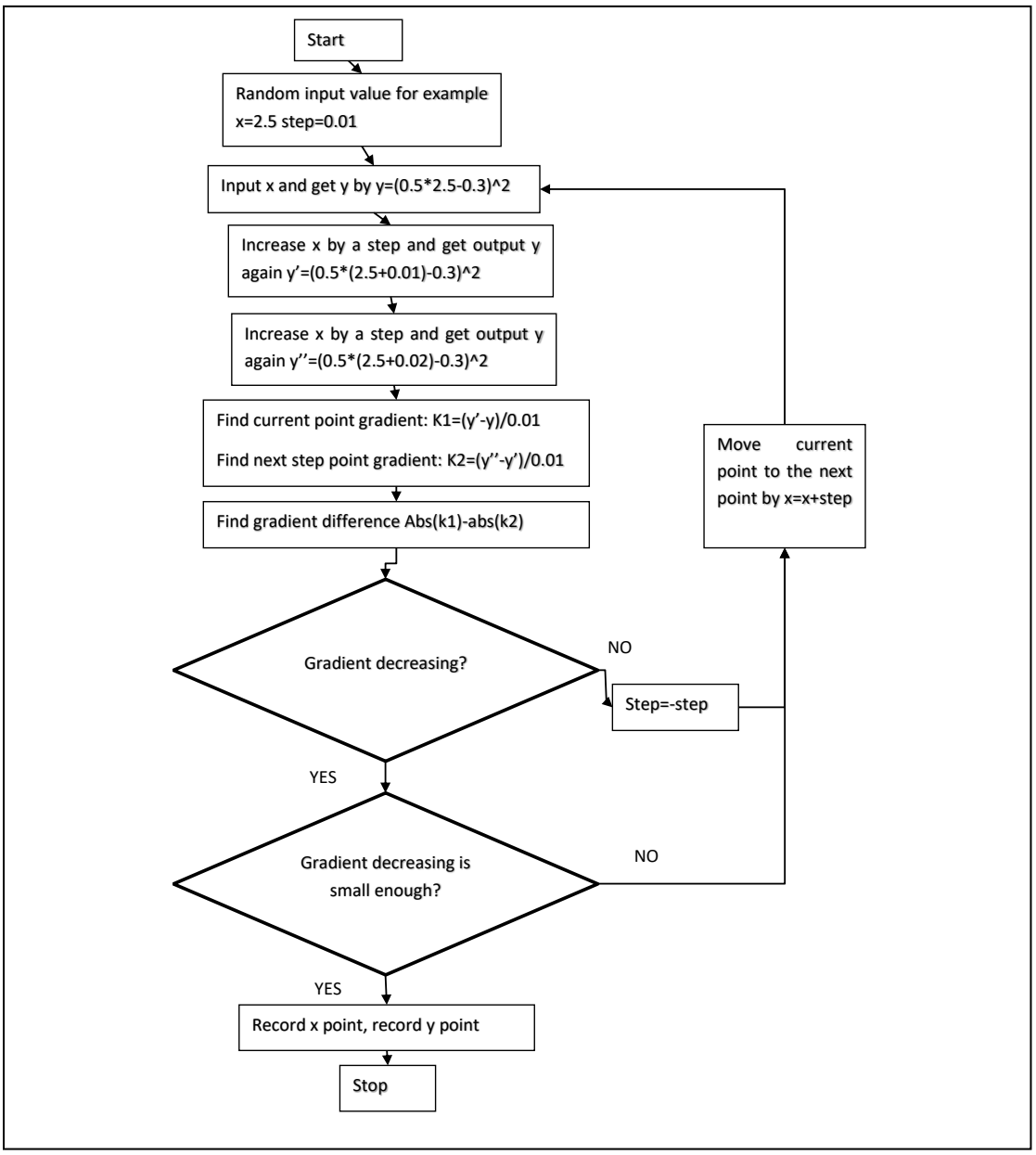


Figure Appendix E.6 Gradient descent algorithm.

Finally, the result is plotted in figure Appendix E.7, it shows the result of applying Gradient descent algorithm. The loss function has the lowest value when w=0.61 which is close to the manually found point w=0.607.

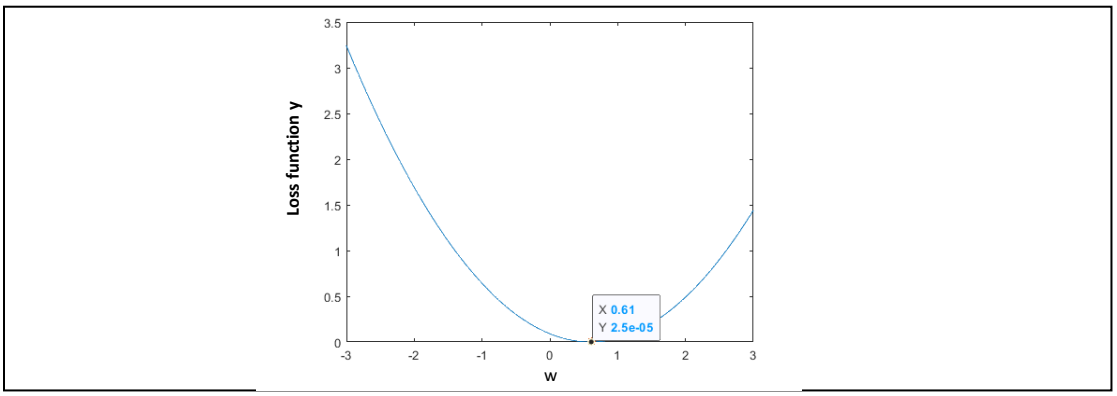


Figure Appendix E.7 Find w by gradient descent algorithm.

1.5.2. Gradient Descent for Multiple Unknown Weight Matrix

Gradient descent algorithm also can be used for higher dimensional weight matrices. In this demonstration, applying gradient descent algorithm for finding a weight matrix **W** that leads the largest cross-correlation. We suppose a 3 unknown weights neural network under input matrix [3 5 3] is provided (eqn. E.6):

$$\mathrm{z}\,(w1, w2, w3) = w1 * 3 + w2 * 5 + w3 * 3 \qquad \textbf{(E.6)}$$

If we plot the plane in a 4D space with linear changing independent variable a,b,c. They should be normalized so that w1+w2+w3=1 (eqn. E.7).

$$\mathrm{z}(a, b, c) = \frac{a}{a+b+c} * 3 + \frac{b}{a+b+c} * 5 + \frac{c}{a+b+c} * 3 \qquad \textbf{(E.7)}$$

If c is constant, the equation becomes a plane in 3D space (eqn. E.8).

$$z\,(x,y) = \frac{x}{x+y+c} * 3 + \frac{y}{x+y+c} * 5 + \frac{c}{x+y+c} * 3 \qquad \textbf{(E.8)}$$

Next, we plot some projections from the 4D equation to 3D space. As shown in figure Appendix E.8 a), b), and c), we can find that the extremum of z is at x gradient=0 and y gradient=0.

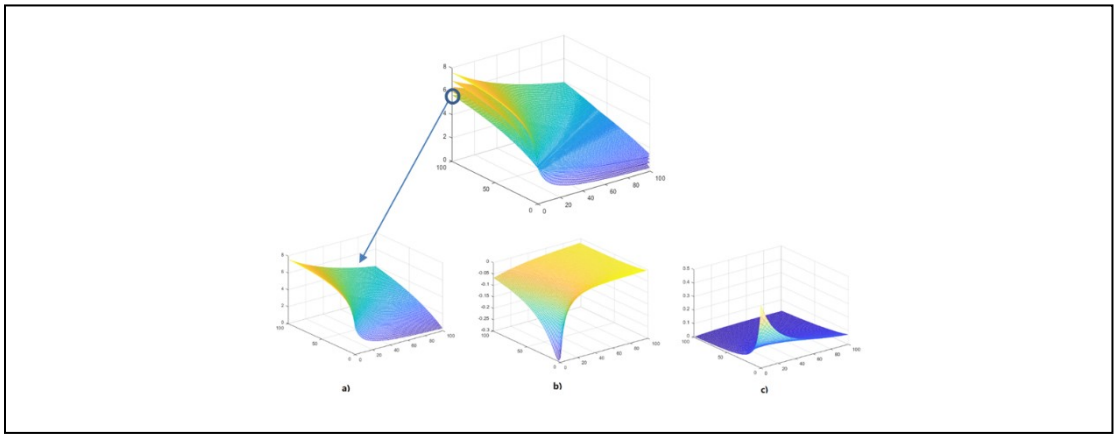


Figure Appendix E.8 a) z=3*w1+5*w2+3*w3 plane where w1+w2+w3=1. b) gradient on x direction c) gradient on y direction.

Finally, we apply Gradient descent algorithm on both directions and plotting the results frame by frame. As shown in figure Appendix E.9, the red point is going to the position that has gradient [0,0]. The stop position is at the extremum of the plane where z has the largest cross-correlation.

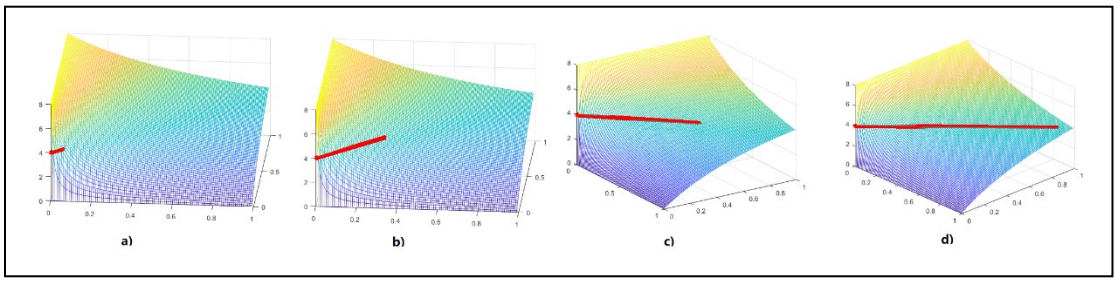


Figure Appendix E.9 Applying gradient descent to move the red point to the extremum a) frame 1 at time 1 b) frame 2 at time 2. c) frame 3 at time 3. d) frame 4 at time 4.

1.6. PID Control Algorithm

A PID controller contains three variables for calculating adjusting values for the next step (proportional, Integral, differential). PID controlling theory is used for an unknown system that only input and output can be measured.

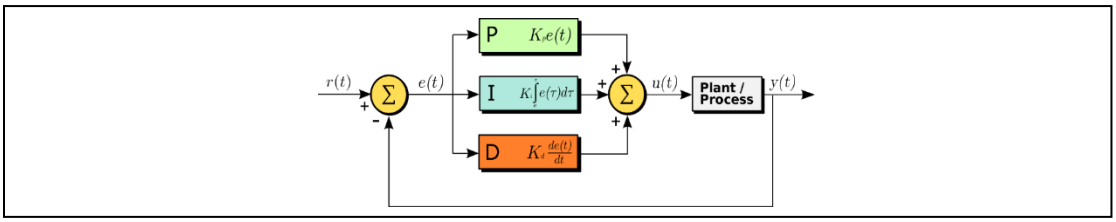


Figure Appendix E.10 Illustration for PID controlling algorithm [48].

Figure Appendix E.10 shows that the adjusting steps of PID controllers are based on errors between outputs and expected outputs: 'P' represents taking proportion of error as adjusting step, 'I' represents taking integration of error, and 'D' represents differentiation of error. Three of them can be applied for adjustments simultaneously or respectively. In this thesis, proportional control is used for controlling the feedback loop so that to attenuate fluctuation of stable stages. During the computer simulation, plotting a self-adjustment curve without proportional control first.

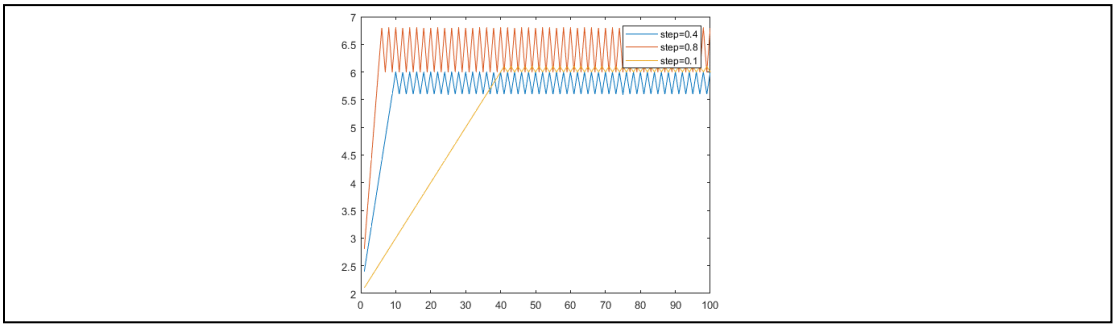


Figure Appendix E.11 Self-adjustment curve under different step sizes 0.1,0.4,0.8.

Figure Appendix E.11 shows that larger step has faster adjustment speed but has large ripple on the stable area. However, small step has lower adjustment speed but smaller ripple on the stable area. Next, we apply proportional control and compare the output curves. The algorithm is shown in figure Appendix E.12.

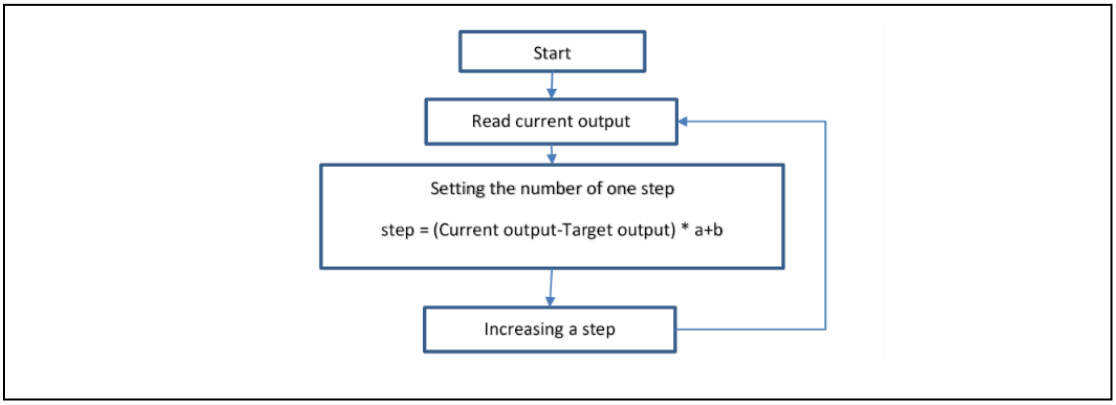


Figure Appendix E.12 Proportional control algorithm.

As shown in figure Appendix E.13, the differences of adjustment speeds are the same, but ripples of the flat area are attenuated.

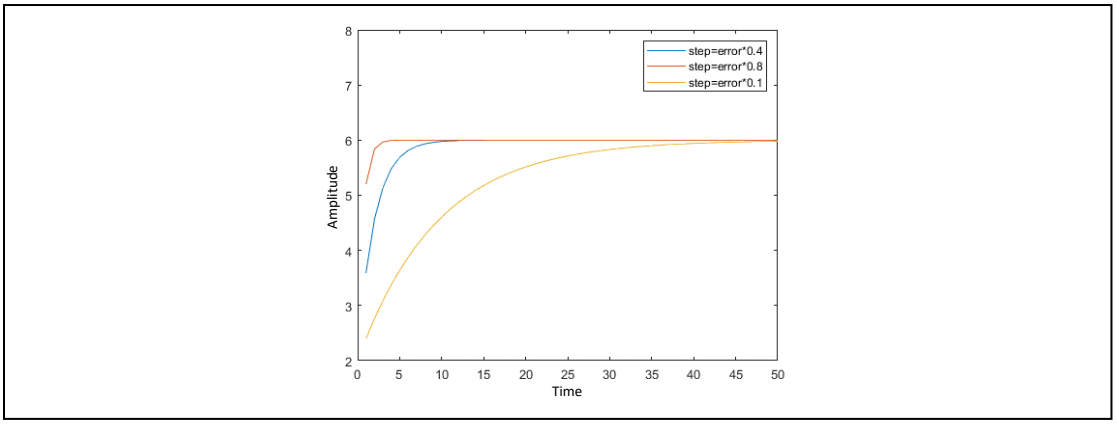


Figure Appendix E.13 Self adjustment curve under different step sizes under proportional feedback control.

1.7. Convolutional Neural Network Simulations

This section illustrates a preset convolutional neural network without training. Convolutional neural networks extract features of an image by multiple convolution kernels and making decisions by fully connected neural networks.

1.7.1. Pooling Layer

'Pooling' can down sample an input image for converting images to a smaller matrix with meaningful data. Figure Appendix E.14 shows that the input image is down sampled after pooling and the matrix is small but contains enough useful information.

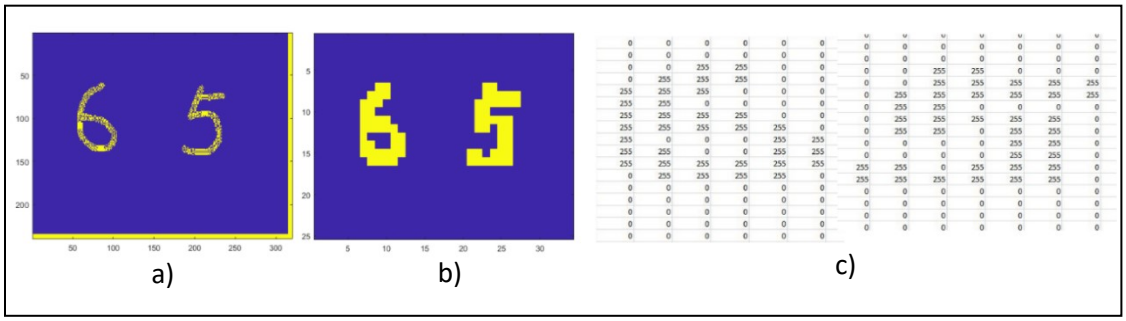


Figure Appendix E.14 Doing pooling operation for the input hand-writing image a) input image b) down-sampled image c) numerical matrix of b).

1.7.2. Random Convolution Kernels

Patterns can be recognized by manually setting convolution kernels, but it is hard to specify patterns manually from a complicate image such as a normal photo. Deep

learning solves this problem by self-adjustment through feedback stops the adjustments once its error matrix is small enough. We do a test for patterns extraction by applying some random 3*3 convolution kernels with bias and activation functions (figure Appendix E.15). From the results we find that some random convolution kernels are able to extract the numbers' features. Next, passing the convolution result through the pooling layer so that useful information is retained (figure Appendix E.16).

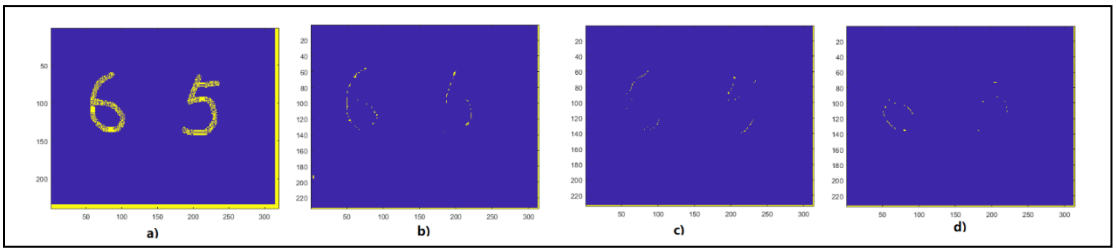


Figure Appendix E.15 Extracting random characteristics by random convolution kernels a) input image. b) extracted result 1. c) extracted result 2. d) extracted result 3.

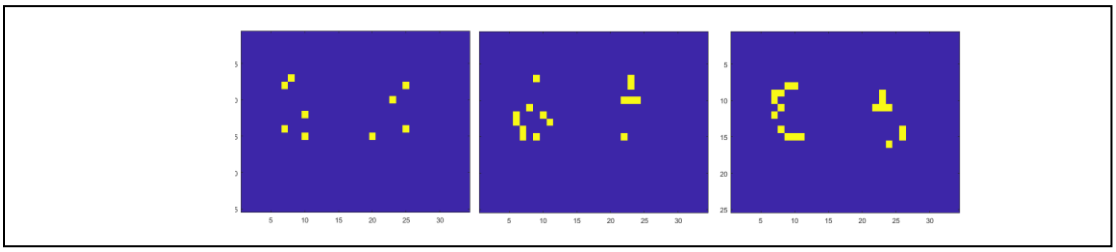


Figure Appendix E.16 Convolution results after pooling layer.

In order to distinguish '5' with '6', we create a simple convolutional neural network with a convolutional layer and a fully connected layers after the pooling layer. The full connection layer is generated randomly and self-adjusted according to the errors. Figure Appendix E.17 shows each step of a small convolutional neural network. The image is decomposed to some matrices for the fully connected neural network making decisions.

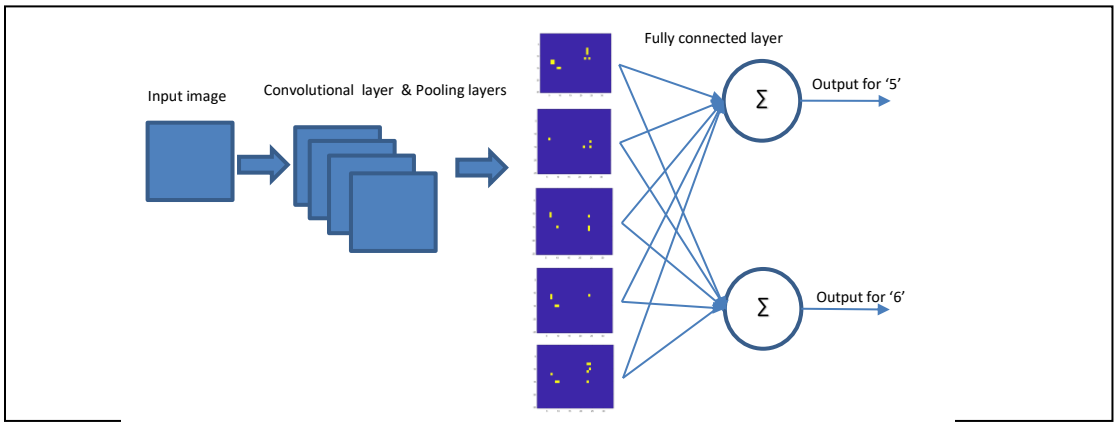


Figure Appendix E.17 Convolutional neural network structure.

Appendix F

Calculation of DSP Filters

Appendix F shows the calculations of DSP filters. As shown in the computer simulations, edge detection filters have best performance at $\omega_c=\pm\pi/2$. Therefore, we use $\omega_c=\pm\pi/2$ for the following calculations for demonstration. We assume a high pass filter has $zero_{1,2}= \pm\pi/2$, $zero_3=0$ and gain=1, the pole-zero of the FIR filter equation F.1 is

$$H(z) = \left(1 - e^{j\frac{\pi}{2}} * z^{-1}\right) * \left(1 - e^{-j\frac{\pi}{2}} * z^{-1}\right) * (1 - e^{0} * z^{-1}) \quad \textbf{(F.1)}$$

We can verify the frequency response by plotting the zero-pole plot at z-domain and frequency response (figure Appendix F.1), where frequency ω is the normalized angular frequency.

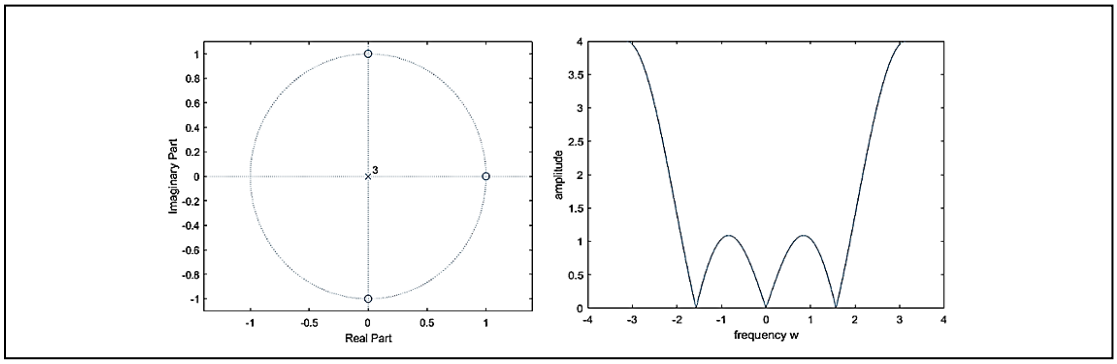


Figure Appendix F.1 Plotting 'pole-zero' figure and frequency response figure for verifying the filter's function.

Next, converting the H(z) to polynomial form of frequency response at z-domain (eqn. F.2).

$$\frac{Y(z)}{X(z)} = H(z) = \frac{\sum_{k=0}^{M} b_k * z^{-k}}{1+\sum_{k=1}^{M} a_k * z^{-k}} = 1 - 1 * z^{-1} + 1 * z^{-2} - z^{-3} \quad \textbf{(F.2)}$$

Next, converting H(z) to its corresponding difference equation F.3

$$y(n) = -\sum_{k=1}^{N} a_k * y(n-k) + \sum_{k=0}^{M} b_k * x(n-k) = 1 * x(n) - 1 * x(n-1) + 1 * x(n-2) - x(n-3) \quad \textbf{(F.3)}$$

Next, we can draw a block diagram by referring the difference equation. Parameters of the filter is [1, -1, 1, -1] which is also a convolutional kernel. The block diagram of figure Appendix F.2 can be used for hardware realization.

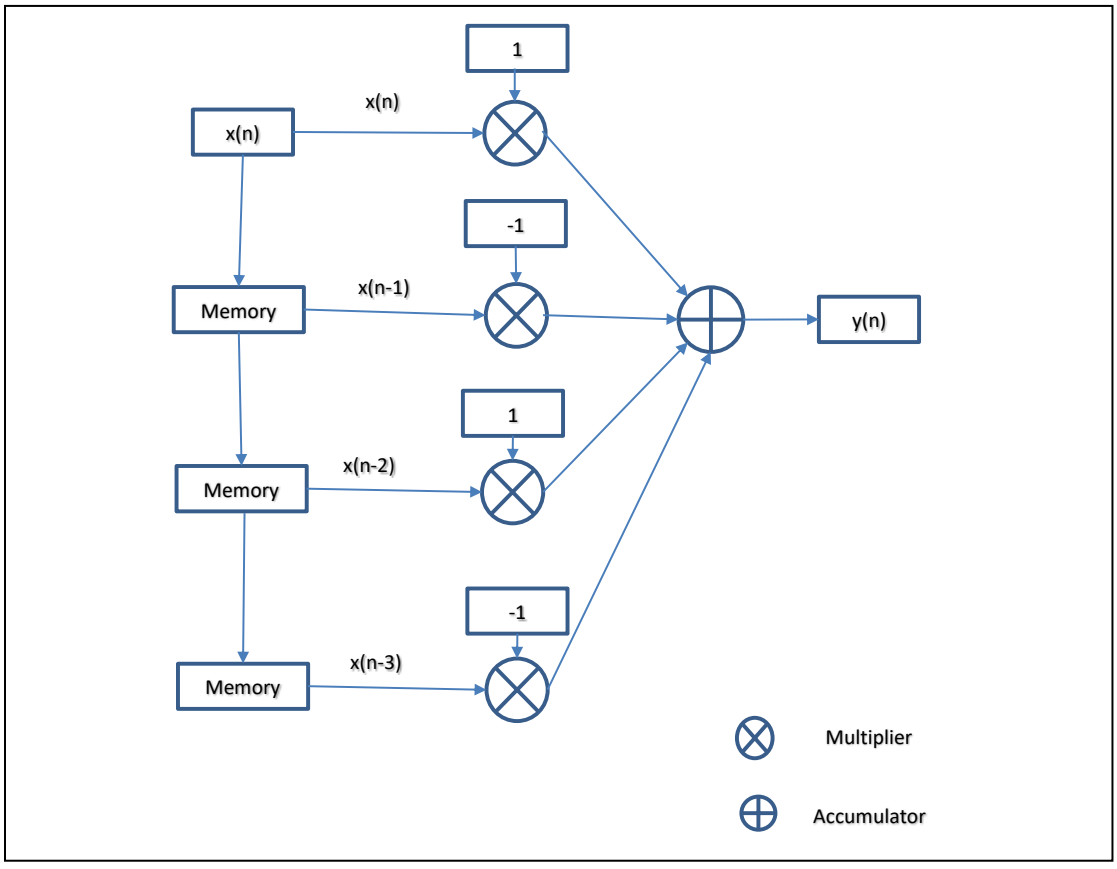


Figure Appendix F.2 Block diagram for realization.

Appendix G

Basic Performance Tests for the Analog Multiply-Accumulation Array and the System

1. Digital Potentiometer Swapping Range Tests

1.1. Test Introduction

Tests in this section are for selecting an appropriated type of Wheatstone bridge which is important for multiply-accumulation implementations. We consider the performance implementations of output swapping range and linearity. A Wheatstone bridge can be implemented by one, two, or four digital potentiometers. In the following tests, the different types of Wheatstone bridges have been given specific names to distinguish them.

1.2. Test Bench Design and Setup

Table G.1 Pre-setting convolution kernel test bench

Inputs	Measurement Device	Controller	Device under test	Analysis Software
5V power supply	Analog discovery 2	STM32F103	Resistor array	MATLAB

The test bench is connected as indicated in table G.1 and figure Appendix G.1. Schematic of a 'Full bridge' (figure Appendix G.2).

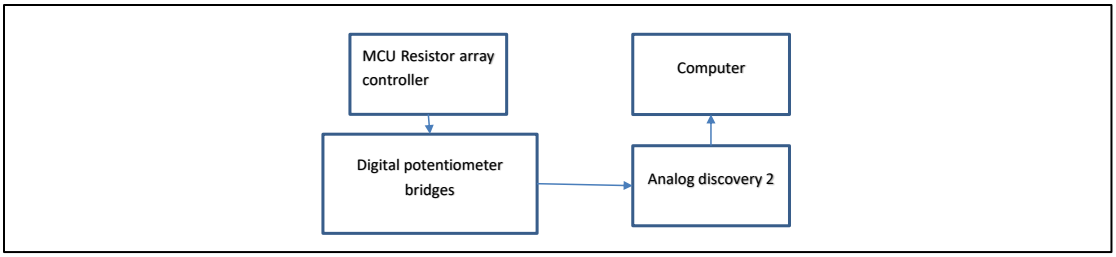


Figure Appendix G.1 Block diagram of the testbench.

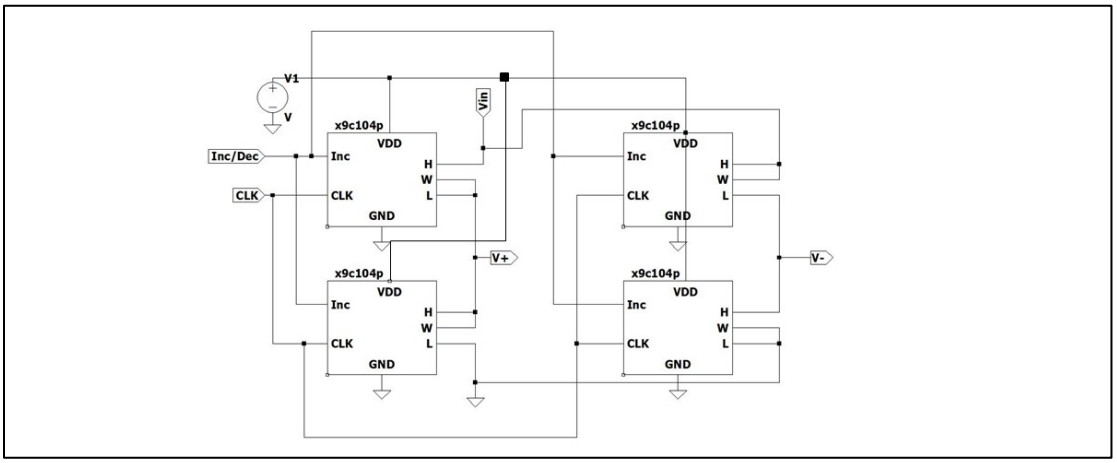


Figure Appendix G.2 The 4*x9c104p full Wheatstone bridge.

Schematic of a 'Half bridge' (figure Appendix G.3)

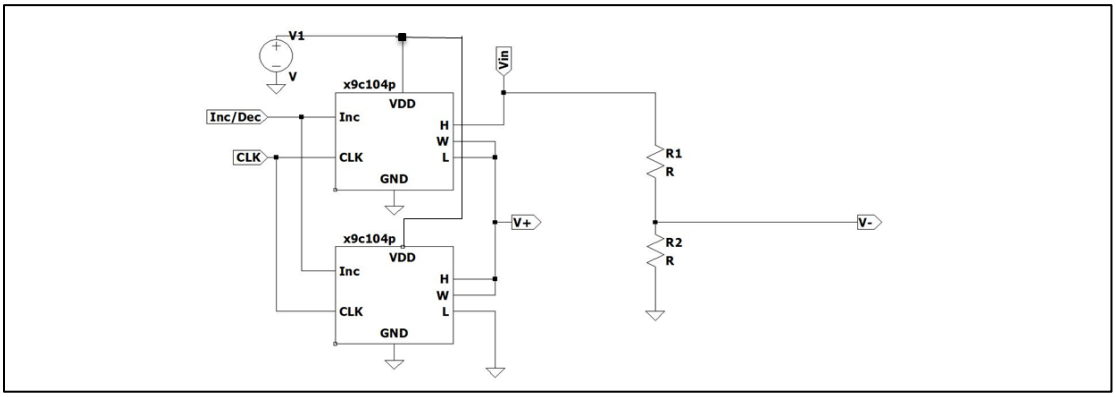


Figure Appendix G.3 The 2*x9c104p+2 fixed resistors half Wheatstone bridge.

Schematic of a ‘Single’ (figure Appendix G.4)

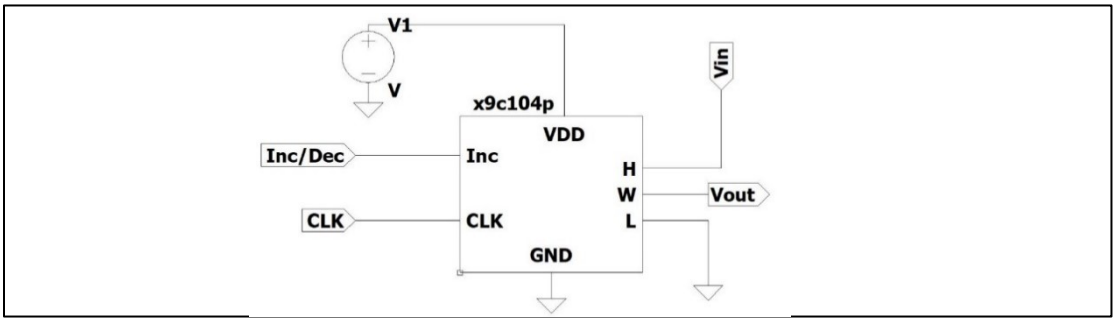


Figure Appendix G.4 Single x9c104p.

Schematic of a ‘Single half’ (figure Appendix G.5)

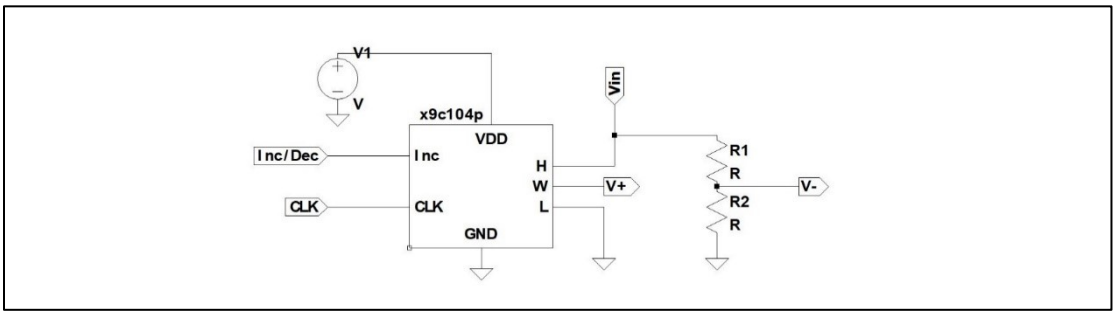


Figure Appendix G.5 Single x9c104p with 2 fixed resistors constructed half Wheatstone bridge.

Schematic of a ‘Parallel double’ (figure Appendix G.6)

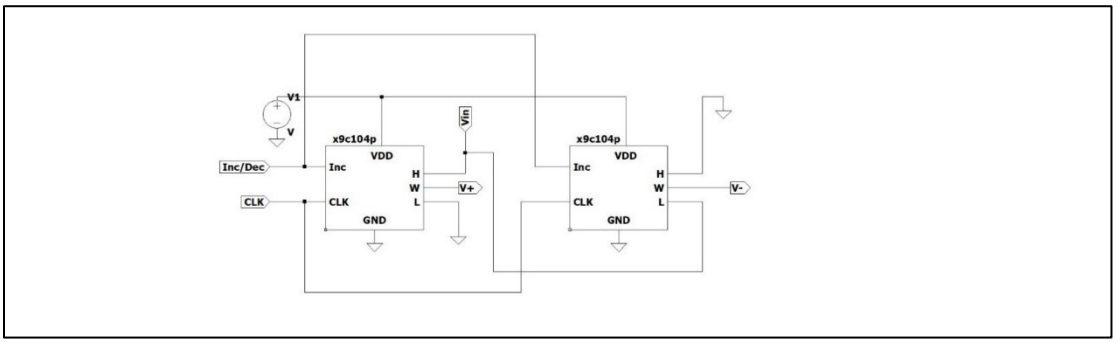


Figure Appendix G.6 Double x9c104p constructed full Wheatstone bridge.

1.3. Test Method

To test the Wheatstone bridges, we implement the following: 1. program the MCU controller to swap the digital potentiometers' resistances. 2. simultaneously, using the Analog discovery 2 oscilloscope to capture their outputs.

1.4. Test Result

The test result is shown in figure Appendix G.7. The measured linearities for each are presented in Table G.2.

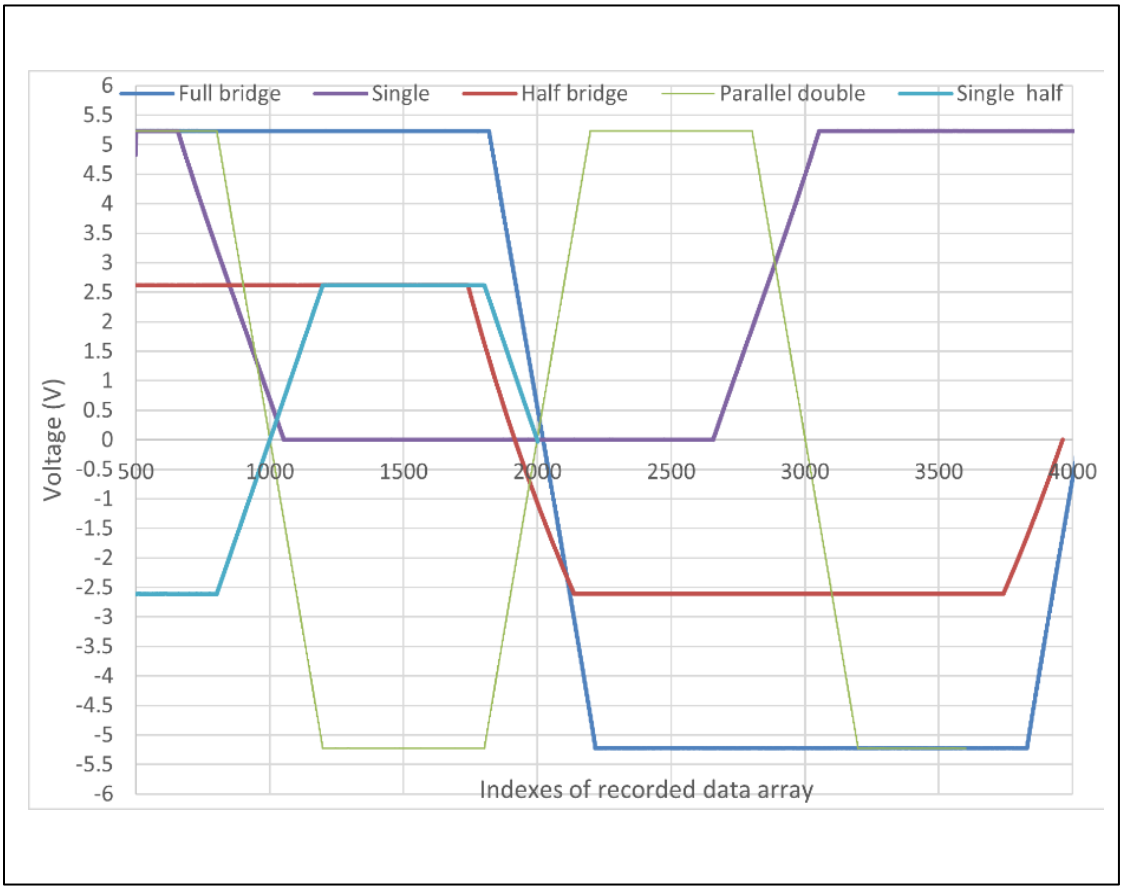


Figure Appendix G.7 Under 5V output swap of different Wheatstone bridges.

Table G.2 Performance of different Wheatstone bridges

Bridge type	Number of potential meters	Range	Linearity
Full bridge	4	-5.22V~5.23V	0.9999
Half bridge	2	-2.6V~2.62V	0.9962
Single	1	0V~5V	0.9998
Single half	1	-2.60V~2.62V	0.9999
Parallel double	2	-5.22V~5.22V	0.9999

1.5. Conclusion

The test result table G.2 shows that the potentiometers have been adjusted to high linearity during design in factory. The digital potentiometer type which has name 'Single half' is selected since it provides positive and negative weights by a single unit.

2. Storing Binary Patterns

2.1. Test Introduction

This section tests managing capabilities of the 'signal monitoring board' and the firmware and pattern memorizing capability of the digital potentiometer matrices.

2.2. Test Method

First, we initialize all elements of the two digital potentiometer matrices to the lowest resistance (about 0Ω) and setting all the DACs' outputs to 5V. Next, we set the first element of both matrices to 100kΩ and reading outputs of the DAC matrix and potentiometer matrix through the monitoring boards and the Arduino Mega. Next, we

store the voltage matrices in the MATLAB. Next, all elements are initialized, and we set the second element as above. The process is repeated until the ninth element. Finally, we convert the voltage matrices to images by equation 7.1.

$$\text{Image value} = \frac{V_{Read}}{5} * 255 \quad \textbf{(7.1)}$$

2.3. Test Results

First, we test each element of a 'convolution core' which contains two 3*3 digital potentiometer metrices so that all ICs are soldered on PCBs properly. During the test, we use the 'Signal monitoring board' with the Arduino Mega for collecting outputs and recording in MATLAB. Also, this is a test for the 'Signal monitoring board' so that it can read voltage from the DAC matrices and the digital potentiometer matrices (figure Appendix G.8).

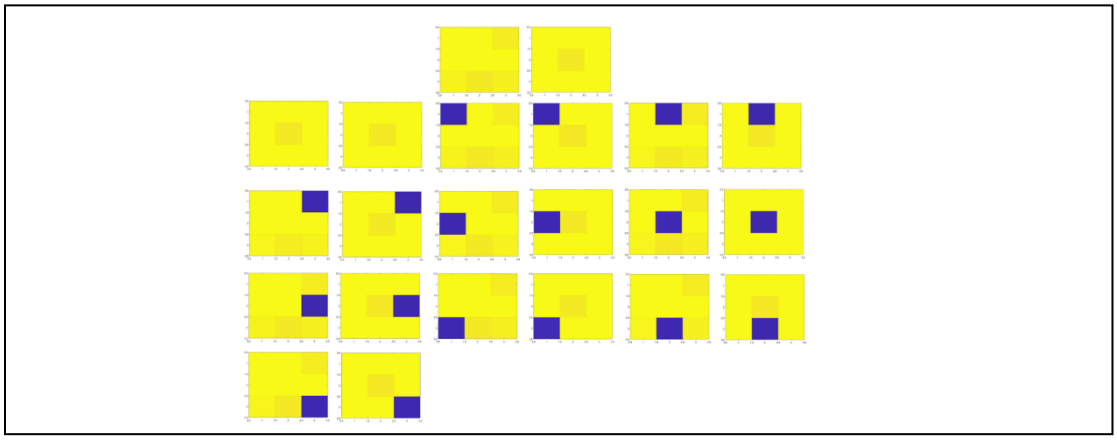


Figure Appendix G.8 Pattern memorizing performance test of the two cores. The first row: DAC inputs for the two 3*3 matrices. The others: digital potentiometer matrices controlling test.

Next, we set some random binary value patterns for testing its patterns memorizing performance (figure Appendix G.9).

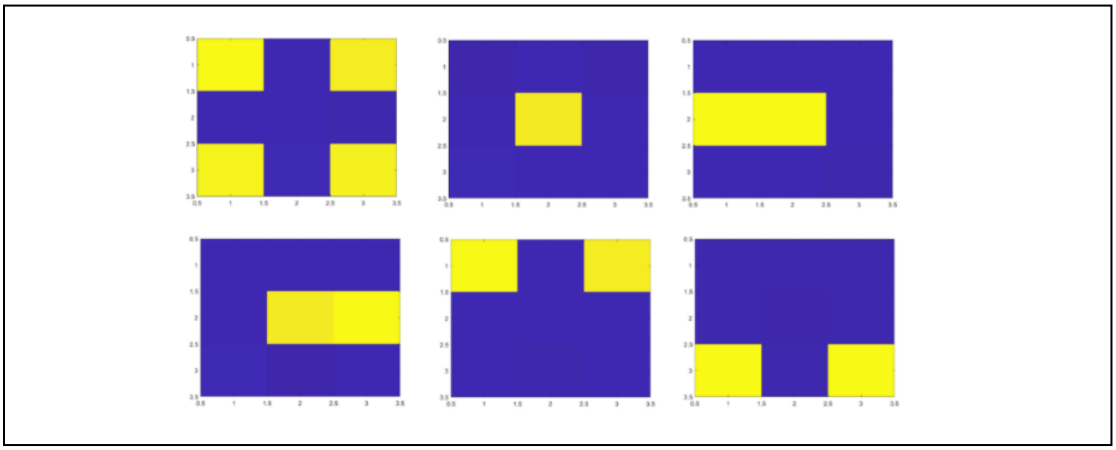


Figure Appendix G.9 Random binary value patterns memorizing performance test.

3. Storing Multiple Gray-Level Patterns

3.1. Test Introduction

Tests of this section ensure the combinations of DACs and digital potentiometers are workable for a digital image filter with multiple levels.

3.2. Test Method

We set some random gray-level patterns for testing the system's multiple value memorizing performance. The digital potentiometer matrices are set through the firmware (figure Appendix G.10). Next, we set all DACs to 5V and reading the output voltages and record the results in MATLAB.

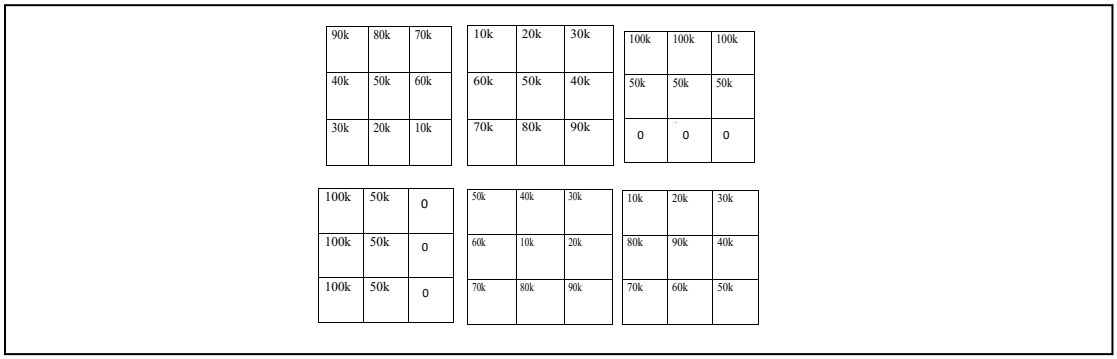


Figure Appendix G.10 Actual resistance sets for the digital potentiometer matrices.

3.3. Test Results

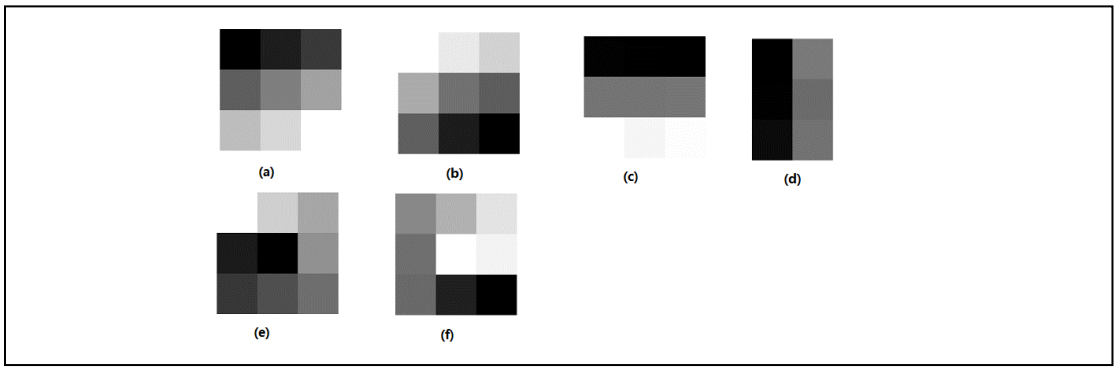


Figure Appendix G.11 Multiple level pattern memorizing performance test.

Figure Appendix G.11 shows capability of the system for storing multiple grey levels patterns.

4. Basic Tests of the CNN implementation

4.1. Pattern Learning

4.1.1. Test Introduction

This section shows tests of patterns learning and recognition based on convolutional calculations of the system. Convolution kernels can be learnt from input

matrices in multiple ways. For the digital potentiometer matrices, one specific method is to follow the DAC input voltages through the feedback loops. However, the restriction of this method is that inputs should be kept between 0V->5V so that the digital potentiometer matrices can follow. In practice, input matrices may have different ranges. Tests show convolutional kernel learning from inputs by applying the latter learning method.

4.1.2. Test Method

We start by generating a random 3*3 binary matrix in MATLAB and feeding it into the DACs matrix. Next, we apply the algorithm of convolutional training for finding a convolutional kernel which leads the highest cross-correlation between the input pattern and the convolutional kernel (Mentioned in Section 1.5 of Appendix E). Next, we set all the DACs to output 5V for reading the saved patterns and record the patterns with MATLAB. Finally, we repeat the test for 3 times comparing the input patterns with the stored patterns.

4.1.3. Test Result

Figure Appendix G.12 shows comparisons of the feed in patterns, actual input patterns, and the learnt patterns.

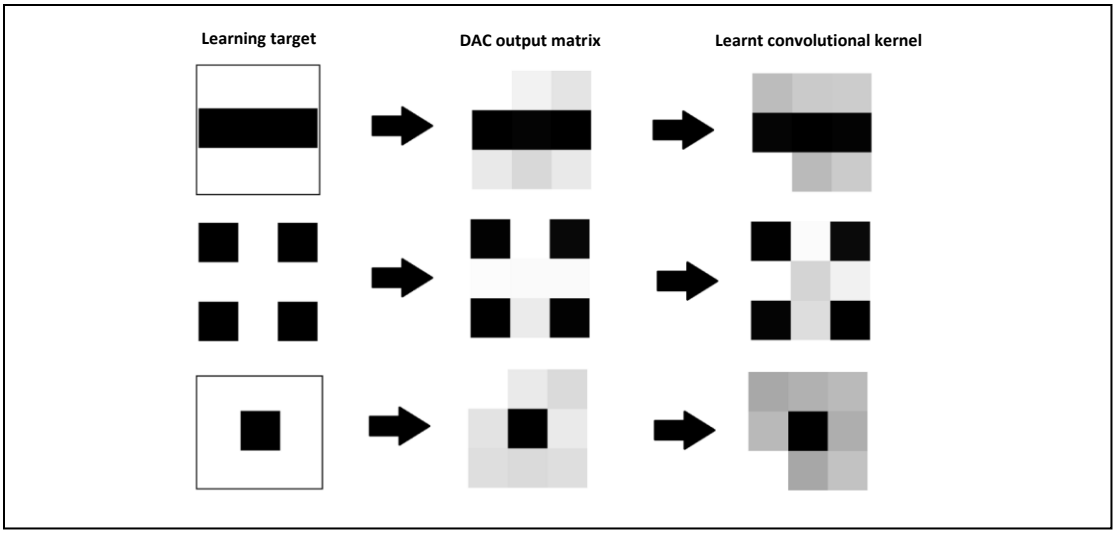


Figure Appendix G.12 Test result of pattern learning.

4.2. Pattern Recognition

4.2.1. Test Method

First we use the system to learn a pattern by the convolutional kernel training algorithm and store the pattern using MATLAB. In this test, we select the first pattern of figure 7.8 which has voltage matrix [5V,5V,5V; 0V, 0V, 0V; 5V,5V,5V]. Next, we feed the stored pattern into the DAC matrix and record its auto-correlation result for comparison. The recorded wight matrix in the digital potentiometer matrix is [0.5,0.5,0.5; -0.5, -0.5, -0.5; 0.5,0.5,0.5]. Next, we feed 5 random patterns into the DAC matrix and recording their cross-correlation results. Finally, the cross-correlation of random patterns are compared with auto-correlation of the target pattern.

4.2.2. Test Results

As shown in figure Appendix G.13, similarity of a random pattern can be reflected by its cross-correlation. The 5th random pattern has the highest similarity and the highest cross-correlation.

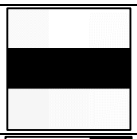
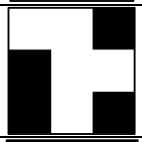
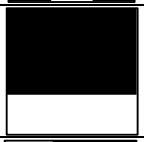
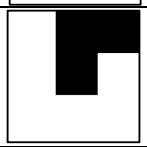
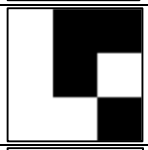
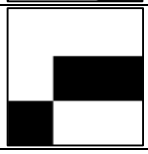
		Correlation
Target pattern		15
Random pattern 1		2.5
Random pattern 2		7.5
Random pattern 3		5
Random pattern 4		2.5
Random pattern 5		10

Figure Appendix G.13 Target pattern and 5 random patterns.

4.3. A Simple Neural Network for Distinguishing Patterns

4.3.1. Test Introduction

Instead of a single weight, a 3*3 pattern can also be treated as a weight for constructing a fully-connected neural network. In the test of this section, creasing a

simple fully-connected neural network based on two patterns for decisions making.

4.3.2. Test Method

We implement the learning and recording of two patterns, which are ‘cross’ and ‘square’, into the two digital potentiometer matrices. Next, we feed in a ‘cross’ pattern to the neural network and record the multiply-accumulation result through MATLAB. Next, we feed in the ‘square’ pattern to the neural network and recording multiply-accumulation result through MATLAB (figure Appendix G.14). Finally, we compare the results.

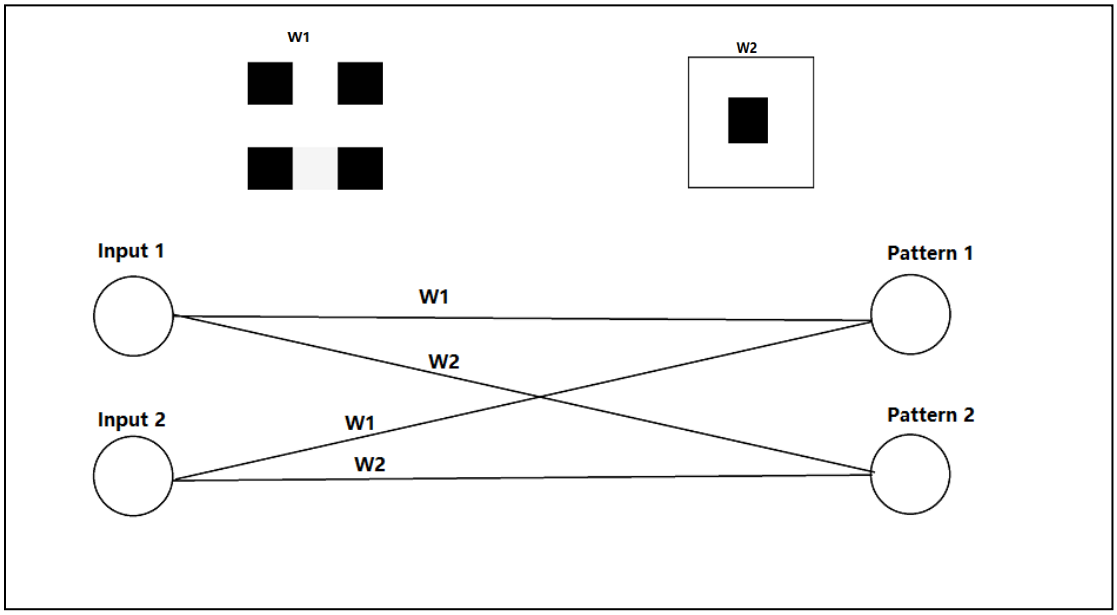


Figure Appendix G.14 A simple neural network with pattern weights w1, w2.

4.3.3. Test Results

The tests results are shown below by gray-level images where the black area represents 0 and the white areas represent 1. As shown in figure Appendix G.15, the

'square' filter has higher convolution output when the input is 'square' and the 'cross' filter has higher convolution output when the input is 'cross'. By helping with software, the system can implement neural networks for pattern recognition tasks.

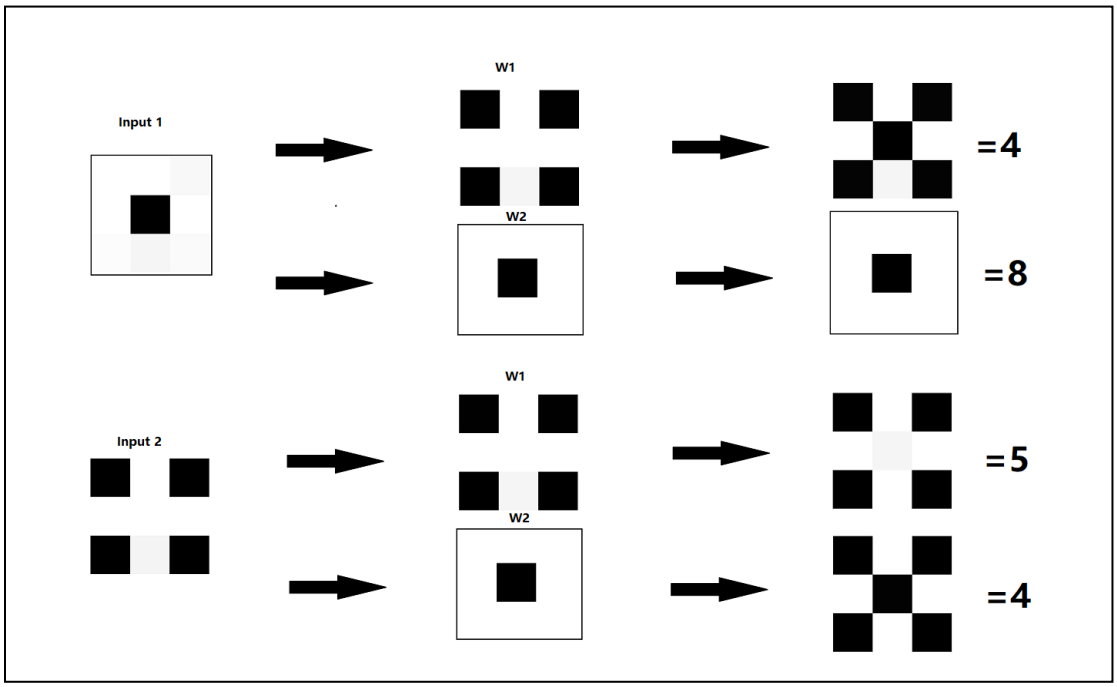


Figure Appendix G.15 Test results of distinguishing two patterns by pattern recognition (white represents 1 and black represents 0).

Appendix H

Calculations Concerning the Image Filters of Computer Vision Transplantation

Appendix H shows image filters transplantations, we have four filters in the test: $\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$, $\begin{bmatrix}-1 & 2 & -1\\ 0 & 0 & 0\\ -1 & 2 & -1\end{bmatrix}$, $\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$, and $\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$. Also, each digital potentiometer has 100 steps and output voltage/step is 0.01 V_{in}. Voltage divider ratio of a multiply unit is [-0.5V/V, 0.5V/V]. Therefore, we can convert the four filters as the following equations.

For the first filter, we set the middle element to 0.4V/V, the other elements are -0.4V/V/8 = -0.05V/V. Next, we can calculate their steps.

Voltage divider ratio matrix:

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix} \rightarrow \begin{bmatrix}-0.05 & -0.05 & -0.05\\ -0.05 & 0.4 & -0.05\\ -0.05 & -0.05 & -0.05\end{bmatrix} \quad \textbf{(H.1)}$$

Steps matrix for the firmware:

$$\begin{bmatrix}50-\frac{0.05}{0.5}*50 & 50-\frac{0.05}{0.5}*50 & 50-\frac{0.05}{0.5}*50\\ 50-\frac{0.05}{0.5}*50 & 50+\frac{0.4}{0.5}*50 & 50-\frac{0.05}{0.5}*50\\ 50-\frac{0.05}{0.5}*50 & 50-\frac{0.05}{0.5}*50 & 50-\frac{0.05}{0.5}*50\end{bmatrix} \rightarrow \begin{bmatrix}45 & 45 & 45\\ 45 & 90 & 45\\ 45 & 45 & 45\end{bmatrix} \quad \textbf{(H.2)}$$

Similarly, for the second filter, we can set 2 to 0.5V/V and -1 to -0.25V/V.

$$\begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix} \rightarrow \begin{bmatrix} -0.25 & 0.5 & -0.25 \\ 0 & 0 & 0 \\ -0.25 & 0.5 & -0.25 \end{bmatrix} \quad \textbf{(H.3)}$$

Its step matrix is

$$\begin{bmatrix} 50 - \frac{0.25}{0.5} * 50 & 50 + \frac{0.5}{0.5} * 50 & 50 - \frac{0.25}{0.5} * 50 \\ 50 & 50 & 50 \\ 50 - \frac{0.25}{0.5} * 50 & 50 + \frac{0.5}{0.5} * 50 & 50 - \frac{0.25}{0.5} * 50 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 100 & 25 \\ 50 & 50 & 50 \\ 25 & 100 & 25 \end{bmatrix} \quad \textbf{(H.4)}$$

For the Mean filter, we set all elements to 0.5V/V for convenient.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad \textbf{(H.5)}$$

Its step matrix is

$$\begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \rightarrow \begin{bmatrix} 100 & 100 & 100 \\ 100 & 100 & 100 \\ 100 & 100 & 100 \end{bmatrix} \quad \textbf{(H.6)}$$

For the Median filter, we set 4 to 0.4V/V, 2 to 0.2V/V, and 1 to 0.1V/V for convenient.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0.1 & 0.2 & 0.1 \\ 0.2 & 0.4 & 0.2 \\ 0.1 & 0.2 & 0.1 \end{bmatrix} \quad \textbf{(H.7)}$$

Its step matrix is

$$\begin{bmatrix} 50 + \frac{0.1}{0.5} * 50 & 50 + \frac{0.2}{0.5} * 50 & 50 + \frac{0.1}{0.5} * 50 \\ 50 + \frac{0.2}{0.5} * 50 & 50 + \frac{0.4}{0.5} * 50 & 50 + \frac{0.2}{0.5} * 50 \\ 50 + \frac{0.1}{0.5} * 50 & 50 + \frac{0.2}{0.5} * 50 & 50 + \frac{0.1}{0.5} * 50 \end{bmatrix} \rightarrow \begin{bmatrix} 60 & 70 & 60 \\ 70 & 90 & 70 \\ 60 & 70 & 60 \end{bmatrix} \quad \textbf{(H.8)}$$