TABLE OF CONTENTS

Contents

ACKNOWLEGEMENTS

I would like to thank Dipl.-Inf. Matthias Jung as my supervisor for his invaluable input in regards of my project. None of this would have been possible without the constant support and advice. This project is my first experience in the field of hardware system design. Before the project, all of my study are just focus on the image processing software system developing. With much help from Matthias, right now, I can implement the normal algorithm on the hardware platform. It makes the content of my study completely and meaningfully.

1 Introduction

1.1 Background and motivation

With the development of the society, logistics becomes more and more important in our life. As a consumer at the end of the value chains, usually we don't need to care about how it really works at behind. However, if the management of the logistics flow is easier and faster, we all would benefit from it. During the goods transport and storage, forklift trucks are indispensable tools for manual pallet handing. In order to aid the forklift driver especially under limited view condition, we usually equip the forklift with the ability of image processing.

The essential function of the forklift truck is to transport the goods and manage them in the warehouse. The most common forklift trucks are no ability to dealing with anything about the image processing. They are just simply equipped with camera and monitor. The lift pole, the fork and the roof of the forklift truck usually limit the view of the driver.

To support the forklift truck driver during his work, there is already some of the image processing assistant system available now [1]. From the image processing views, commonly, two kinds of the methods are used. One is that installing the powerful computer and with a great graphic card to make the system good enough and solve most problem using software program. Another way is that we design some of simple and small special-purpose logics to do the computing in avoid of much power needed and with cheaper and faster speed.

As the need of the people become higher and higher, the image size and bit depths grow larger and larger. HD video is the standard for every camera and even Full HD or 4K are quite popular. Software has become less useful in the video processing realm, because for the normal use, real-time system is the minimum requirement when we install it on the forklift. We can see that some of forklifts have already connected with PC and use the software to improve the image quality or read and detect the barcode of the goods. However, the cost of the PC is much higher than the special-purpose logics. Not only the normal computer itself, but also the cable behind it, as the cable is expensive to install and use. From this reason, it is obviously advantageous that if we could design some of the hardware logics and use those logics on the forklift truck to deal with the most common issues. It is the goal of the project to develop the image

processing system under the hardware logics and integrate them into the forklift instead of the powerful PC for cheaper, more portable and faster speed.

1.2 Related platforms

1.2.1 Raspberry Pi

Recently, the Raspberry Pi [2] has become a viable target platform for the using as a micro PC. The little of the credit-card size can be used in electronics projects as same as the desktop PC does. With the ability of the playing high-definition video, it is a good choice to dealing with the image and video processing. Using the battery, no any other external power suppliers are needed.

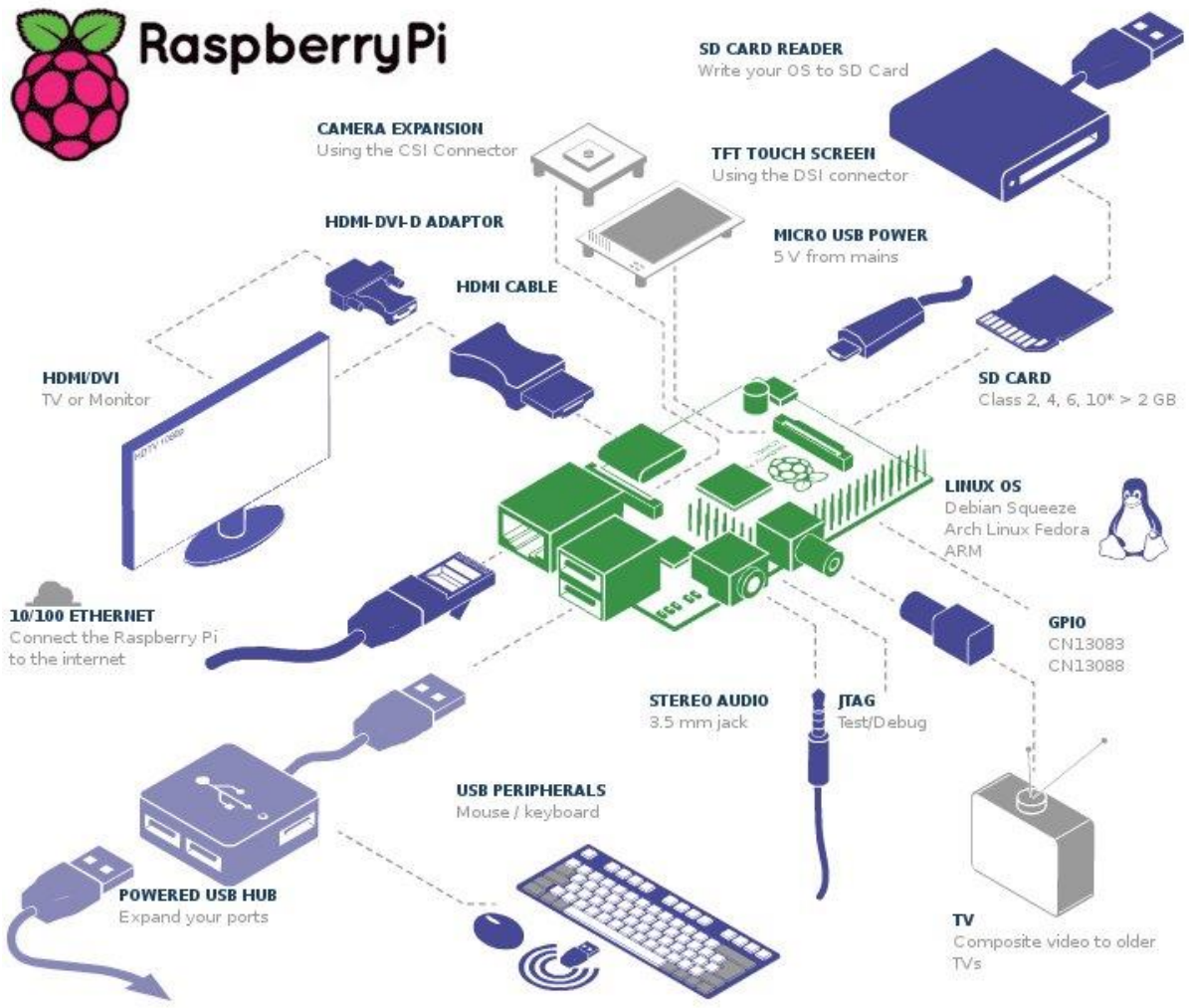


Figure 1-1 Raspberry Pi [2]

From the website of the Raspberry Pi, we know that it is manufactured in three board configurations which are Newark element14, RS Components and Egoman. The Raspberry Pi is composed of one system on chip (SoC), Broadcom BCM2835, including an ARM1176JZF-S 700MHz processor, VideoCore IV GPU, and 512 megabytes RAM in the model B [3]. The Broadcom SoC in the Raspberry Pi operates at 700MHz by default and equivalents to the 0.041 GFLOPS. The GPU in the Raspberry Pi also has enough power which providers 1 Gpixel/s or 1.5 Gpixel/s of graphics processing and 24 GFLOPS of general purpose computing performance. This ability of the graphics is enough to decode the full HD video. By the peripherals part, the generic USB port is included on the Raspberry Pi board. The USB keyboard and mic are also compatible. The SD card is used as the hard disk or solid-state drive for booting and persistent storage. HDMI port is for video and audio displaying. 26xGPIO could be used for specific functions including I2C, SPI, UART, PCM, and PWM.

Figure 1-2 the board of Raspberry Pi [3]

Technical Features		
Chip	Broadcom BCM2835 SoC full HD multimedia applications processor	Broadcom BCM2835 SoC full HD multimedia applications processor
CPU	700 MHz Low Power ARM1176JZ-F Applications Processor	700 MHz Low Power ARM1176JZ-F Applications Processor
GPU	Dual Core VideoCore IV® Multimedia Co-Processor	Dual Core VideoCore IV® Multimedia Co-Processor
Memory	256MB SDRAM	512MB SDRAM
Ethernet	None	onboard 10/100 Ethernet RJ45 jack
USB 2.0	Single USB Connector	Dual USB Connector
Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Audio Output	3.5mm jack, HDMI	3.5mm jack, HDMI
Onboard Storage	SD, MMC, SDIO card slot	SD, MMC, SDIO card slot
Operating System	Linux	Linux
Dimensions	8.6cm x 5.4cm x 1.5cm	8.6cm x 5.4cm x 1.7cm

Figure 1-3 Mode A and Mode B [3]

We use the model B as the system platform because it has 512 megabytes RAM, which is double than the model A and a built-in USB Ethernet adapter also included.

Instead of using the monitor and key board, we use the SSH to manage the Raspberry Pi by remote control.

In this project, the software, operating system is indispensable. The Raspberry Pi primarily uses Linux kernel-based operating systems. The most recommended operation system is the Raspbian [4]. Raspbian is a free operating system based on the Debian and optimized for the Raspberry Pi. It is easy to use as it is quite similar to the Ubuntu operating system. Based on the Linux kernel system, we install the OpenCV library [5] to develop the software program and solve some of the image processing tasks.

1.2.2 Field Programmable Gate Arrays (FPGAs)

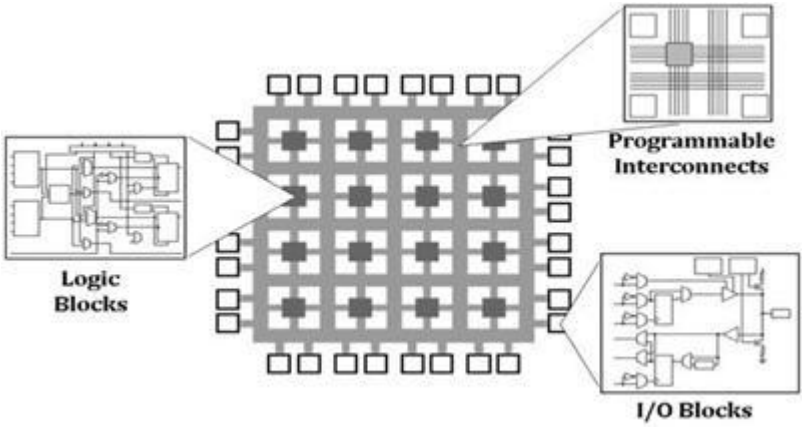


Figure 1-4 physical structure of FPGAs [11]

Field Programmable Gate Arrays (FPGAs) [11] are known ideally suit for the special-purpose logic developing. It generally consists of a system of logic blocks and some of them may have the Random Access Memory (RAM). Using the hardware description language (HDL), we could develop our own circuits which may meet special tasks. The FPGAs have large resources of logic gates and RAM blocks. They are able to implement complex digital computations and parallel design methodology. It is a good choice for us as in our project, we need to implement the much complex computation of the image processing algorithm. Usually, the image processing algorithm could benefit from the parallel processing.

1.2.3 PIF - FPGA for the Raspberry Pi

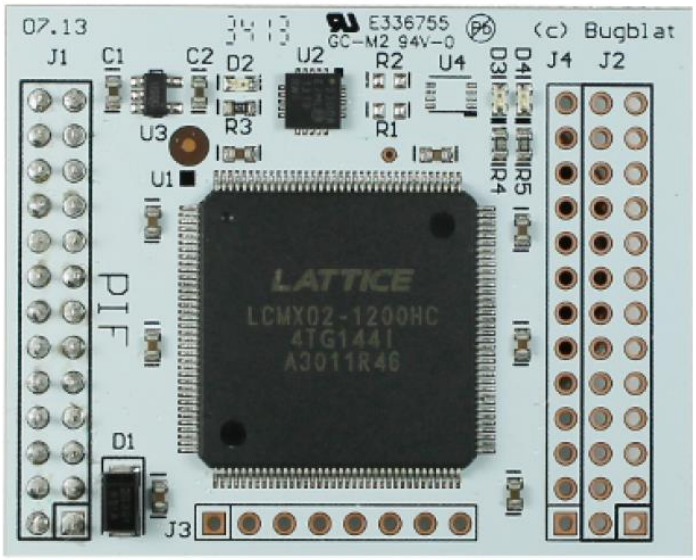


Figure 1-5 the PIF board [6]

In this project, the Raspberry Pi is just used as the platform like mini PC to communicate with the special-purpose logic, pif board [6]. The pif board (FPGA for the Raspberry Pi) is the key component for design of the hardware system. From the front words we have known that FPGAs are an ideal electronic components to develop the simpler and faster logics. The pif board gives a powerful FPGAs to the Raspberry Pi and we could know the features of the pif FPGAs from the information lists below.

- Complete FPGAs development target, no need of the FPGAs programming hardware.
- Plenty of on-chip 4 input LUTs, the Lookup tables (LUTs) are used to implement digital logics.
- Plenty of on-chip 9-Kbit SRAM blocks, which are flip flops and used for storage.
- The non-volatile on-chip flash memory is for the configuration bit stream and up to 256Kbits for user.
- Hard coded I2C, SPI, PLL and timer/counter blocks.
- No external power is needed, powered by Raspberry Pi expansion connector (P1).
- 47 pins of expansion connectors, one 26-pin (2x13-pin) connect the Raspberry Pi's P1 connector.
- Routing tracks to connect everything together.
- Red and green LEDs.

- Use provided software for injecting a new firmware.

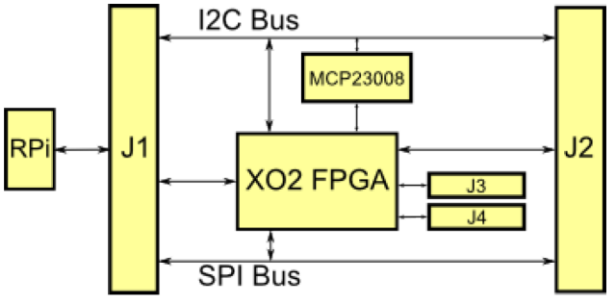


Figure 1-6 the PIF structure [6]

The most important components of the pif board are XO2 device, a Lattice Semiconductor MachXO2 FPGA and MCP23008, a Microchip I2C port expander. In our project, we use the pif-7000 board which includes a LCMXO2-7000HC-4TG144C FPGAs. The J1 are the pins connect with the GPIO ports on the Raspberry Pi which are the physical connection for the I2C or SPI bus with sending and receiving data. The firmware will use these resource in the FPGAs.

In this project, we use the VHSIC hardware description language (VHDL) [7] to describe the behavior of the logic. After the synthesis, the program of the VHDL is converted into a configuration bit stream and then injected into the FPGAs.

2 Project objective and algorithm

2.1 Project objective

The tasks of the project is going to develop common image processing filters to improve the image quality captured by the camera and also makes the auto detection by machine learning methods.

We think that the most common image filters are generally divided into two parts, linear filters and non-linear filters. Among the linear filters, Gaussian filter, Laplace filter, and derivative filter are all called as spatial filters. The spatial filters use a wide variety of masks or kernels to get the different results and give the yield smoothing, low pass filtering or edge detection. For non-linear filers, the sorting algorithm and rank order filter are most useful. The project at here use the Gaussian filter and median filter as examples of hardware implementation for the linear filters and non-linear filters.

Besides the image filters, machine learning is also a very popular method in the application of the image processing. From the OpenCV library, a lot of methods are listed. They are normal Bayes classifier, k-nearest neighbors, support vector machines, decision trees, boosting, gradient boosted trees, random trees, extremely randomized trees, expectation maximization and neural networks [5]. Because we want to develop the hardware system of the machine learning, we think that the neural networks is most suitable one. The reason is that statics methods like normal Bayes classifier and expectation maximization, k-nearest neighbors and all kind of decision trees are not suitable in the hardware. Statics methods, k-nearest neighbors are fast enough by the software implementation. No parallelization available for all of the tree structures and we know that only the feasible of parallelization could improve the speed of the algorithm by the hardware. Besides the algorithm complexity, support vector machines may be a good choice. As all the reason above, we decide to design the multilayer perceptron neural networks as an example of the machine learning methods in the hardware implementation to detect the letters. For the neural networks, it include the training and classification. We don’t need to train the neural networks on the hardware system due to the reason that the training part usually takes much of the time. We only need to use the software and powerful computer to train the neural networks before the using of it. After the long time of the training, load the weights and bias into the hardware system and then make the fast speed of the hardware implementation gives ability of the real-time classification.

2.2 Algorithm

2.2.1 Algorithm of the windowing operators

In this project, the Gaussian filter and median filter are all belong to a category known as the windowing operator [8]. Using a 3x3 window and operating on the image data gives the output. There are many kinds of the computation could be operated by the window operator and one of them is performing an operation to find the average of all pixels around the neighbors. The middle of the window is called origin and after the operation, the origin data value would be changed to the result of the operation. Figure 2-1, shows a 3 by 3 pixels window and the corresponding origin. Not only the three is able to be the size of the window, five, and seven or even larger odd numbers are also could be used. In this project, we choose the three as the window size, because it is the most common size for the image processing operator. It is enough to work correctly and efficiently on the hardware system.

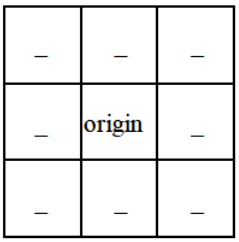


Figure 2-1 3x3 window

2.2.2 Convolution

From the mathematics respect, convolution is an integer that expresses the amount of overlap of one function g, as it is shifted over another function f [8]. In the image processing field, duo to property of discretely, the convolution is an operation to determine the value of the origin by adding the weighted values of all its neighbors together. For each input pixel window, the values in that window are multiplied by the convolution mask and added all the results, divided the sum by the number of the pixels in the window. The output of the result is going to replace the origin data and we call the output as a new modified filtered data. After the moving of the window, all of the image data are changed and a new filtered image would be generated.

Kernel is a small matrix that is used in image convolution. The size of the kernel usually as same as the image window. Different size of the kernel which contains different patterns of numbers and produce different results after the convolution operation. We also call the kernel as the convolution mask. One of the generic aspect of the convolution algorithm is that, we just need to change the value of the kernel or convolution mask, then many of the filters are available. It is flexible for us to implement different kinds of the application.

1	2	1
2	4	2
1	2	1

* 1 / 16

Figure 2-2 Gaussian kernel

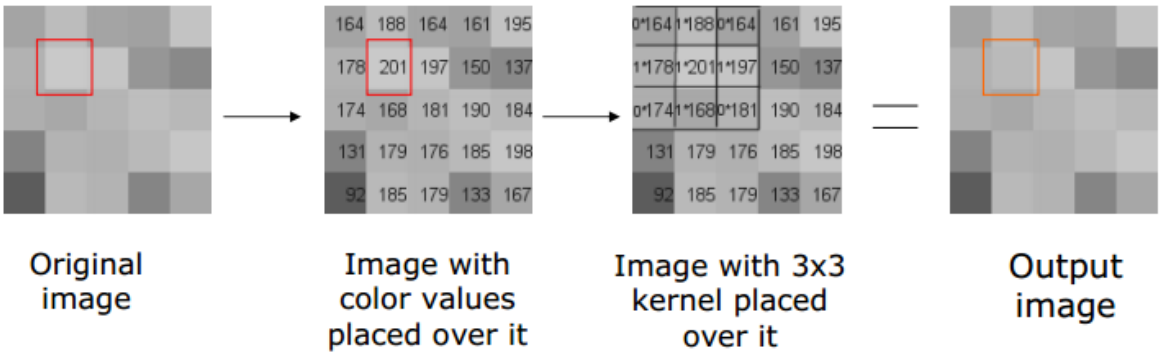


Figure 2-3 convolution algorithm [8]

2.2.3 Sorting algorithm (rank order filter)

Sorting algorithm is the algorithm that makes the elements of the data in a certain order [8]. In the image processing, it is a kind of non-linear filters which replaces each entry of the value with the nth ordered data. As a non-linear filter, the kernel or convolution mask is not used during the operation and it is usually used to remove effects on the image. When the nth order equals middle, it is called the median filter.

The main processing of the sorting algorithm is to run through the image pixel by pixel, after that, change each pixel with the nth order value among its neighbors. We use the window operator as a pattern to decide the size of the neighbors. In the most common situation, the 3x3 size of window is chosen.

```
allocate outputPixelValue[image width][image height]
   allocate window[window width * window height]
   edgex := (window width / 2) rounded down
   edgey := (window height / 2) rounded down
   for x from edgex to image width - edgex
       for y from edgey to image height - edgey
           i = 0
           for fx from 0 to window width
               for fy from 0 to window height
                   window[i] := inputPixelValue[x + fx - edgex][y + fy -
edgey]
                   i := i + 1
           sort entries in window[]
```

```
            outputPixelValue[x][y] := window[window width * window height /
2]
```

Figure 2-4 sorting algorithm

2.2.4 Machine learning and neural networks

In the computer science and statistics, people have being using the programed instructions for a long time [9]. Besides the followed explicit instruction or program, we want the computer could learn something from the data like human beings. With the development of artificial intelligence and optimization, many of the machine learning methods are available right now.

There are so many kind of approaches are used in the machine learning methods [9] and one of these is statistical models The normal Bayes classifier could give us a fast result and the expectation maximization algorithm could estimate the parameters of the multivariate probability density function other than only ones. K-nearest neighbors is often used as a method to group the examples by the non-supervised machine learning and decision tree is either for classification or for regression. A single decision tree may be used as alone or as a base class in tree ensembles which you can find them in the boosting and random trees. Support vector machines (SVM) and artificial neural networks are two popular strategies for supervised machine learning and classification. Given a set of training examples and mark each of them as one of two categories, SVM method trains the set of data and builds a model to classify the new samples into one of the category. Not only the linear classification, but also the non-linear classification could be done by using the different kernel mappings.

2.2.5 Artificial neural networks

Artificial neural networks is the machine learning model which inspirited by an animal's central nervous systems and able to recognize patterns. It is a non-linear statistical data modeling tool and helpful in the model of complex relationship between two data sets. We could use this algorithm to find patterns and the mapping relations. With the structure of neural networks, it is quite suitable for the hardware implementation as the parallelization. In the project, a generic artificial neural networks is designed and implemented for the detection of characters.

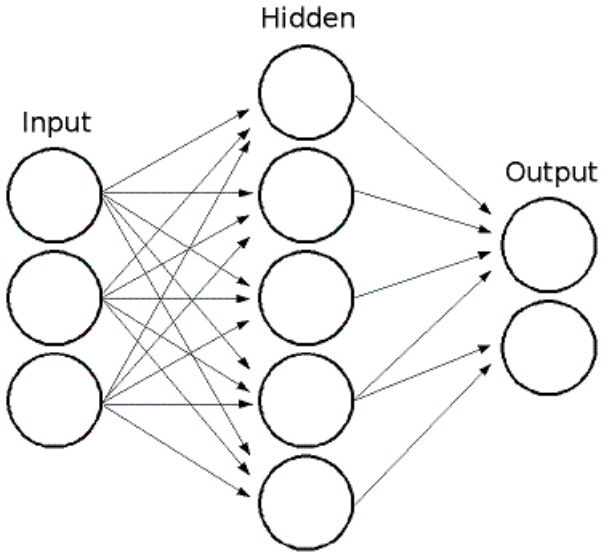


Figure 2-5 neural networks [5]

An artificial neural networks is composed of three layers and the perceptions which are connected between each other. There are input layer, hidden layers and output layer. The perceptron in each layer of the neural networks is directionally linked with the perceptron from the previous and the next layer. The perceptron in the hidden layer could be called as neuron. The link between each of neuron is always with a weight and each of them also has a nonlinear activation function. The sum of the outputs from other neutrons goes through the nonlinear activation function and gives out the output to the next layer. With the different weights and bias, the output varies and usually the sigmoid function works as the activation function.

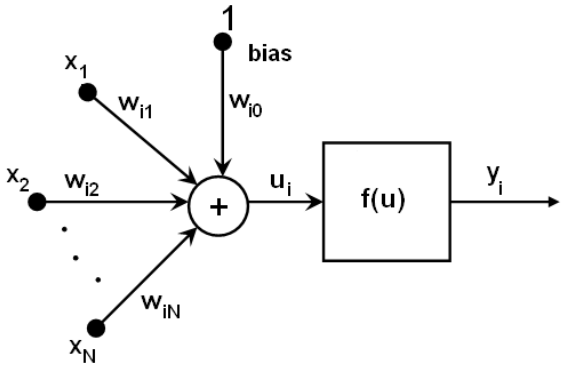


Figure 2-6 weighs, bias and activation function [5]

The mathematics formula [5] of the neural networks as above.

$$y_i = f(u_i)$$

$$u_i = \sum_j (w_{i,j}^{n+1} * x_j) + w_{i,bias}^{n+1}$$

The x_j is the output of layer n, and the y_i is the output of the next layer n+1.

Sigmoid function [5]

$$f(x) = \beta * (1 - e^{-\alpha x})/(1 + e^{-\alpha x})$$

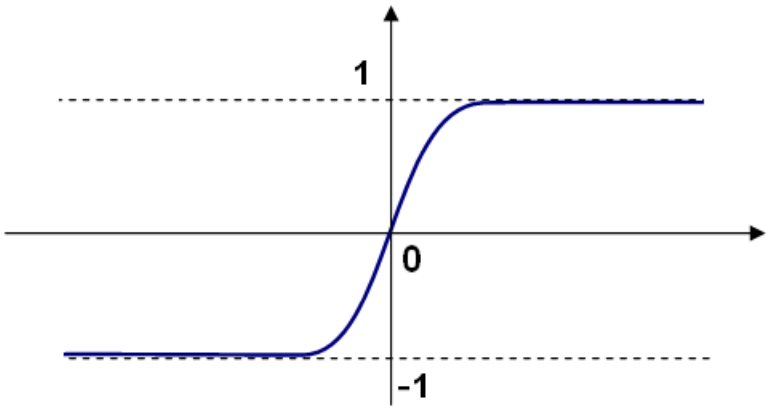


Figure 2-7 sigmoid function [5]

Symmetrical sigmoid function is the default choice for artificial neural networks.

The structure and procedure of the whole trained network works as follows.

1. Select the feature vector as input, the size of the feature is as same as input layer.
2. Sent the values as input to the first layer.
3. The weighted sum of all input connections is calculated.
4. The weighted sum goes through the activation function and gives the result as the output.
5. The output goes to the next layer as input and then repeats the procedure 2 to 4.
6. Until the output layer, the result is the final output of the whole neural networks.

In the real time application, the training is a significant time consuming procedure. The training of neural networks is often done offline and before the running time. Duo to this reason, in this project, the implementation of the neural networks has no ability of training. It is just in the running mode and idle mode. A loading component is used to load the trained parameters into the FPGAs.

3 Related works of hardware design

Prior the hardware implementation of the algorithm discussed in Chapter 2, we need to know some of the related works. These works include the hardware interface of FPGAs and other components provided by the manufactures.

3.1 FPGAs hardware

The FPGA used in this project is pif (FPGA for the Raspberry Pi) board, which has a XO2-7000HC FPGA component. All the MachXO2 FPGAs family combine a FPGA with built-in, harden control functions and on-chip User Flash Memory (UFM). All the control functions as well as the communication interface we used in the hardware design are physically located in the Embedded Function Block (EFB) [10].

1. Two I2C cores
2. One SPI core
3. One 16 bit timer/counter
4. Interface to Flash memory includes:
   - User Flash Memory
   - Configuration logic
5. Interface to Dynamic PLL configuration settings
6. Interface to On-chip Power Controller through I2C and SPI.

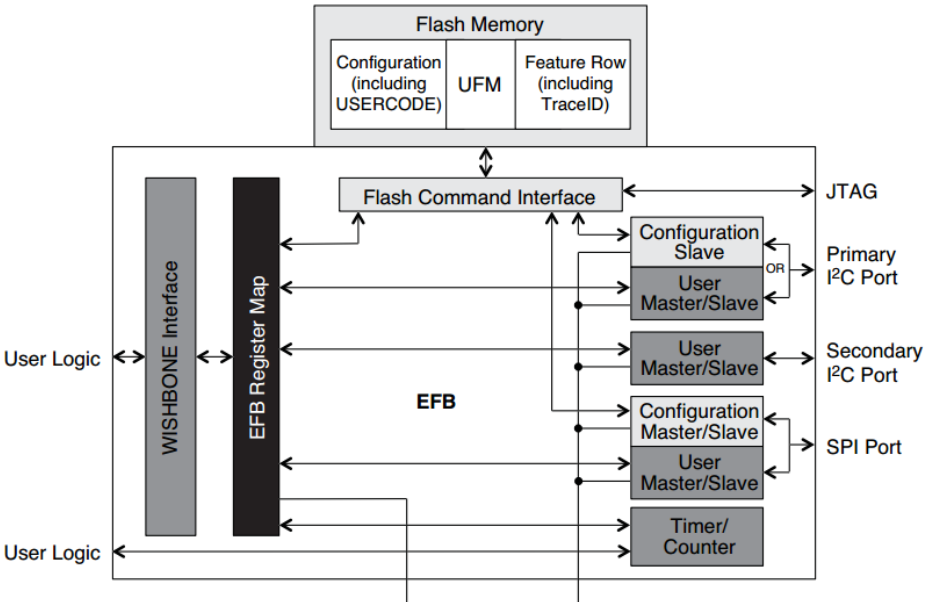


Figure 3-1 EFB of pif FPGAs [16]

3.1.1 Wishbone bus

The wishbone bus interface is an open source hardware computer bus used to communicate each parts of the integrated circuit [11]. The wishbone bus in the MachXO2 provides the connection between FPGA's user logic and the EFB function blocks. The design of the hardware should follow the wishbone standard.

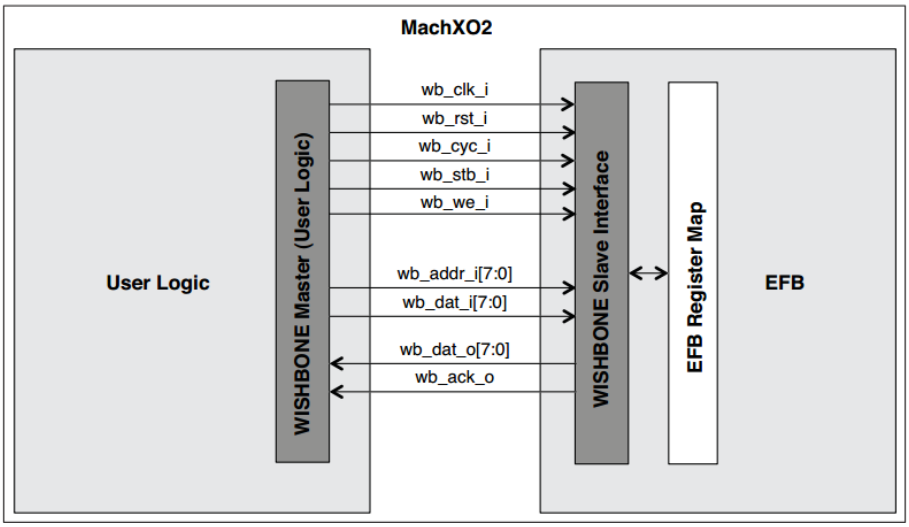


Figure 3-2 wishbone bus[16]

3.1.2 I2C vs SPI [11]

Duo to the structure of the FPGAs, two I2C and SPI cores are available as the hardware interface which communicate with the Raspberry Pi. The I2C is much slower than the SPI as the transfer speed is just 500 kHz. The I2C is a half-duplex, synchronous and multi-master bus. From the speed consideration, in this project, we use the SPI bus as the main interface and the details of the bus are described as below.

SPI bus is a synchronous serial data link and operates in the full duplex mode. There are master/slave modes in the SPI bus, which means when the device is in the master mode, it initiates the data frame and the transfer clock. At meanwhile, the device in the slave mode just receives the data from the master. Multiple slaves are available when the slave select signal (SS) are set. SCLK signal works as SPI clock. MOSI and MISO are two signals dealing with data. In the master mode, MOSI is out and MISO is in. For the slave mode, it is just in an opposite direction.

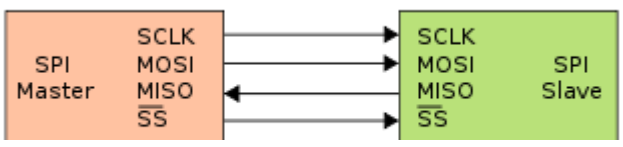


Figure 3-3 SPI bus

The Raspberry Pi acts as the master in this project, and the pif works as a slave. We could configure the speed of the SPI by the Raspberry Pi software program. After the experiment, we configure it at 7.8125MHz which gives us the fastest speed with a stable results.

After the principle of the SPI introduction, we need to know how about it on the target FPGAs hardware. In order to get the data from the Raspberry Pi, we need to access the EFB Register by through the slave wishbone bus. The figure 3-4 at below give us a whole look about the communication.

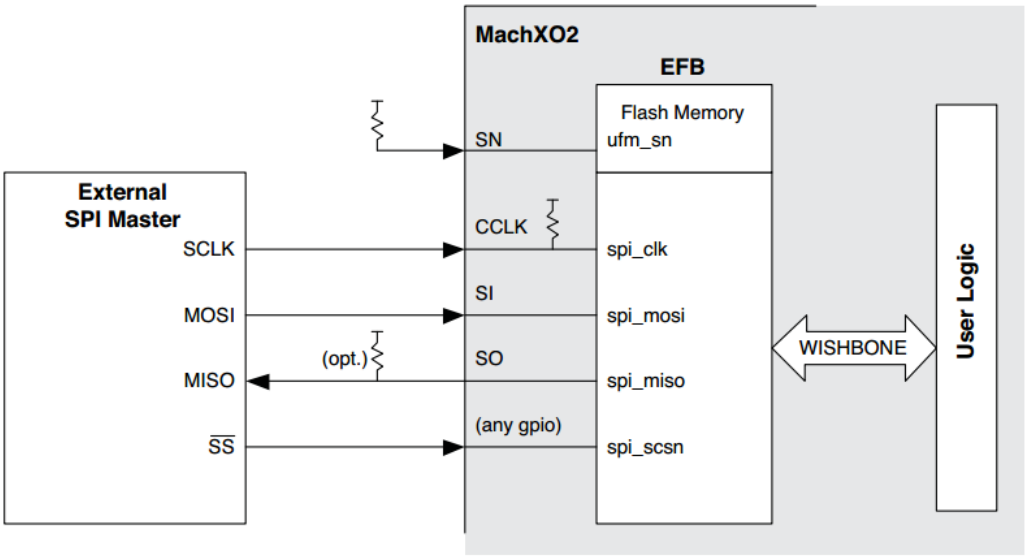


Figure 3-4 the master and slave devices [16]

The Raspberry Pi is the SPI master at here and EFB is the function blocks inside of the FPGAs works as the slave device. Using wishbone bus, the user logic which is the implementation of each image processing algorithm or neural networks receives the data from outside and sends them back after the calculation.

3.1.3 Internal oscillator (OSCH)

The internal oscillator [12], [13] is the clock source signal for the user logic design. In this project, we don't need any global clock source

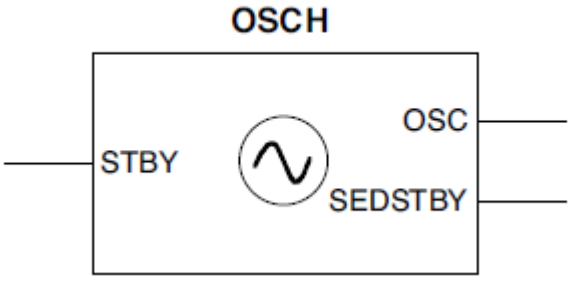


Figure 3-5 OSCH clock component [16]

from outside. For this device, only some constant frequency are supported. They are 2.08, 2.15, 2.22 … 66.5, 88.67, and 133.0. We set the frequency in 38 Mhz.

3.1.4 Memory model

Besides the interface and internal clock, the memory is also a useful component in the design. The manufactures of the FPGAs, Lattice semiconductor provides us the IPexpress [13],[14] to generate memory used in the FPGAs. The IPexpress GUI allows us to specify the memory type and size. In this project, a dual port RAM is generated in the VHDL and then integrated into the whole system.

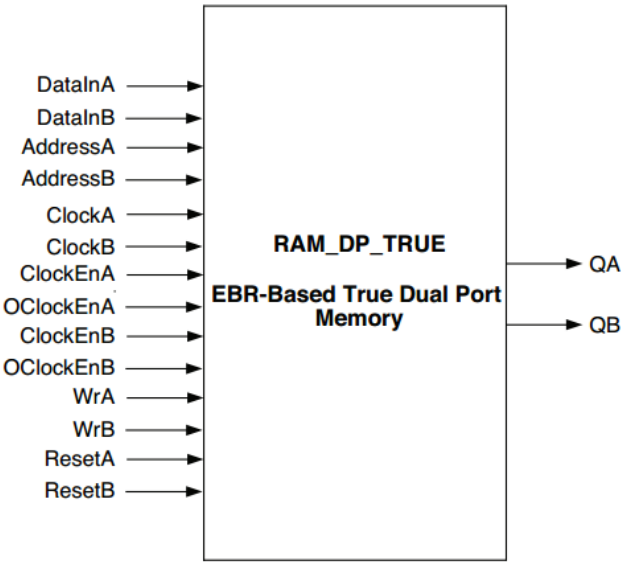


Figure 3-6 dual port ram component [16]

After all the components of the hardware provided by the development tools, the next chapter is going to talk about the hardware implement of each algorithm mentioned in the chapter 2.

4 System design and implementation

As the most designers in the hardware implementation, one of the main concepts is to facilitate code reusability and develop a common hierarchy. When we are going to the design and implementation of the hardware system, we need to follow the step of these principles [11]

1. Apply hierarchy to the design, at the most highest level of the system, we should always use the larger function blocks which is composed of one or more hierarchical blocks. The key point is to increase the abstraction.
2. For each of the functional blocks, apply register transfer level (RTL) description. The RTL description is the logic that define the behavior of each block.
3. Partition the hardware system by the nature performed operations. Usually, the system has two parts of data and control. Data path unit is for data processing and control unit is for the signal controlling.

As a result, each subsystem should be synthesized independently which means that the circuit is isolated as independent modules. In this project, each of the algorithm in our design consists of several subsystems. All of the hierarchies will be described as below.

At first, some of the components are reused in the design. The independency of the components make each hardware algorithm implementation easier.

4.1 Global sets

Within each of the whole system, the reset signal is used for the global resetting of each components [15].

```
--signal reset
reset0: reset
port map(clk => clk_sig,
rst => rst,
sda => SDA
);
```

Figure 4-1 reset component

As we don't use the I2C bus for data transferring, the SDA signal is used at here as the input for the reset signal. The CLK is the system clock signal and the RST signal is the output to reset whole hardware system. When the RST signal sets to 1, the whole system will do reset and for 0, nothing happens.

4.2 SPI and OSCH

SPI component is the control unit functional block dealing with the data receiving and sending. CLK signal is always as the system clock. There are two 8 bit signals which are WRITE_DATA and READ_DATA. The WRITE_DATA is the data frame received from the SPI bus and READ_DATA is the frame which are sent the back to the SPI bus. When the WRITE_REQ signal equals 1, the WRITE_DATA's value would be write into the logic and the NEW_DATA signal changes to 1. Four of the SPI signals are connected to the EFB SPI component, which is generated from the IPexpress and works as the data unit during the SPI transferring.

```
spi0 : spi2
    port map(   clk=> clk_sig,

                write_data => tx,
                write_req => write_req,

                read_data => rx,
                new_data  => new_data,
                spi_clk   => spi_clk,
                spi_miso  => spi_miso,
                spi_mosi  => spi_mosi,
                spi_cs    => spi_cs
            );
```

Figure 4-2 SPI component

The OSCH [12] component is the component for the system clock generation. The NOM_FREQ is the string for setting of the frequency and here we set it at 38Mhz. The OSC works as the output and gives the system global clock.

```
-- clock
    OSC0: OSCH
      Generic Map (NOM_FREQ  => "66.50")
      Port Map (STDBY => '0', OSC => clk_sig,  SEDSTDBY => open);
```

Figure 4-3 OSCH component

4.3 Sorting and median filter

The median filter hardware hierarchy has four components. The first three are the common reused components for the reset, interface and clock generation. However, the last one is the algorithm part and also it is the most important part.

```
entity main is
…
architecture rtl of main is
---------------------------------------------------
component spi2
 …
end component;

component reset is
…
end component;

component OSCH
……
end component;


component sort_filter
…
end component sort_filter;

end architecture rtl;
```

Figure 4-4 sorting filter main hierarchy

Sorting filter component, figure 4-5, shows the genetic and the interface of the sorting filter.

```
entity sort_filter is
    generic (
    DATA_WIDTH: integer:=8;
    order: integer:=5;
    num_cols: integer:=128;
    num_rows: integer:=128
    );
    port (
    clk : in std_logic;
    rst : in std_logic;
    DataIn : in std_logic_vector(DATA_WIDTH-1 downto 0);
    DataOut : out std_logic_vector(DATA_WIDTH -1 downto 0);
    DV : out std_logic
    );
end sort_filter;
```

Figure 4-5 sorting filter component

DATA_WIDTH represents the width of the input and output data, the default value is 8 bit in this project for each of the pixel value from 0 to 255. The order configures the output value, as the 1 means the first, the 5 means the median and 9 means the last. NUM_COLS and NUM_ROWS define the size of image which is going to process. For the interface, CLK and RST are the common signal as before. DataIn and DataOut are two signals dealing with data input and output. The DV output signals the validation of the data output. When the DV signals 1, the output data is the valid data and then sent the data back to the SPI bus at same time.

After the interface the sorting filter, there are also details as blow. Figure. 4-6 is the structure of the sort_filter component.

```
library ieee;
use ieee.std_logic_1164.all;

entity sort_filter is
……
end sort_filter;

architecture rtl of sort_filter is

component sort_3x3
……
end component sort_3x3;

COMPONENT window_3x3
……
END COMPONENT window_3x3;

component rc_counter
……
end component rc_counter;

begin
    sort_filter_0 : process(clk, rst)
end process;
end rtl;
```

Figure 4-6 sorting filter hierarchy

Three components are used in the sorting filter algorithm and the first one is the 3x3 pixel window generator, the window_3x3 block. As discussed above, the sorting algorithm must first sort the pixel value in a window by a rank order. The image data works

as stream and input the data into the system one by one. The component consists of 2 FIFO buffers and 7 registers.

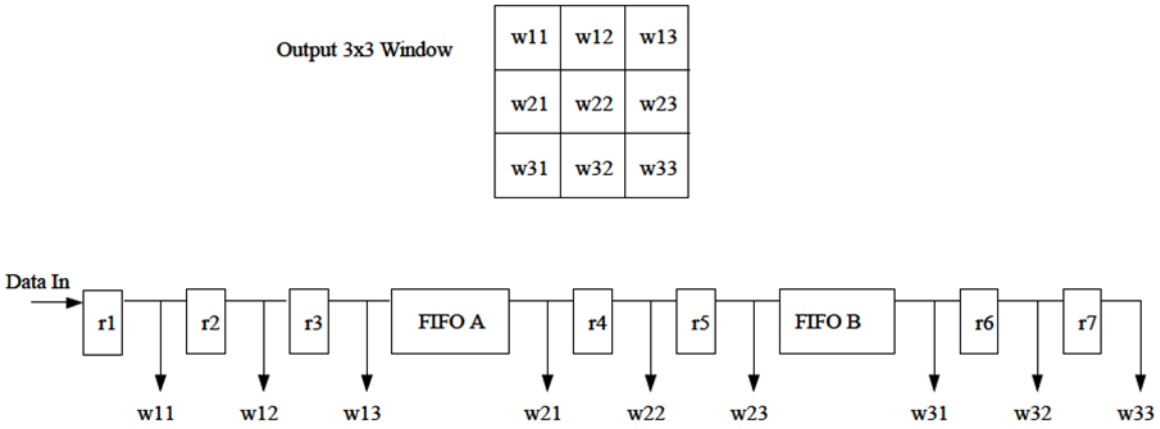


Figure 4-7 window operator [17]

After the window generation, in each clock, 9 values as the output data are going to the next stage and ready to be sorted. The use of the next component, sort_3x3 component, gives us the result of the sorted pixels. It is a system of hardware compare and

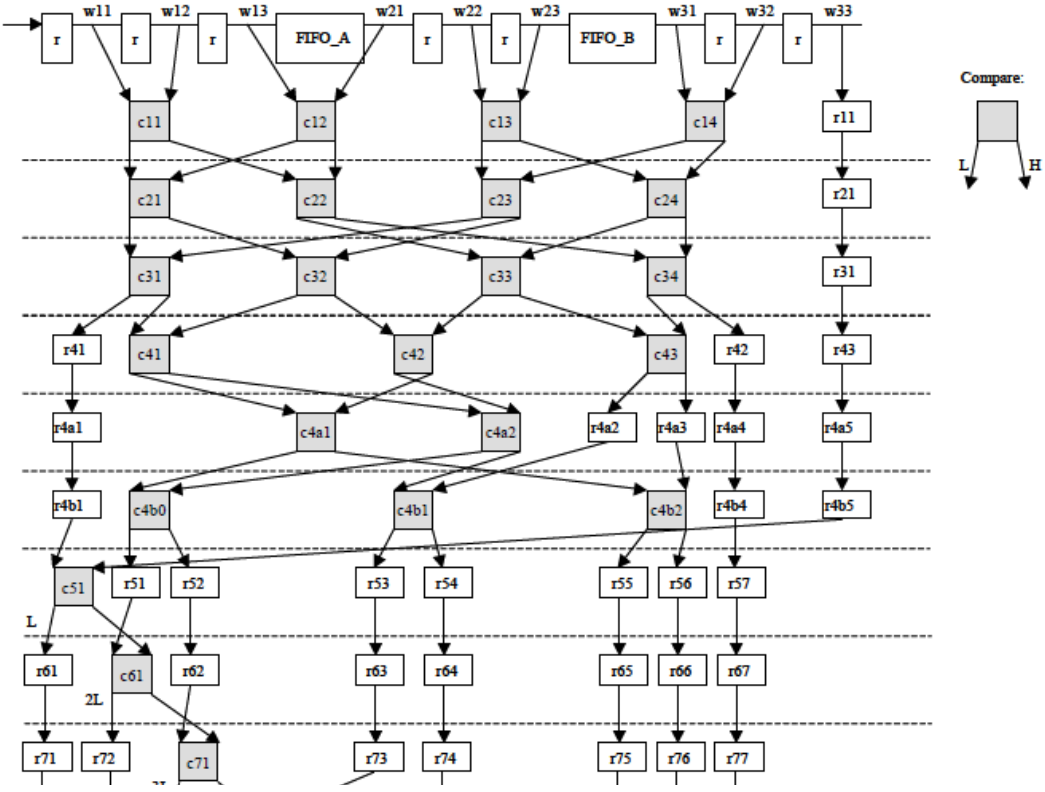


Figure 4-8 sorting algorithm implementation [17]

sort units, which make the pixels in the window as an ordered list for use in the later procedures.

The sorting component is implemented as the structure in the Figure 4-8. It is just composed of registers and compares. The result is a list in a sorted rank after an initial latency and produced on every clock cycle.

As we known, the output of the image should be as same size as the input one. For this implementation, the input is the pixel stream and the output as well. Lack of the counter block rc_counter for the column and row, the output cannot be constructed into a normal image. Not only for the size of image, but also for the window operator. In this implementation, two counters are used for the column and row which tell the algorithm when the border start. The counter counts the movement of the window and when it meets the border, it just set the border pixel to 0 for the output. This component is also able to reuse and needed in the convolution algorithm as well.

4.4 Convolution and Gaussian filter

Compare to the design of the sorting algorithm, the convolution algorithm is much more difficult. It is because that the sorting algorithm just uses the registers and compares in the implementation. At meanwhile the convolution needs more calculation like adders, multipliers and dividers. The more complex algorithm is, the more calculation time costs. As many people have already tried the algorithm on each kinds of the FPGAs, in this project, we think implement the convolution algorithm is an achievable goal.

Another different part between the algorithm of the convolution and sorting is that the data value is not always positive anymore. The kernel of the convolution may be negative although the input is as same as the median filter before. In a proper convolution algorithm, the mask often consists of negative numbers. Duo to this reason, the VHDL had to be able to handle the data in the signed type. The first bit of the data is used to present the sign bit with the 0 for positive and 1 for negative.

The addition and multiplication are just using the simple + and * signs in the VHDL code. The simple signs have already meet the needs of the calculation and the synthesis tool could map them into an efficient hardware designs. But for the computation of dividing, there are no sign for this kind of the calculation in the VHDL code. During the last computation of the convolution, the sum of weighted data would divide the size of the window. Instead of the device-specific parameterized modules for the dividers,

the shift operation is an optional result. As only possible with powers of two, usually the 8 is used for the divide of 9. In this project, shift of 4 bit is exactly for the Gaussian kernel.

Figure 4-9, shows a highest hierarchy of components.

```
entity main is
…
architecture rtl of main is
-----------------------------------------------------
component spi2
 …
end component;

component reset is
…
end component;

component OSCH
……
end component;


component conv_3x3
…
end component conv3x3;
begin
…
end architecture rtl;
```

Figure 4-9 convolution filter main hierarchy

As same as the sorting filter, the SPI, RESET and OSCH are the reused component for the interface of the data transferring, components resetting and global clock generation. The conv_3x3 is the algorithm part and give the result for image processing.

4.5 Convolution package and component.

In the convolution algorithm, different convolution masks give us different results. The file conv_3x3_pkg.vhd defines the kernel value, the data width and the image size. Constant k0 to k8 are the nine values for the convolution mask and DATA_WIDTH presents the 8bit grayscale image pixels from 0 to 255. The NUM_COLS and NUM_ROWS are set as 640 and 360 for the input image size.

The interface of the convolution algorithm is as same as the sorting algorithm. The CLK and RST are two signals for clock cycle and reset control. The DataIn and DataOut are both signals for input and output. With the DV signals 1, the output data is recognized as valid data which is sent back to the SPI bus.

```
entity conv_3x3 is
    port (
    clk : in std_logic;
    rst : in std_logic;
    DataIn : in std_logic_vector(DATA_WIDTH-1 downto 0);
    DataOut: out std_logic_vector((DATA_WIDTH*2) downto 0);
    DV     : out std_logic
    );
end conv_3x3;
```

Figure 4-10 convolution component

For the algorithm component, the Figure 4-11 at below shows a graphic representation. It is also include the component of the window generator and pixel counter for the input

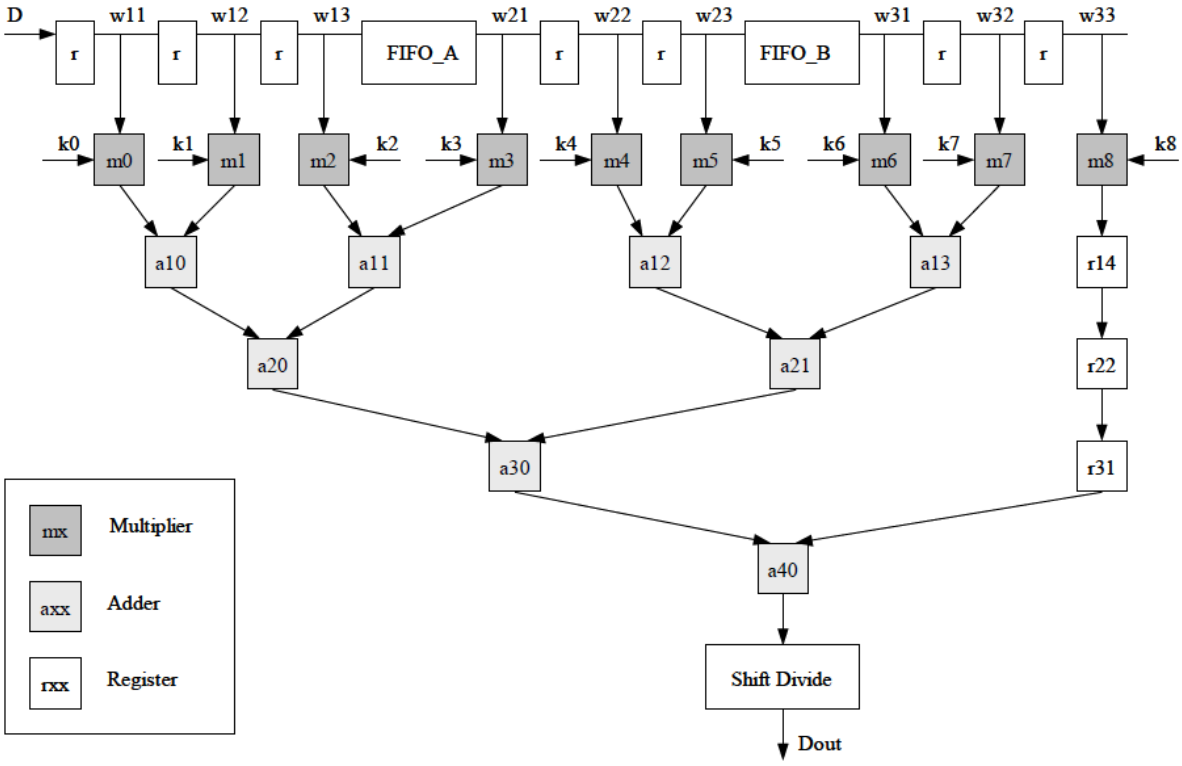


Figure 4-11 convolution algorithm implementation [17]

image stream and they work as same as the algorithm before. After the window generator, all the output are going to the convolution calculation and the output gives the result.

4.6 Neural networks

In this project, a generic neural network hardware implementation is also included [18] [19]. The neural networks is a very popular machine learning method and has a great generalization ability. No learning phase is integrated as we need the system works in the real-time. The structure of the neural networks is an arbitrary structure and with no limitation for the inputs and outputs. The following pages describe the multilayer perceptron neural networks which is implemented in this project using the VHDL codes. Figure 4-12 is the graphic representation for the whole neural networks as below.

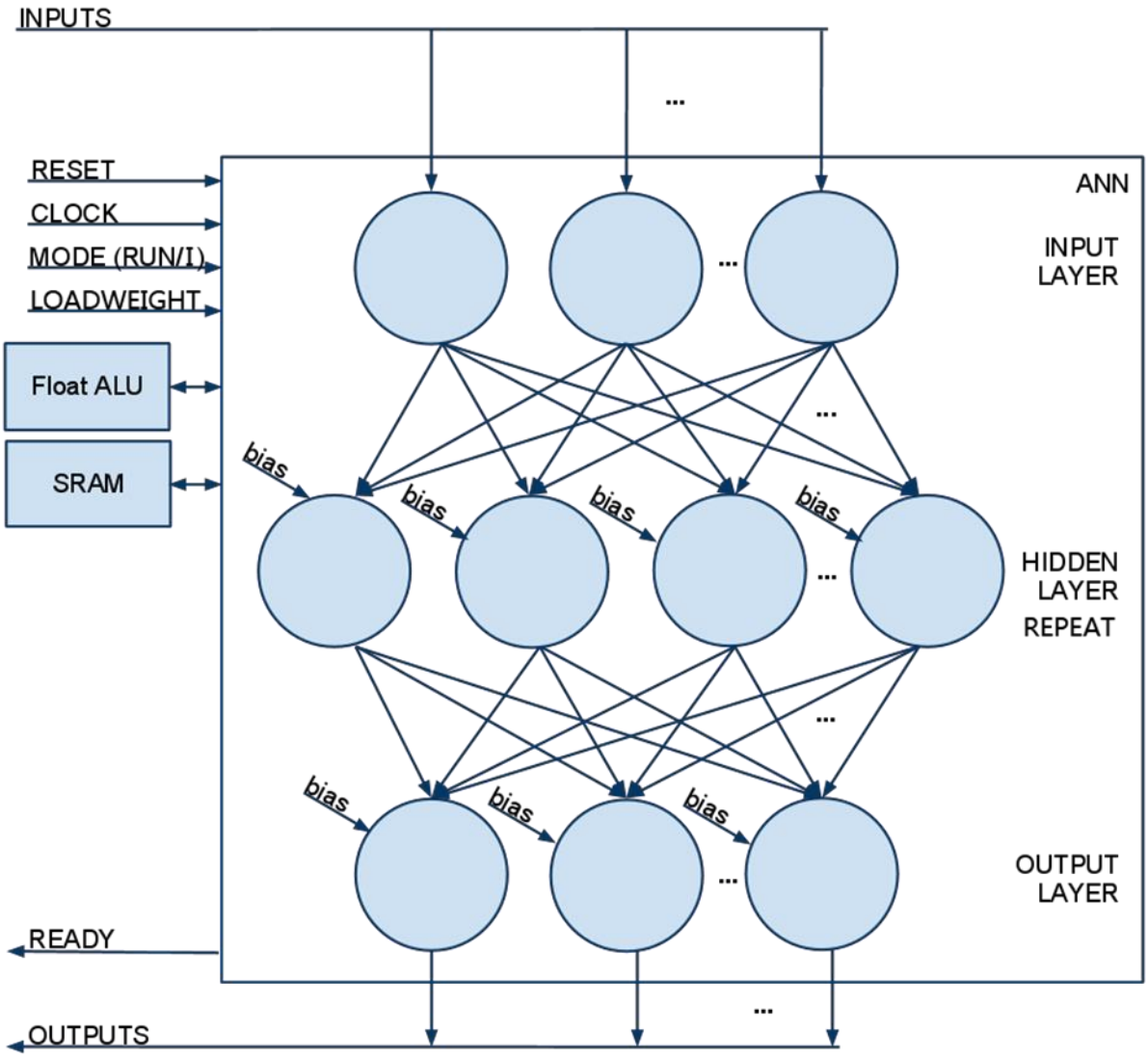


Figure 4-12 neural networks architecture

Before the details about the hardware implementation, all of the parameters are defined in the file of mlp_pkg.vhd. Parameters are flexible and global which would meet different kinds of the neural networks structures.

```
package mlp_pkg is

    constant N_I : integer := 16; -- number of perceptrons at input layer
    constant N_H : integer := 1072; -- total number of weights and bias --
changed
    constant N_O : integer := 16; -- number of perceptrons at output layer
    constant N_L : integer := 1;  -- number of hidden layer

    constant MAX_H : integer := 40; -- max number of perceptrons at each
hidden layer

    type int_array is array (0 to 1) of integer;
    constant numP : int_array := (32,0);     -- each number of perceptrons
at hidden layer

end mlp_pkg;
```

Figure 4-13 neural networks parameters

The comments explain the meaning of each constant and all of them are used in the whole system of the neural networks. You can change the value as you like to meet the needs of your self's neural networks structures. The example at here means the input of the neural networks has 3 layers, 16 inputs for the input layers, 32 neutrons in the hidden layer and 16 outputs for the output layer.

The first part of the neutral networks design is about the peripheral components. They are the data transferring units, computation units and storage components. The interface of the neural networks with Raspberry Pi is as same as the image filters used in the above chapter, but the data width of the neural network is 32 bit other than 8 bit. Because of the higher accuracy of the data calculation, 8 bit for the image pixel is not enough anymore and the data type has changed to the type of 32 bit floating-point. Duo to the limitation of the data width of the FPGAs hardware, only 8 bit is transferred at each time. The component "receiver" in file of receiver.vhd receives every 4 8bit data frames and then generate a new 32 bit data frame for the input of the neural networks. CLOCK and RESET are as same input as before. The NEW_DATA is the signal that indicates the data received from the SPI bus with an 8 bit value. The DATA_DONE means a new 32bit data frame is going to send to the logic with the value in the DataOut. The count4x is the counter, gives the number of the 32 bit data frame generated.

```
entity receiver is
    port(   clock : in std_logic;
            reset : in std_logic;
            new_data : in std_logic;
            data_done : out std_logic;
            count4x  : out integer;
            DataIn   : in  std_logic_vector(7 downto 0);
            DataOut  : out std_logic_vector(31 downto 0)
        );
end receiver;
```

Figure 4-14 receiver component

The component of output is in another way. After the processing, the 32bit data frame need to be divided into 8bit by 4, and then sent back to the SPI bus. The only different part is that, the output component works should always after the finish of computation as the input MLP_DONE signal is 1. The OUTPUT_NEW signal means the output of the data is valid for the SPI bus, which needs to write the data first and then could read the valid data. The DataIn is the output of the neural networks and DataOut gives the 8 bit result data back to the SPI. When the OUTPUT_DONE is high, the output of the result is finished.

```
entity output is
    port(   clk : in std_logic;
            rst : in std_logic;
            mlp_done : in std_logic;
            DataIn : in  float_vector(N_O -1 downto 0);

            output_new : OUT std_logic;
            DataOut  : out std_logic_vector(7 downto 0);
            output_done : out std_logic

        );
end output;
```

Figure 4-15 output component

The ALU component and RAM component are two data operation components. In this project, a new floating-point calculation library is introduced. The library of the float point has the ability of addition, subtraction, multiplication, division and exponent computation. The library is called FPLibrary [20] which is a library of prameterizable arithmetic operators for “real” numbers. It supports both floating-point and logarithmic number systems, is really helpful for this project. The FLOAT_ALU component in the fp.vhd works as the control unit to different kinds of the calculation.

```
entity float_alu is
    port (
    rst, clk : in std_logic;
    a,b  : in  float;
    c    : out float;
    mode : in  float_alu_mode;
    ready : out std_logic
    );
end entity float_alu;
```

Figure 4-16 float_alu component

The A and B are the inputs of the two variables and the C is the output. The mode refers to the operation as adder, multiplier, divider or exponent. When the floating ALU component is working in the ready mode, the data are valid for the calculation as the READY signal equals 1.

```
entity sram_dp is
    port (
        resetA : in std_logic;
        resetB : in std_logic;
        clockA : in std_logic;
        clockB : in std_logic;
        addrA : in sram_address;
        addrB : in sram_address;
        inputA : in sram_data;
        inputB : in sram_data;
        outputA : out sram_data;
        outputB : out sram_data;
        wrA  : in std_logic;
        wrB  : in std_logic;
        readyA : in std_logic;
        readyB : in std_logic
    );
end entity sram_dp;
```

Figure 4-17 dual port ram component

RAM component is the place to store the data as inputs, weights and bias of the neural networks. Due to the large number of data used in the neural networks, the Dural-Port-RAM is a good choice. The data depth of the RAM is 12 bits and each data is 32 bits floating-point. The other signals work as the controllers for the write and read purpose.

The port A of the RAM is used as the writing port for the data received from the Raspberry Pi and the port B is the port for reading, because the neural networks need to read the weights and bias at each time of calculation. The Port A and Port B are independently and both of them refer to the same area of the RAM. The code of the RAM is generated from the IPexpress tools. The size of the RAM is limited and more of the space will be exceed the total resource of the FPGAs.

```
entity loadWeight is

    port (
    rst : in std_logic;
    clk : in std_logic;
    new_data: in std_logic;
    inputs : in sram_data;

    checker : in std_logic;
    inputNumber : in integer;
    weightNumber : in integer;
    weight_done : out std_logic;

    sram_addr : out sram_address;
    sram_input : out sram_data;
    sram_output : in sram_data;
    sram_mode :  out std_logic;
    sram_ready : out std_logic

    );
end entity loadWeight;
```

Figure 4-18 load the weight and bias into RAM

The loadWeight component is also an important data transfer controlling component. It does the job that load the weights and bias into the RAM at the first time when the system starts. NEW_DATA indicates the new input data and inputs are the data from the 32 bits data generation components. The INPUTNUMBER + WEIGHTNUMBER is the total number of the data. The CHECKER signal with high means it is the first time of working and the low means the other time. Besides that, 5 signals are the interface to control the data storage with the RAM component.

```
entity mlp is
    port (
    rst : in std_logic;
    clk : in std_logic;
    mode : in ann_mode;
    inputs : in float_vector(N_I -1 downto 0);
    outputs : out float_vector(N_O -1 downto 0);
    ready : out std_logic;
    done  : out std_logic;

    float_alu_a : out float;
    float_alu_b : out float;
    float_alu_c : in  float;
    float_alu_mode : inout float_alu_mode;
    float_alu_ready : in std_logic;

    sram_addr : out sram_address;
    sram_input : out sram_data;
    sram_output : in sram_data;
    sram_mode : INOUT sram_mode;
    sram_ready : in std_logic
    );
```

```
end entity mlp;
```

Figure 4-19 MLP neural networks interface

After all peripheral parts, let's go to the most important one, the neural networks algorithm component. The MLP component works as the controlling unit connected with ALU and RAM. It reads the data from the RAM and use the ALU unit to calculate them.

FLOAT signals and SRAM signals are just the interface with the ALU and SRAM. The mode of the MLP are idle or run. No inputs are needed as the input of the neural networks are just reading data from the RAM. OUTPUTS are the results for the neural networks with number of N_O. The WEIGHT_DONE is given from the loadWeight component with meaning of that all the data are ready to use. When the MLP neural networks is finish, the DONE signal changes to 1.

After the introduction of the interface, we need to know the details inside. The Figure 4-20 at below tells us the state machines of the initialization in the neural networks which gives the start of the whole system.

When the MLP component receives the signal of finish for the weights loading and RUN for the neural networks, it goes to the INIT state with the loading of inputs.

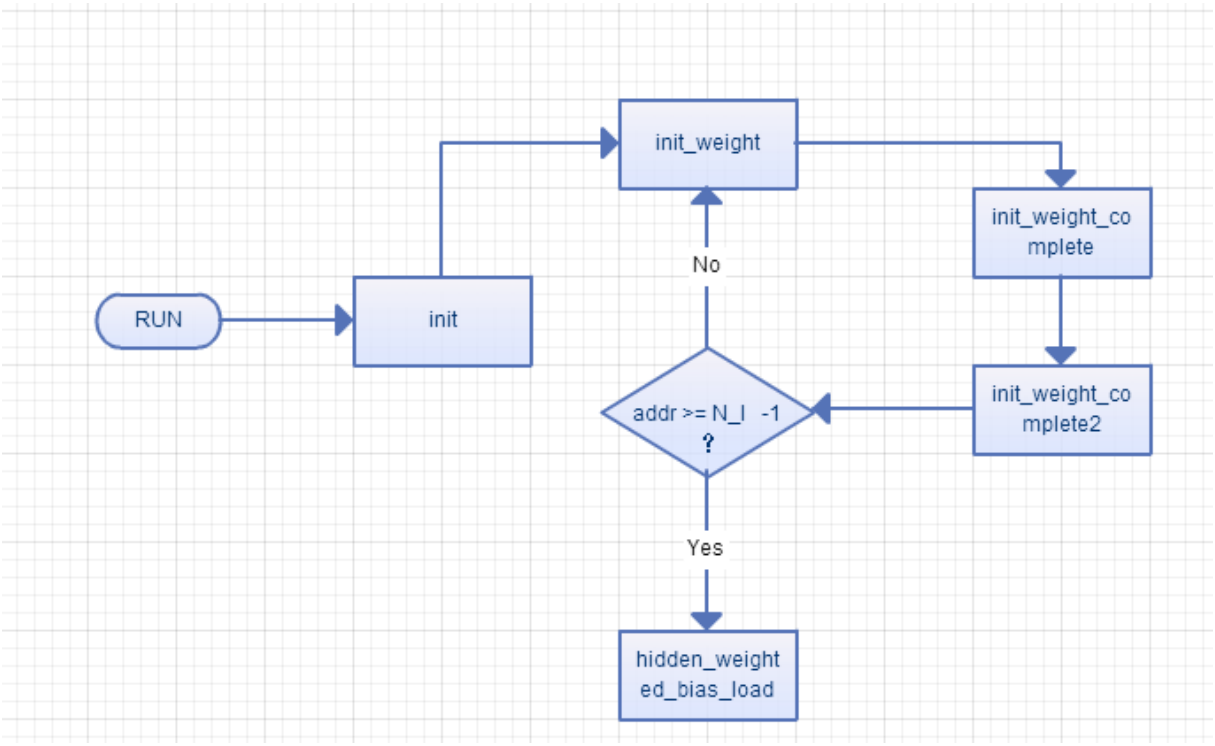


Figure 4-20 initialization of the neural networks, read inputs from RAM

After the global setting, the system reads the data value from the RAM which has the inputs of the neural networks. The state of INIT_WEIGHT_COMPLETTE and INIT_WEIGHT_COMPLETE2 are states for the delay of the RAM reading. When one of the input reading is finish, the read address will come to the next one. If the address of the reading is exceeded the input number, the state goes to the next state to load the weights and bias.

At right now, the inputs procedure of neural networks have been done and we need to get the weights and bias data for the calculation. Each of the load use two state with

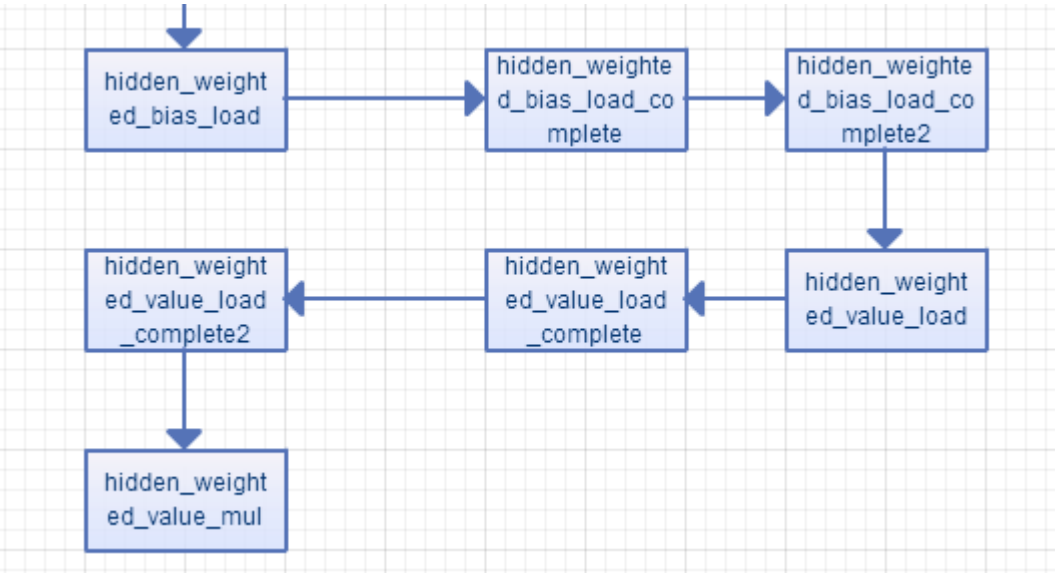


Figure 4-21 read bias and weight for each neutron

one for the operation and another one is for the delay of the RAM reading. The bias load at the first and then value of the weights for the next computation.

The computation is a complex procedure. The weights multiply with the input and then add the bias or other neutron results. The four states are listed here as the name imply.

hidden_weighted_value_mul,

hidden_weighted_value_mul_complete,

hidden_weighted_value_add,

hidden_weighted_value_add_complete,

If all the neutron result have already done with the addition, then the result goes to the calculation of the activation function. As the code here, if it is the first hidden layer and

the input neutron is equals the last input, the state will go to the calculation of the activation function. However, if it is not, just repeat the next input. For the other hidden layer is the same that just check whether is it the last neutron of the front layer.

```
if (numL = 0 ) then
        IF (i = N_I - 1) THEN
            state <= hidden_sig_neg;
        ELSE
            i <= i + 1;
            state <= hidden_weighted_value_load;
        END IF;
    else
        IF (i = numP(numL-1)- 1) THEN
            state <= hidden_sig_neg;

        ELSE
            i <= i + 1;
            state <= hidden_weighted_value_load;
        END IF;
    end if;
```

Figure 4-22 to the next input neutron or to the activation function

```
hidden_sig_neg,

hidden_sig_exp,

hidden_sig_exp_complete,

hidden_sig_add,

hidden_sig_add_complete,

hidden_sig_div,

hidden_sig_div_complete,

hidden_sig_double,                  -- Symmetrical sigmoid

hidden_sig_double_complete,

hidden_sig_sub,
```

hidden_sig_sub_complete,

The states in the above are just doing the activation function, which include the sigmoid function calculation. The function used here is the symmetrical sigmoid function as same as the OpenCV's implementation. At the end of the "hidden_sig_sub_complete" state, it is the important time to check that which layer of the neural networks is calculating right now. If the layer at here is not the last one in the hidden layer, the state will

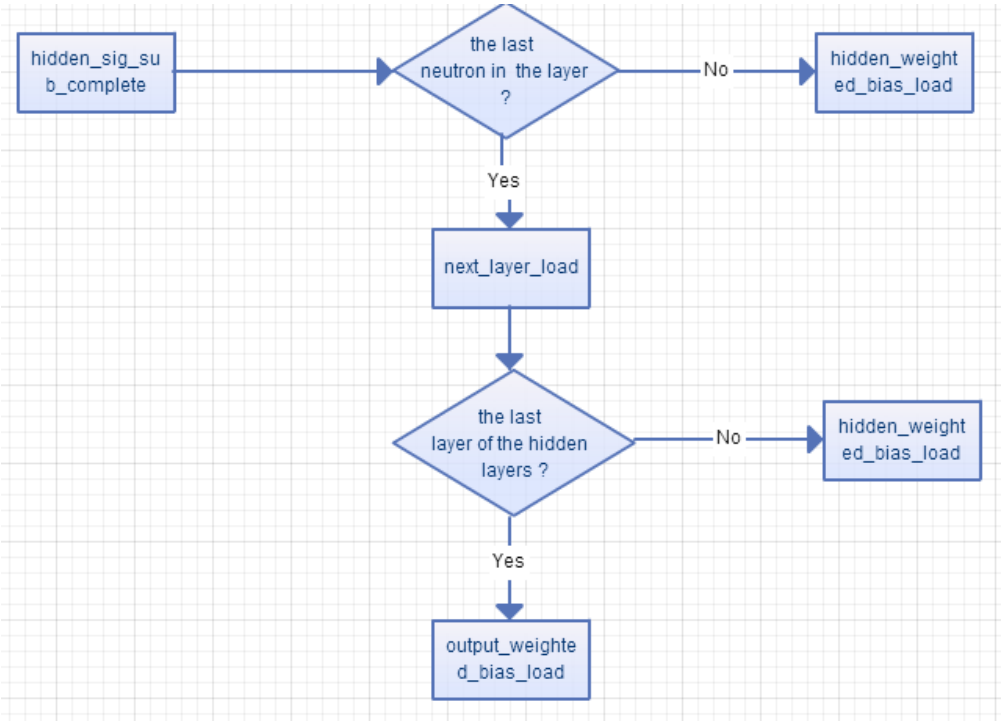


Figure 4-23 to the next layer or the output layer

go to the next layer and repeat the procedure of the calculation. On the other hand, when the layer at here is the last one, the state will skip the repeat procedure and finish the neural networks by the calculation of the output layer.

The output part of the neural networks has the same procedures as the hidden layer. The first thing is also to load the bias and weights for each neutron and add all of them. After the addition, do the activation function on each of the weighted sum result and the result is the final output for the whole neural networks. With a time of delay, when the MLP_DONE signal gives the high, the output of the computation will be treated as the valid result. For the next time of the input, the situation is the same but without the weights and bias loading again. It because that the new inputs are loaded into the system and the others are the same values which would be used again.

5 Experiments and results

In order to demonstrate the functionality of the algorithms, several examples application are used as here. They are median filter, Gaussian filter and the MLP neural networks. As the filters are simple of the complexity, running on the FPGAs has a successful result and the real performance is easily to show. However, the MLP neural networks has much more complexity in the algorithm design and at the end the synthesis tool shows that, nearly double of the Look-Up Tables (LUTs) are required which is much larger than the resource of FPGAs. Therefore, only the simulation result are showed at here.

5.1 SPI interface

The first part of the experiment is to check the speed of the SPI used in the project. The speed of the SPI determines the speed of the algorithm because all of the data need to be transferred in to the FPGAs.

Figures 5-1 and Table 5-1 show that the comparison between the original image and the image received from the SPI bus without any computation. The Table 5-1 is the size of the image and the maximum speed we could achieve. The higher transfer clock could make the speed faster, but also add more errors in the data stream when you read them back.

640x360, grayscale, original

SPI interface result

Figure 5-1 the original image and the SPI transferred result

Table 5-1 the performance of the SPI bus

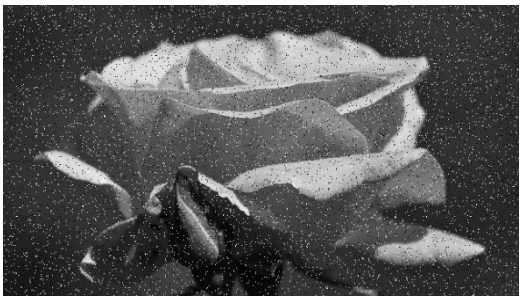
input size	time / speed
230.4KB	0.120s / 1.831MB/s

5.2 Median filter

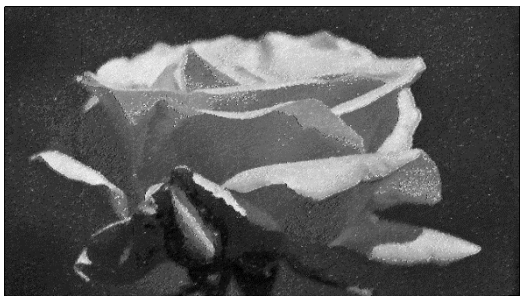
Figure 5-2 shows comparison between the FPGAs and OpenCV implementation of the algorithms for the median filter. The original image is the second image with the pepper and salt noise.

The result of the image implementation is not as good as the software implementation duo to the complex design of the hardware system but still acceptable. As much of the time is used for the transfer in the hardware, the time consuming is near 3 times than the OpenCV's result. If we just look at the image processing part of the hardware, the time is as same as the pure software from the Table 5-2 as below.

640x360, grayscale, original



640x360, grayscale, noised



640x360, grayscale, FPGAs

640x360, grayscale, OpenCV

Figure 5-2 the result of the median filter for FPGAs and OpenCV

Table 5-2 the performance of median filter

	Time(hardware with SPI)	Time(total-transfer)
FPGAs	0.180s	0.060s
OpenCV	0.060s	0.060s

5.3 Gaussian filter

As same as the median filter, Figure 5-3 shows comparison between the FPGAs and OpenCV implementation of the algorithm of the Gaussian filter.

Only the time used is different, as the linear filter has less time to apply than the non-linear filter. The FPGAs implementation of the Gaussian filter takes tiny times which could not be detected by the operation system on the Raspberry Pi. As same as the result of the median filer, the most part of the running time is used at the data transfer part. If we only compare the running time of algorithm, the hardware is at least not slower than the pure software's implementation.

640x360, grayscale, FPGA

640x360, grayscale, OpenCV

Figure 5-3 the result of Gaussian filter for FPGAs and OpenCV

Table 5-3 the performance of Gaussian filter

	Time(hardware with SPI)	Time(total-transfer)
FPGAs	0.120s	~0.000s
OpenCV	0.020s	0.020s

5.4 MLP neural networks

The design of the neural networks is much more difficult than filters of the image processing. Look at the details of the implementation, 49 states are used for the neural networks, one of the 32 bits ALU computational components and more of the components of the peripheral devices are also involved. With the much larger of the design, the resource on the FPGAs we used in this project cannot meet the requirement any more. From the report of the synthesis tool, the SLICEs and LUTs for the logic are nearly double and these resource in the FPGAs are usually used to store the configuration files.

```
Design Summary
   Number of registers:   5458 out of  7209 (76%)
      PFU registers:          5458 out of  6864 (80%)
      PIO registers:             0 out of   345 (0%)
   Number of SLICEs:      6271 out of  3432 (183%)
      SLICEs as Logic/ROM:    6247 out of  3432 (182%)
      SLICEs as RAM:            24 out of  2574 (1%)
      SLICEs as Carry:        1531 out of  3432 (45%)
   Number of LUT4s:        12531 out of  6864 (183%)
      Number of logic LUTs:      9421
      Number of distributed RAM:  24 (48 LUT4s)
      Number of ripple logic:    1531 (3062 LUT4s)
      Number of shift registers:   0
   Number of PIO sites used: 6 + 1(JTAGENB) out of 115 (6%)
   Number of block RAMs:  18 out of 26 (69%)
   Number of GSRs:  1 out of 1 (100%)
   EFB used :       Yes
   JTAG used :      No
   Readback used :  No
   Oscillator used :  Yes
   Startup used :   No
   POR :            On
   Bandgap :        On
```

Figure 5-4 report of the synthesis tools

Although, there is no real experiment for the neural networks, a simulation result is still could give us some of idea about the real result of neural networks. For the experiment, we use the neural networks as a classifier to recognize the pattern on a 4x4 binary grid.

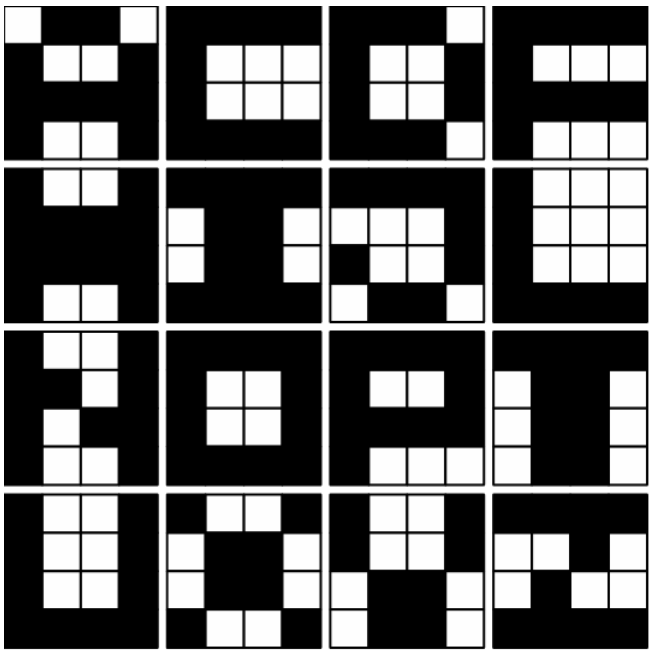


Figure 5-5 patterns for the detection by neural networks [19]

The inputs of the neural networks are some of the patterns among 16 letters of A, C, D, F, H, I, J, L, N, O, P, T, U, X, Y, Z by 4x4 binary girds. After the reading pattern and classification, the neural networks give the result back. For example, one of the result for the pattern is {0.97503 0.998393 0.968699 1.00423 -0.00983099 0.0638887 1.00992 -0.0138897 0.049473 0.975621 0.0662289 -0.020735 0.957592 0.986483 0.990517 0.970726} and we could say it is the letter Z.

From the result of the Table 5-4, the simulation result shows that the FPGAs is faster than the pure software if we only look at the running time of the algorithm part. It is as same as the filter real experiment results. A faster interface of the hardware may much helpful for the performance in this project.

100 samples	Transfer time	Time(hardware with SPI)	Time(total-transfer)
FPGAs (simulation)	0.006s	0.140s	0.008s
OpenCV	0.000s	0.010s	0.010s

Table 5-4 the performance between the hardware simulation result and OpenCV

6 Conclusion

The development of the hardware system in this project is quite tedious. As a newbie for the hardware design, I need to learn the knowledge from the very basic level. The hardware of the implementation of each algorithm really could improve the performance and be helpful in the real situation.

From the experiment results, the most time consuming part of the hardware system is the transferring part. The computation of the FPGAs is faster than the software if we only look at the computation time between the OpenCV and the VHDL. If we could use another FPGAs which has a much faster interface, the result could be much better than this project. A large number of the data like image pixels, weights and bias of the neural networks would need to load into the FPGA's FIFO buffers or SRAM storage in order to the computation in the latter time.

Because of the large requirement of the neural networks, the VHDL codes cannot convert into the bit stream file and work on the FPGAs chip. However, the design at here give us an idea about the implementation to the later use. If we have a more powerful FPGAs and even with some of the off-chip RAM, this implementation may could be done in reality.

After this project, I think I have obtained a great deal of knowledge in the hardware design. FPGAs is a great target for the image processing problems. This project help me understand the image processing and machine learning algorithms more deeply.

Reference

1. Jung M, Hohenstein F, Günthner W. “Staplerauge”: A Framework for Camera-Based Sensor Functions on Forklift Trucks[M]//Efficiency and Innovation in Logistics. Springer International Publishing, 2014: 77-88.
2. Upton E, Halfacree G. Raspberry Pi user guide[M]. John Wiley & Sons, 2013.
3. Pi R. FAQs[J]. Raspberry Pi, 2012.
4. Powers S. The open-source classroom: your first bite of raspberry pi[J]. Linux Journal, 2012, 2012(224): 7.
5. Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. " O'Reilly Media, Inc.", 2008.
6. pif - Raspberry Pi FPGA Board, version 1.0, October 03, 2013, manual.
7. Shahdad M, Lipsett R, Marschner E, et al. VHSIC hardware description language[J]. Computer, 1985, 18(2): 94-103.
8. Gonzalez R C, Woods R E, Eddins S L. Digital image processing using MATLAB[J]. Upper Saddle River, NJ Jensen: Prentice Hall,, 2004.
9. Bishop C M. Pattern recognition and machine learning[M]. New York: springer, 2006.
10. Counters B D L, Counters L D, Counters L U, et al. Primitive Library-MachXO2 and Platform Manager 2[J]. FPGA Libraries Reference Guide, 2013: 91.
11. Richard E. Haskell Darrin M. Hanna , (2009). *Introduction to Digital Design Using Digilent FPGA Boards — Block Diagram / VHDL Examples*. 1st ed. Oakland University, Rochester, Michigan : e.g. Houghton Mifflin.
12. TN1199 - MachXO2 sysCLOCK PLL Design and Usage Guide.
13. TN1201 - Memory Usage Guide for MachXO2 Devices.
14. TN1202 - MachXO2 sysIO Usage Guide.
15. TN1204 - MachXO2 Programming and Configuration Usage Guide.
16. TN1205 - Using User Flash Memory and Hardened Control Functions in MachXO2 Devices.
17. Nelson A E. Implementation of image processing algorithms on FPGA hardware[D]. Vanderbilt University, 2000.
18. Froß, A., Markert, E., Lange, R., & Heinkel, U. Entwicklung einer generischen FPGA-Implementierung Neuronaler Netze.
19. Zhou, J. L. (2010). Artificial Neural Network DSD Project Final Report.
20. Detrey, J., & De Dinechin, F. (2003). A VHDL library of parametrisable floating-point and LNS operators for FPGA. http//www. ens-lyon. fr/~ jdetrey/FPLibrary.

List of Figures

List of Tables

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbständig angefertigt habe.

Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Diese Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt.

München, 26. September 2014

Yisong, Qiao