Cat Recognizer Using Neural Network

Digital Design and Logical Synthesis for Electric Computer Engineering

(36113611)

Course Project

Digital High Level Design

Version 1.0

Maor Assayag 318550746

Refahel Shetrit 204654891

Revision Log

Rev	Change	Description	Reason for change	Done By	Date
0.1	Initial document				
0.2					
0.3					

Table of Content

LIST OF FIGURES

1. BLOCKS FUNCTIONAL DESCRIPTIONS

1.1.1 Functional Description

In this project we will implement such a system which recognizes a cat in an image (Figure 1). Each image has a fixed size of 64x64 pixels with three colors Red, Green, Blue (each color image is comprised of three images representing three color intensities) giving a total of 64x64x3=12288 pixels. Image is transformed into a vector in the following manner: first all the red pixels in the image, followed by all the green pixels and ending with all the blue pixels.

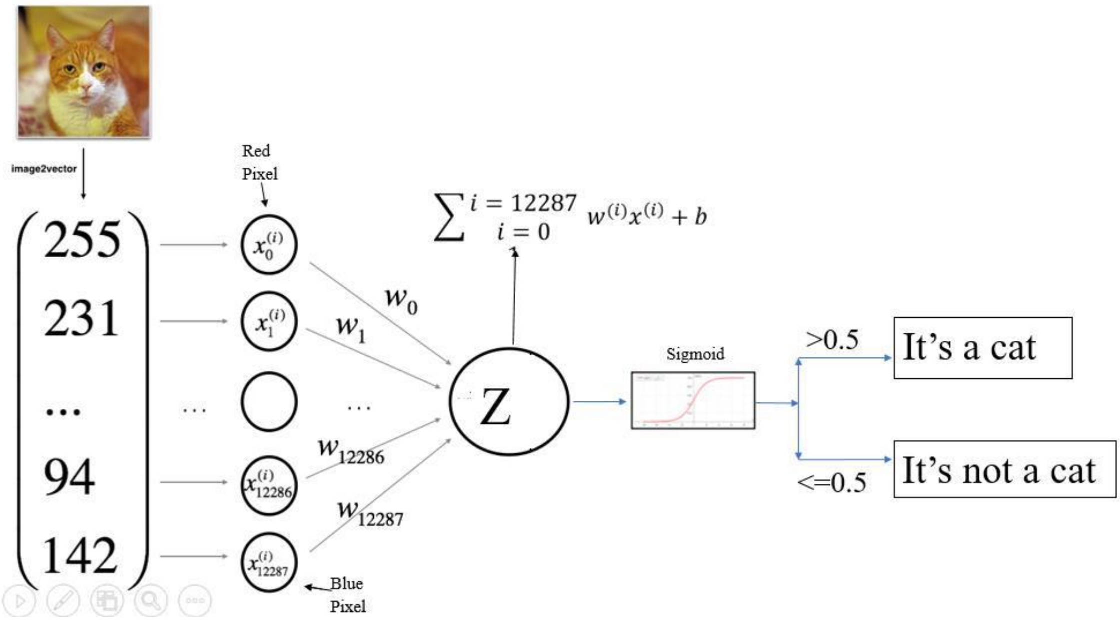

Figure 1: Cat Recognizer Using Neural Nets

Our design is a peripheral part of an ARM Processor and uses APB bus protocols to communicate with it (Figure 2). The CPU can read and write a register bank inside the design (marked in red).

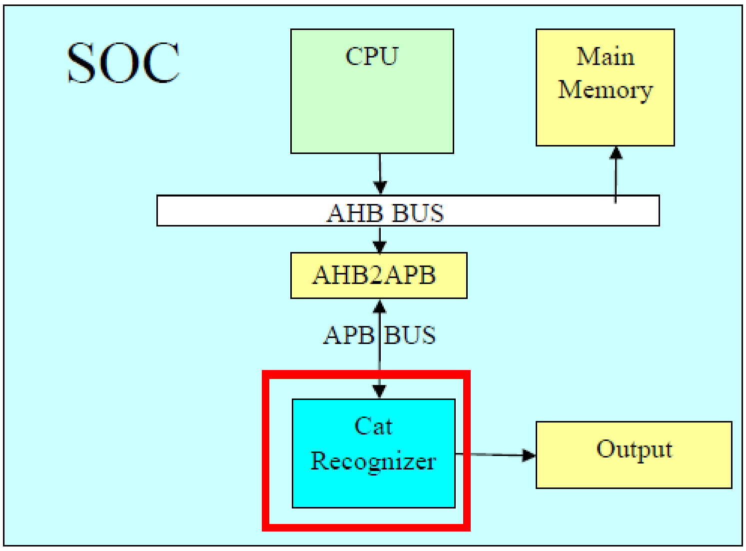

Figure 2: Top view of the SOC environment around the design

The design is a slave to the CPU master, which controls it via APB bridge and register files. The CPU sends the image data to the cat recognizer and it gives an output of "1" for "There is a cat" and "0" "there isn't a cat".

1.1.2 CatRecgonizer.v

The design, named CatRecognizer, is controlled by the CPU which configures the design via APB bus and a register bank inside the design. CPU can write to the register bank and read its content.

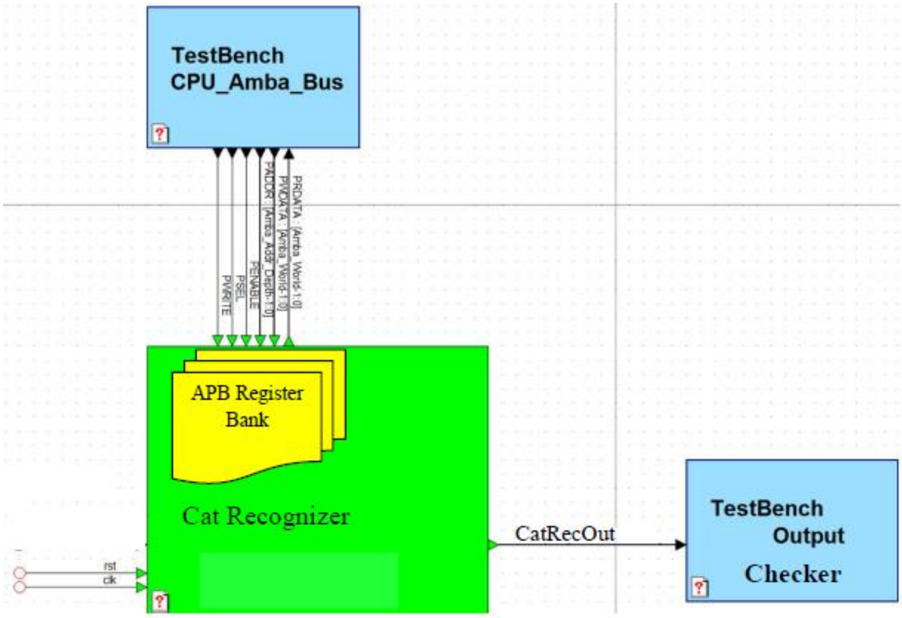

Figure 3 : Top view of the environment around the design of CatRecognizer

1.1.2.1 Interface

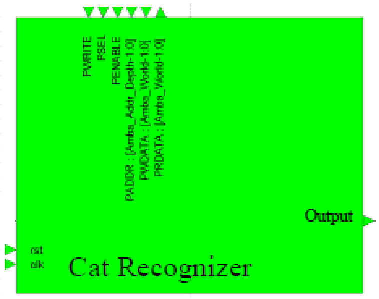

Figure 4: Top view of the block of CatRecognizer

Name	Mode	Type	Bound	Comment
clk	input	Wire	[Amba_Addr_Depth-1:0]	// System clock
rst	Input	Wire		// Reset active low
PSEL	Input	Wire		// APB Bus Select
PENABLE	Input	Wire		// APB Bus Enable/clk
PADDR	Input	Wire		// APB Address Bus
PWDATA	Input	Wire	[Amba_Word-1:0]	// APB Write Data Bus
PWRITE	Input	Wire		// APB Bus Write
PRDATA	Output	Wire	[Amba_Word-1:0]	// APB Read Data Bus
CatRecOut	Output	Wire		// CatRecognizer result

Table 1: Block interface of CatRecognizer

Parameter Name	Range	Defualt values	Comment
Amba_Word	24,32	24	Part of the Amba standard at moodle site
Amba_Addr_Depth	12,13,14	12	Part of the Amba standard at moodle site
WeightPrecision	5,8,16	5	Bit depth of the weights and bias

Table 2: Parameters of CatRecognizer

1.1.2.2 Structure

The model will read each clock if address 0x00 in APB register file is 0x01. If so, the module will start to fetch each 1 clock 1 register data from the pixel bank (APB bank) and 1 register data from the weights bank. The row of data will be sliced to 3 equal section and will be transferred directly to 3 neurons for signed multipion (Figure 5). The results of the previous neurons will be accumulated into currentResult variable. The process of fetch new data and accumulated will finished when the process will fetch the last row of data from the banks (4096 rows). Then, correspond bias (decided by weightPrecsion parameter) will be added to currentResult and the module will output the answer by the result of sigmoid function on currentResult signed number (which transform to the test – is the currentResult positive or negative ?).

The process flow also described as image in *Figure 1*.

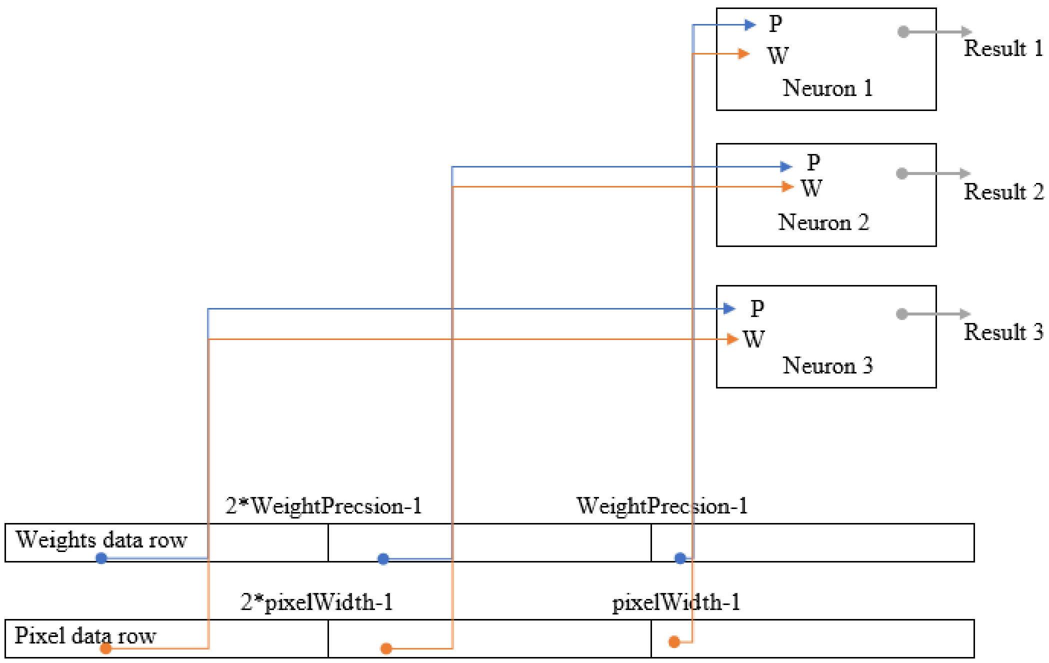

Figure 5: Top view of Neurons getting information.

Parameter Name	Value	Comment
pixelNumber	12288	How many pixel we expect
PixelWidth	8	Width of Data row in the data file
resultWidth	14 + Amba_Word + WeightPrecision	Bit depth of the weights and bias
iteration	4096	Local var to keep track of the process
WeightRowWidth	3 * WeightPrecision	Width of Data row in the weight bank

Table 3: Local Parameters of CatRecognizer

Component Name	Component Label	Comment
APB	APB_Bank	Register file for the pixels data.
WeighstBank	Weights_Bank	Register file for the weights. include_file_5/8/16.v is an assign file genreated from python code.
Neuron	Neuron_1	Basic multiplier of 8bit pixel data and 5/8/16 bit corresponding weight.
Neuron	Neuron_2	Basic multiplier of 8bit pixel data and 5/8/16 bit corresponding weight.
Neuron	Neuron_3	Basic multiplier of 8bit pixel data and 5/8/16 bit corresponding weight.

Table 4: Components Table of CatRecognizer

Variable Name	Type	Bound	Comment
reset	Reg		// Reset active low
doneFlag	Reg		// CatRecognizer produced a result
doneAddingBias	Reg		// The process added bias to the cumulative sum
doneIteration	Reg		// The process done computing the sigma
APB_control	Reg	[1:0]	// control for the APB bank
WeightsData_temp	Reg	[(WeightRowWidth-1):0]	// current row of data of weights
PixelData_temp	Reg	[(Amba_Word-1):0]	// current row of data of pixels
currentAddress	Reg	[(Amba_Addr_Depth -1):0]	// current address to fetch data from
currentResult	Reg signed	[resultWidth - 1: 0]	// current cumulative sum
bias	Reg signed	[15:0] of [2:0]	// Bias bank

Table 5: Important local Variables Table of CatRecognizer

1.1.3 Neuron.v

The design, named CatRecognizer, is a basic signed multiplier of 8bit pixel data and 5/8/16 bit corresponding weight.

1.1.3.1 Interface

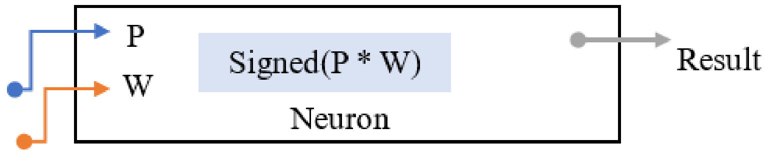

Figure 6: Top view of the block Neuron.

Name	Mode	Type	Bound	Comment
p	input	Signed Wire	[PixelWidth : 0]	// pixel data, padding MSB 0
w	Input	Signed Wire	[WeightWidth : 0]	// weight data
result	Output	Signed Wire	[2*(WeightWidth + PixelWidth):0]	// results of p*w

Table 6: Block interface of Neuron.

Parameter Name	Range	Defualt values	Comment
PixelWidth	8	8	standard pixel width
WeightWidth	5,8,16	5	System parameters

Table 7: Parameters of Neuron

1.1.4 APB.v

The controller has a set of several registers at size amba_word bit registers. The registers are accessible via the APB interface. We implemented this interface according to the AMBA standard (APB section).

Each row contains at least 24 bits, which equal to 3 lines of data from the text file image data.

Register Name	Address Offset	Comments	Access Type
Control (CTRL)	0x00	Controls the design	CPU Read/Write CatRecognizer Read only
PixelData1	0x01	Pixel Data	CPU Read/Write CatRecognizer Read only
PixelData2	0x02	Pixel Data	CPU Read/Write CatRecognizer Read only
PixelData3	0x03	Pixel Data	CPU Read/Write CatRecognizer Read only
. .	. .	Size of the register bank is set by Amba_Addr_Depth	. .

Table 8: Register file structure of APB bank register

1.1.4.1 Interface

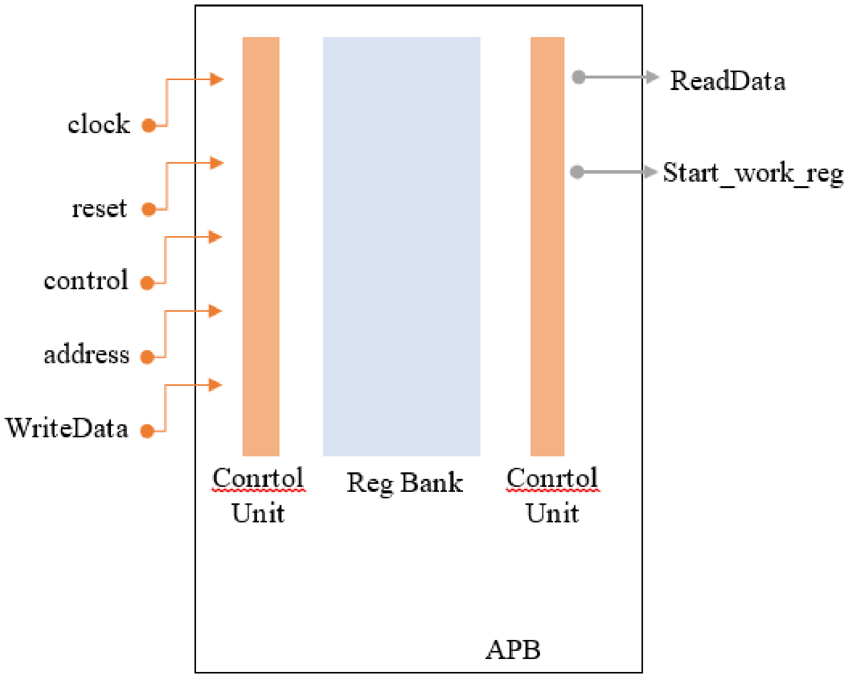

Figure 7: Top view of the block of APB bank register

Name	Mode	Type	Bound	Comment
clock	input	Wire		// System clock
reset	Input	Wire		// Reset active low
control	Input	Wire	[1:0]	// APB Bus Select
address	Input	Wire	[Amba_Addr_Depth-1:0]	// APB Address
WriteData	Input	Wire	[Amba_Word-1:0]	// APB Write - Data bus
ReadData	Output	Wire	[Amba_Word-1:0]	// APB Read - Data Bus
Start_work_reg	Output	Wire		// control the design

Table 9: Block interface of APB bank register

Parameter Name	Range	Defualt values	Comment
Amba_Word	24,32	24	Part of the Amba standard at moodle site
Amba_Addr_Depth	12,13,14	12	Part of the Amba standard at moodle site

Table 10: Parameters of APB bank register

1.1.4.2 Structure

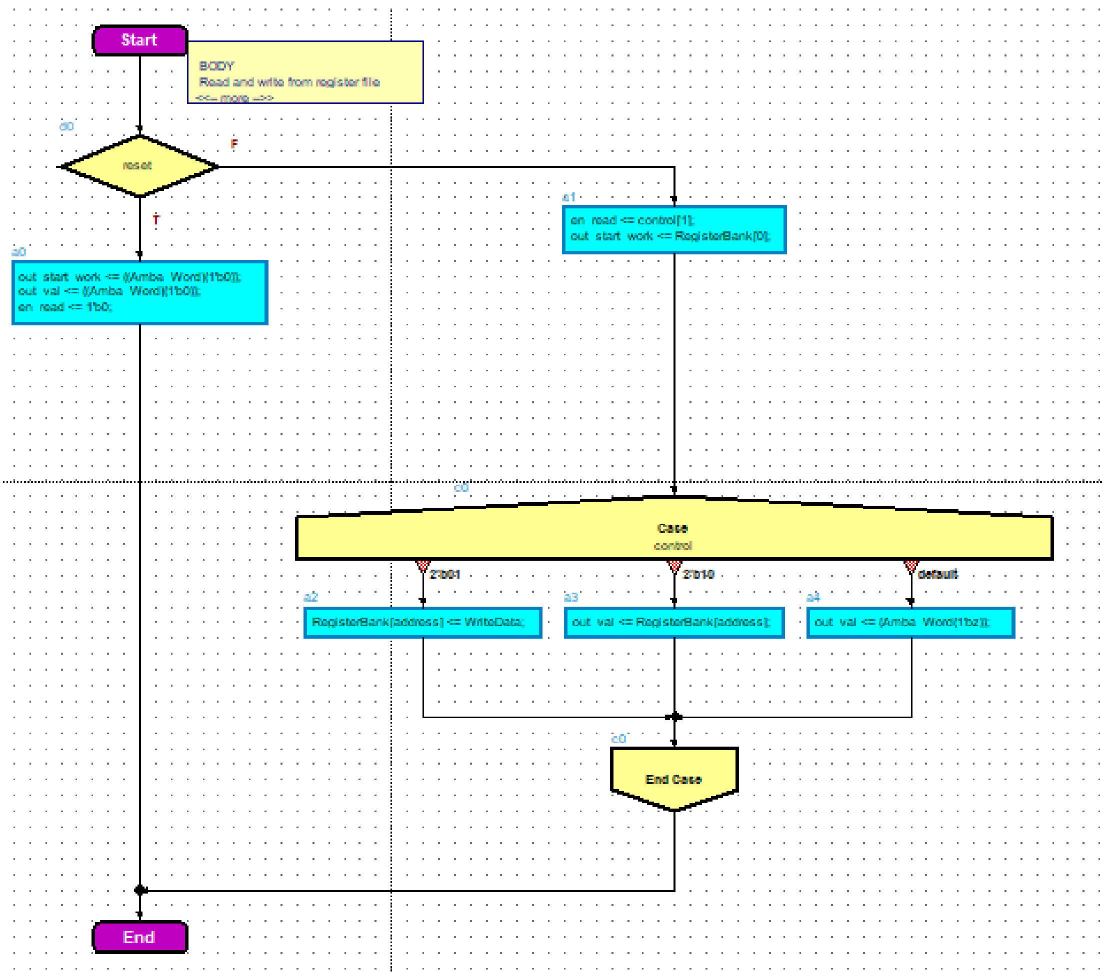

Figure 8: *Top view of of APB bank register*

Variable Name	Type	Bound	Comment
RegisterBank	Reg	[(2**Amba_Addr_Depth)-1:0] of [(Amba_Word -1):0]	// Register bank
out_val	Reg	[(Amba_Word -1):0]	// Read data output
out_start_work	Reg	[Amba_Word -1:0]	// Read data of address 0x00
en_read	Reg		// Enable output out_val

Table 11: Important local Variables Table of APB bank register

1.1.5 WeightsBank.v

The controller has a set of several registers at size amba_word bit registers. The registers are accessible only to CatRecognizer We implemented this interface according to the AMBA standard (APB section). The design required an initial reset for initialize the values to the correspond WeightPrecsion include file. The include files is generated by python code which include in the appendix.

Each row contains at least 15 bits, which equal to 3 lines of data from the text file weights data.

Register Name	Address Offset	Comments	Access Type
WeightsData1	0x00	Weight Data	CatRecognizer Read / Write
WeightsData2	0x01	Weight Data	CatRecognizer Read / Write
WeightsData3	0x02	Weight Data	CatRecognizer Read / Write
. .	. .	Size of the register bank is set by Amba_Addr_Depth	. .

Table 12: Register file structure of Weights bank register

1.1.5.1 Interface

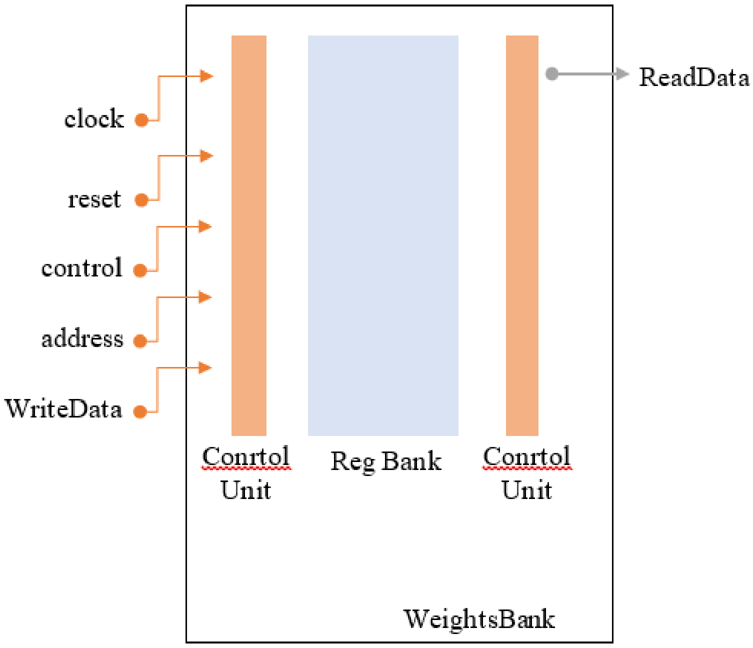

Figure 9: Top view of the block of Weights bank register

Name	Mode	Type	Bound	Comment
clock	input	Wire		// System clock
reset	Input	Wire		// Reset active low
control	Input	Wire	[1:0]	// APB Bus Select
address	Input	Wire	[Amba_Addr_Depth-1:0]	// APB Address
WriteData	Input	Wire	[Amba_Word-1:0]	// APB Write - Data bus
ReadData	Output	Wire	[Amba_Word-1:0]	// APB Read - Data Bus

Table 13: Block interface of Weights bank register

Parameter Name	Range	Defualt values	Comment
WeightPrecision	5,8,16	5	Weight width
Amba_Addr_Depth	12,13,14	12	Part of the Amba standard at moodle site
WeightRowWidth	15,24,48	15	Register width = WeightPrecision*3

Table 14: Parameters of Weights bank register

Local Parameter Name	Range	Defualt values	Comment
WRITE	2'b01	2'b01	State Machine
READ	2'b10	2'b10	State Machine

Table 15: Parameters of Weights bank register

1.1.5.2 Structure

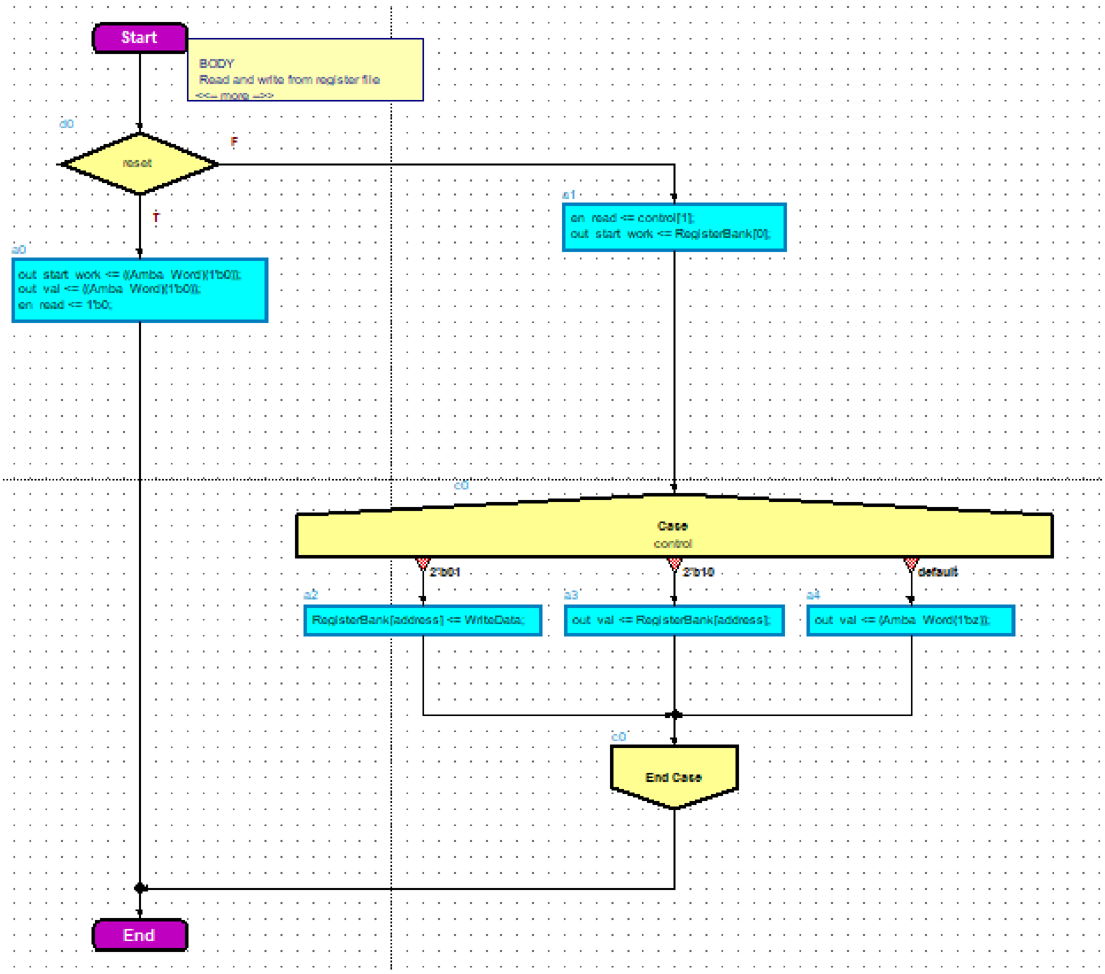

Figure 10: Top view of Weights bank register

Variable Name	Type	Bound	Comment
RegisterBank	Reg	[(2**Amba_Addr_Depth)-1:0] of [(WeightRowWidth -1):0]	// Register bank
out_val	Reg	[(WeightRowWidth -1):0]	// Read data output
en_read	Reg		// Enable output out_val

Table 16: Important local Variables Table of Weights bank register

2. TEST BENCH

2.1 AllImages_tb.v

For testing the module CatRecognizer we built a test bench that will simulate a CPU.

First, the test bench will fetch 40 correct answer for 40 images. We can decide which weights precision we want to test (5/8/16) and change only ONE integer in the simulation – WeightPrecision.

Secondly, we made sure to reset the design correctly.

Then, for each image we will read 3 lines from the correspond text file (for e.g. image4.txt) and write to the APB interface the data and address of the concatenation data that we extracted. The writing process according to AMBA protocol involved writing SEL and only in the next cycle writing ENABLE and enable writing to the APB bank registers.

After each image is writing we compare the output answer of CatRecgonizer to the right answer and write to the MODELSIM terminal the results (success / fail).

After all 40 images has been process and calculate we display the final result of how much the model is accurate.

The simulation required 4800us time.

```
# TestBench for image 1 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 2 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 3 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 4 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 5 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 6 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 7 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 8 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 9 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 10 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 11 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 12 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 13 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 14 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 15 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 16 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 17 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 18 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 19 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 20 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 21 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 22 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 23 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 24 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 25 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 26 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 27 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 28 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 29 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 30 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 31 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 32 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 33 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 34 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 35 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 36 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 37 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 38 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 39 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench done successfuly, CatRecgonizer is accurate
```

Figure 11: Simulation results of WeightPrecsion = 16

```
# TestBench for image 2 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 3 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 4 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 5 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 6 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 7 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 8 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 9 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 10 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 11 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 12 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 13 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 14 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 15 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 16 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 17 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 18 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 19 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 20 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 21 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 22 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 23 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 24 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 25 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 26 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 27 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 28 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 29 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 30 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 31 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 32 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 33 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 34 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 35 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 36 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 37 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 38 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 39 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench done successfuly, CatRecgonizer is accurate
```

Figure 12: Simulation results of WeightPrecsion = 8

```
# TestBench for image 1 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 2 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 3 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 4 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 5 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 6 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 7 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 8 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 9 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 10 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 11 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 12 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 13 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 14 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 15 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 16 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 17 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 18 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 19 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 20 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 21 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 22 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 23 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 24 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 25 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 26 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 27 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 28 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 29 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 30 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 31 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 32 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 33 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 34 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 35 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 36 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 37 : true, Correct Answer = 1, CatRecgonizer = 1
# TestBench for image 38 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench for image 39 : true, Correct Answer = 0, CatRecgonizer = 0
# TestBench done successfuly, CatRecgonizer is accurate
```

Figure 13: Simulation results of WeightPrecsion = 5

3. APPENDIX

3.1 Terminology

LSB - Least Significant Bit

MSB - Most Significant Bit

TBR - To Be Reviewed

TBD - To Be Defined

3.2 References

3.2.1 "Verilog HDL Operators". University of Texas at dallas - Nanometer design laboratory –[online]. https://www.utdallas.edu/~akshay.sridharan/index_files/Page5212.htm

3.2.2 AMBA Specification (Rev 2.0). May 1999. [online] https://www.arm.com.

3.2.3 "Reading and Writing Files in Python". July 2013. [online] /www.pythonforbeginners.com.

3.3 Python code for creating include weights files – generateVerilog,py

```
import sys
file_to_read = "WeightsPrecision16.txt"
file_to_write = 'include_file_16.v'

orig_stdout = sys.stdout
f = open(file_to_write, 'w')
sys.stdout = f

def to_twoscomplement(bits, value):
    if value < 0:
        value = ( 1<<bits ) + value;
    formatstring = '{:0%ib}' % bits;
    return formatstring.format(value)

def LoadTextFile():
        with open(file_to_read, "r") as ins:
            array = []
            for line in ins:
                array.append(int(line))
        return array

def generateVerilog(DataWidth, array):
    s = "";
    j=0;
    for i in range(0, len(array)):
        s = s + to_twoscomplement(int(DataWidth/3), array[i]);
        if ((i > 0) and ((i+1)%3==0)):
            print ("\t\tRegisterBank[%d] <= %d'b%s;" % (j,DataWidth,s))
```

```
        j=j+1;
        s = "";

generateVerilog(3 * 16, LoadTextFile())

sys.stdout = orig_stdout
f.close()
```