VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

LOGIC DESIGN PROJECT (CO3091)

Semester 231:

"Implement Activation Function (AF) calculation blocks on FPGAs"

Advisors: Tran Ngoc Thinh, CSE-HCMUT
Huynh Phuc Nghi, CSE-HCMUT

Students: Bien Cong Thanh - 2053424
Dinh Xuan Quang - 2053359

HO CHI MINH CITY, January 2024

Contents

1 Member list & Workload

No.	Full name	Student ID	Problems	Percentage of work
1	Bien Cong Thanh	2053424	Sigmoid-Implement-Compare	100%
2	Dinh Xuan Quang	2053359	ReLu-Implement-Compare	100%

2 Introduction

Activation functions play a crucial role in Convolutional Neural Networks (CNNs) by introducing non-linearity into the network. This non-linearity is essential because it allows the CNN to learn complex patterns and perform tasks beyond mere linear classification1. Without activation functions, a CNN would simply perform a linear transformation of the input data, which is insufficient for handling the complexities of real-world data.

We choose two activation functions Sigmoid and Rectified for our project because they are the most basic and popular functions nowadays.Then we will compare these activate functions base on there performance in Arty-z7-20 board by using Vivaldo.

In summary, we want to implement two activate functions Sigmoid and Rectified and compare them for application in CNN.

3 Sigmoid Activation function

3.1 Definition

The Sigmoid activation function is a type of activation function used in artificial neural networks. It transforms any real-valued number into a value between 0 and 1. The sigmoid function is defined by the formula:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

3.2 Properties

- Parametric:The Sigmoid function has parameters (weights and biases) that are learned during training.
- Monotonic: The Sigmoid function is also monotonically increasing or decreasing depending on the positive input.
- Smooth: The sigmoid function is smooth and continuously differentiable. This property makes it suitable for gradient-based optimization algorithms.
- Bounded: The Sigmoid function outputs in the range $[0, 1]$.
- Output range: The output of the sigmoid function is always in the range $[0, 1]$. It can approximate binary decision boundaries and is often used for problems where the goal is to estimate probabilities.

3.3 Application

- Image Based applications:
  - Commonly used in the output layer of binary image classification tasks
  - Suitable for tasks where the goal is to predict probabilities, such as identifying whether an image contains a specific object.
- Time Series Applications:
  - Used in tasks like predicting the probability of an event occurring at a specific time
  - Suitable for binary time series classification problems.
- Text-Based Applications:
  - Applied in sentiment analysis where the goal is to predict positive or negative sentiment probabilities.
  - Useful in binary text classification tasks.
- Signal-Based Applications:
  - Used in tasks where the goal is to detect specific patterns or events in a signal with a binary outcome.
  - Applied in biomedical signal processing for binary classification tasks.
- Games-Related Applications:
  - Applied in games for binary decisions, such as determining whether a character should take a specific action or not.
  - Useful in reinforcement learning scenarios for binary choices.

4 Rectified Activation function (ReLU)

4.1 Definition

Rectified Linear Unit (ReLU) is a simple function which is the identity function for positive input and zero for negative input and given as:

$$ReLU(x) = max(0, x) = \begin{cases} x, if x \geq 0 \\ 0, otherwise \end{cases} \tag{2}$$

4.2 Properties

- No-parametric: The ReLU function does not have parameters that are learned; it simply threshold the input values.
- Monotonic: The ReLU function is also monotonically increasing or decreasing depending on the positive input.
- Non-smooth: Its derivative is discontinuous at 0, which can pose challenges for gradient-based optimization algorithms (gradients become very small as they propagate through multiple layers).

- Unbounded: The ReLU function outputs values in the range $[0, +\infty)$, making it unbounded for positive inputs.
- Outputs in the range $[0, +\infty)$. It retains positive inputs and sets negative inputs to zero. Not suitable for tasks requiring a bounded output.

4.3 Application

- Image Based applications:
  - Feature extraction: ReLU's sparsity and computational efficiency make it ideal for convolutional neural networks (CNNs) used for image feature extraction and classification.
  - Image recognition and generation: ReLU's ability to handle large gradients efficiently makes it beneficial for deep CNNs used in tasks like image recognition and image generation.
- Time Series Applications:
  - Time series forecasting: ReLU's computational efficiency is beneficial for long-term forecasting tasks where large datasets are involved.
  - Regression on time series data: ReLU can be used in RNNs to predict future values in time series data, such as stock prices or sensor readings.
- Text-Based Applications:
  - Sentiment analysis: Sigmoid can be used to predict the sentiment of a text as positive, negative, or neutral based on its probability score.
  - Text classification: Sigmoid can be used in RNNs to classify text data into different categories (e.g., topic modeling, spam detection).
- Signal-based applications:
  - Signal processing: ReLU can be used in tasks like audio denoising, signal compression, and feature extraction from audio data.
  - Suitable for extracting features from signals using deep learning architectures.
- Games-related applications:
  - Learning to play: Effective for learning complex representations from game state data
  - Action selection: ReLU can be used in control systems of game AI to select actions based on their predicted value or expected reward.

5 Implementation

Constructing digital circuits that generate activation functions poses significant challenges and demands substantial computational resources.

Hence, we generally pre-calculate their values (since we will be aware of range of x, the input) and store in a ROM.

These ROMs will be also called Look Up Tables (LUTs), these LUT ROMs are built either using block RAMs or distributed RAMs (FFs and LUTs).

We use python to create Memory Initialization files.

```
import math

def genSigContent(dataWidth,sigmoidSize,weightIntSize,inputIntSize):
    f = open("sigContent.mif","w")
    fractBits = sigmoidSize-(weightIntSize+inputIntSize)
    if fractBits < 0: #Sigmoid size is smaller the integer part of the MAC
    operation
        fractBits = 0
    x = -2**(weightIntSize+inputIntSize-1)#Smallest input going to the Sigmoid LUT
     from the neuron
    for i in range(0,2**sigmoidSize):
        y = sigmoid(x)
        z = DtoB(y,dataWidth,dataWidth-inputIntSize)
        f.write(z+'\n')
        x=x+(2**-fractBits)
    f.close()

def DtoB(num,dataWidth,fracBits):#funtion for converting into two's complement
    format
    if num >= 0:
        num = num * (2**fracBits)
        num = int(num)
        e = bin(num)[2:]
    else:
        num = -num
        num = num * (2**fracBits)#number of fractional bits
        num = int(num)
        if num == 0:
            d = 0
        else:
            d = 2**dataWidth - num
        e = bin(d)[2:]
    return e


def sigmoid(x):
    try:
        return 1 / (1+math.exp(-x))#for x less than -1023 will give value error
    except:
        return 0


if __name__ == "__main__":
    genSigContent(dataWidth=16,sigmoidSize=10,weightIntSize=4,inputIntSize=1)
```

5.1 Folder Structure

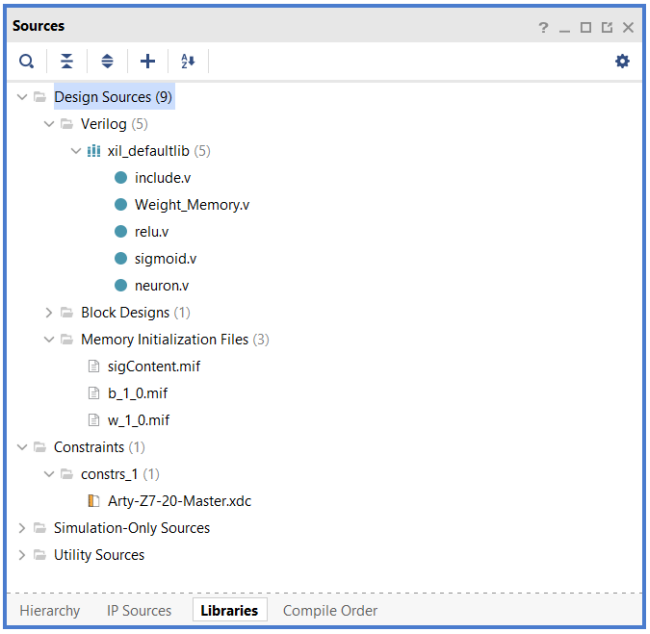


Figure 1: *Folder structure*

5.2 Code Verilog for Sigmoid function

```
`timescale 1ns / 1ps

module Sig_ROM #(parameter inWidth=10, dataWidth=16) (
    input           clk,
    input   [inWidth-1:0]   x,
    output  [dataWidth-1:0]  out
    );

    reg [dataWidth-1:0] mem [2**inWidth-1:0];
    reg [inWidth-1:0] y;

  initial
  begin
    $readmemb("sigContent.mif",mem);
  end

    always @(posedge clk)
    begin
        if($signed(x) >= 0)
            y <= x+(2**(inWidth-1));
        else
            y <= x-(2**(inWidth-1));
    end

    assign out = mem[y];

endmodule
```

5.3 Code Verilog for ReLU function

```
module ReLU  #(parameter dataWidth=16,weightIntWidth=4) (
    input           clk,
    input   [2*dataWidth-1:0]   x,
    output  reg [dataWidth-1:0]  out
);

always @(posedge clk)
begin
    if($signed(x) >= 0)
    begin
        if(|x[2*dataWidth-1-:weightIntWidth+1]) //over flow to sign bit of integer
     part
            out <= {1'b0,{(dataWidth-1){1'b1}}}; //positive saturate
        else
            out <= x[2*dataWidth-1-weightIntWidth-:dataWidth];
    end
    else
        out <= 0;
end

endmodule
```

5.4 Code Verilog of neuron.v

We decide a neuron to run two activation functions by **neuron.v**

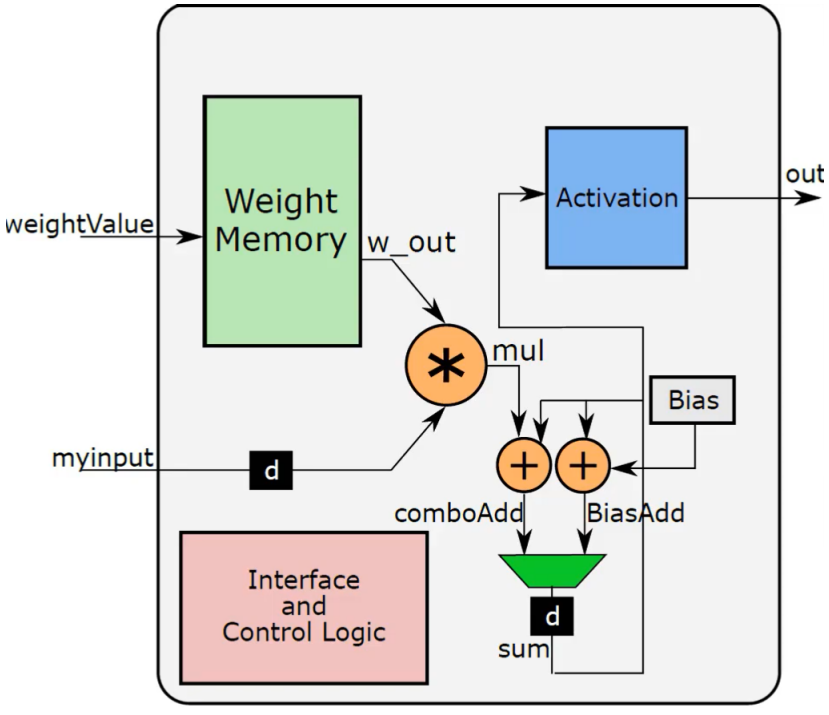


Figure 2: *Neuron Architecture*

Link source code https://github.com/thanhbien0402/LogicDesign

To run implement Sigmoid function, we type **sigmoid** in the file **neuron.v**:

```
module neuron #(parameter layerNo=0,neuronNo=0,numWeight=784,dataWidth=16,
    sigmoidSize=10,weightIntWidth=4,actType="sigmoid",biasFile="b_1_0.mif",
    weightFile="w_1_0.mif")
```

And to run ReLU function, we type **relu** in the file **neuron.v**:

```
module neuron #(parameter layerNo=0,neuronNo=0,numWeight=784,dataWidth=16,
    sigmoidSize=10,weightIntWidth=4,actType="relu",biasFile="b_1_0.mif",weightFile
    ="w_1_0.mif")
```

6 Compare with two Activation functions

We compare Sigmoid and ReLU implementations in terms of resource utilization, timing performance and power consumption.

Resource Utilization:

- LUTs (Look-Up Tables): Compare the number of LUTs used in each implementation. Lower LUT usage generally indicates a more efficient design, potentially leaving more resources available for other functions.
- FFs (Flip-Flops): Compare the number of FFs used. Lower FF usage can lead to smaller designs and potentially lower power consumption.
- BRAMs (Block RAMs): Compare the number of BRAMs used. If memory usage is a concern, prioritize implementations with lower BRAM utilization.
- DSPs (Digital Signal Processors): Compare the number of DSPs used. If your design relies heavily on signal processing operations, consider implementations that efficiently utilize DSP resources.

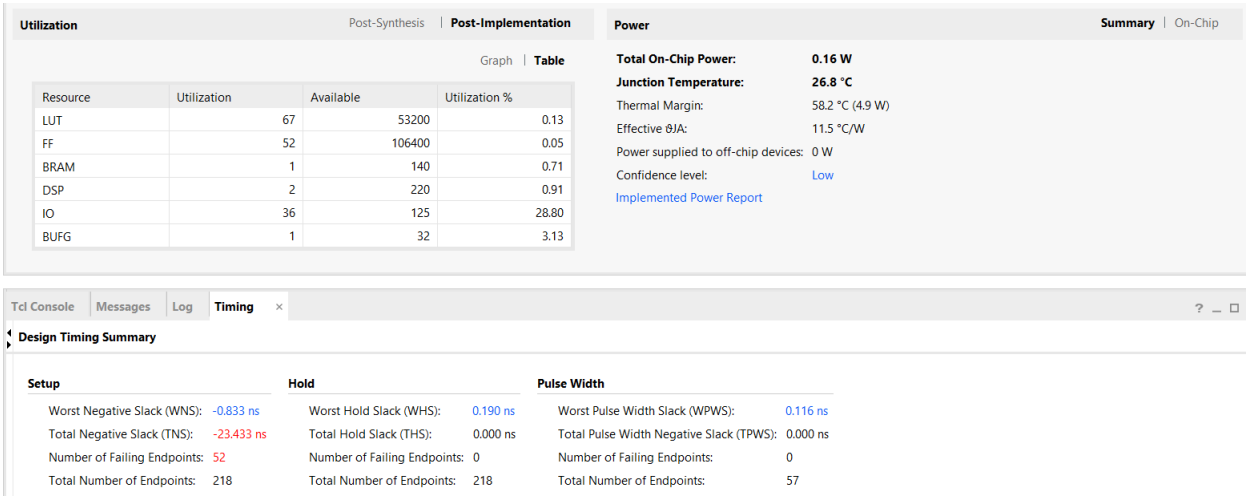
Timing Analysis:

- WNS (Worst Negative Slack): Compare the WNS values. Lower (more negative) WNS indicates a higher risk of timing violations, potentially leading to functional errors. Prioritize implementations with higher (less negative) WNS values.
- TNS (Total Negative Slack): Compare the TNS values. Lower TNS generally indicates better overall timing performance across multiple paths in the design.

Power:

- Total on-chip power: Compare the estimated power consumption of each implementation. Choose the implementation with the lowest power consumption if power constraints are critical.
- Junction temperature: Compare the estimated junction temperatures. Ensure temperatures remain within safe operating limits to avoid thermal issues.

6.1 Sigmoid function report



Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	67	53200	0.13
FF	52	106400	0.05
BRAM	1	140	0.71
DSP	2	220	0.91
IO	36	125	28.80
BUFG	1	32	3.13

Power — **Summary** | On-Chip

Total On-Chip Power:	**0.16 W**
Junction Temperature:	**26.8 °C**
Thermal Margin:	58.2 °C (4.9 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

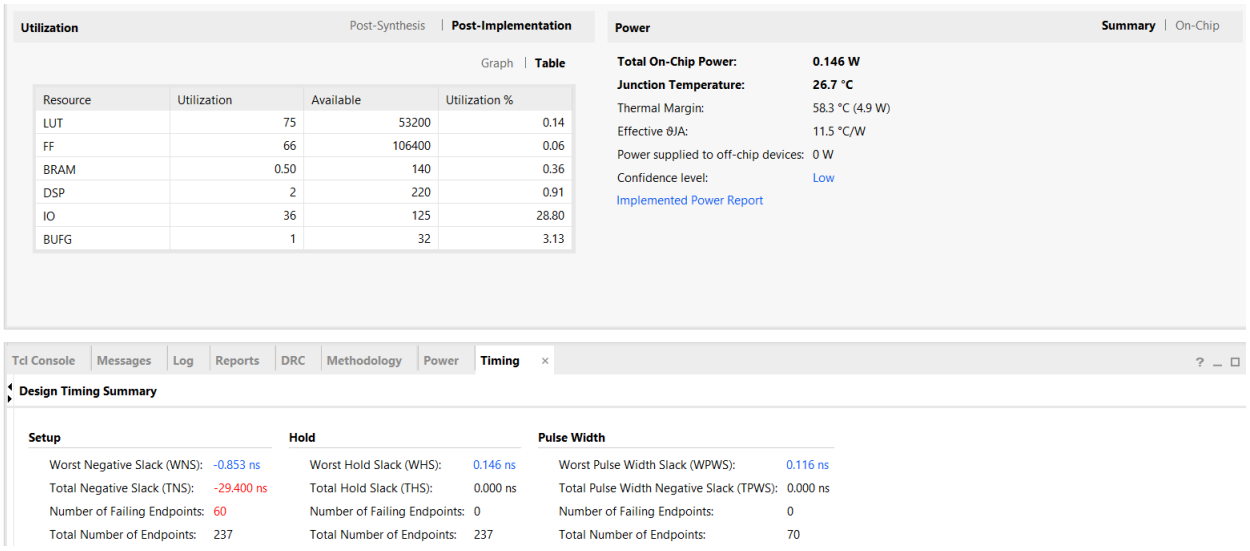
Tcl Console | Messages | Log | **Timing**

Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	-0.833 ns	Worst Hold Slack (WHS):	0.190 ns	Worst Pulse Width Slack (WPWS):	0.116 ns
Total Negative Slack (TNS):	-23.433 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	52	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	218	Total Number of Endpoints:	218	Total Number of Endpoints:	57

Figure 3: *Sigmoid function report*

6.2 ReLU function report



Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	75	53200	0.14
FF	66	106400	0.06
BRAM	0.50	140	0.36
DSP	2	220	0.91
IO	36	125	28.80
BUFG	1	32	3.13

Power — **Summary** | On-Chip

Total On-Chip Power:	**0.146 W**
Junction Temperature:	**26.7 °C**
Thermal Margin:	58.3 °C (4.9 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Tcl Console | Messages | Log | Reports | DRC | Methodology | Power | **Timing**

Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	-0.853 ns	Worst Hold Slack (WHS):	0.146 ns	Worst Pulse Width Slack (WPWS):	0.116 ns
Total Negative Slack (TNS):	-29.400 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	60	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	237	Total Number of Endpoints:	237	Total Number of Endpoints:	70

Figure 4: *ReLU function report*

6.3 Conclusion

Properties	Sigmoid	Relu
LUT(%)	0.13	0.14
FF(%)	0.05	0.06
BRAM(%)	0.71	0.36
DSP(%)	0.91	0.91
WNS(ns)	-0.833	-0.853
TNS(ns)	23.433	29.4
Total on chip power(W)	0.16	0.146
Junction temperature (C)	26.8	26.7

- Sigmoid:
  - More efficient design (smaller design),potentially leaving more resources available for other functions. (less LUTs AND FFs)
  - Less potential for leading to function error (higher WNS)
- ReLu:
  - Allowing for larger and more complex neural network models to be deployed on the FPGA, more scalable and cost-effective FPGA implementations (less BRAMs)
  - Better timing performance (higher TNS)
  - Saving more power comsumption (less power)

7 Re-evaluate the project

7.1 Achievement

- We implement Sigmoid and ReLU activate function by verilog language in Vivaldo.
- We have more knowledge about activate functions in deep learning.

7.2 Difficulties

- We have not yet implemented activation functions on the chip. We are still working on optimizing the design.

8 References

[1] Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri. “Activation functions in deep learning: A comprehensive survey and benchmark.” Neurocomputing (2022).

[2] Chigozie Nwankpa,W. L. Ijomah, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”, (12/2020)