徽标

描述已自动生成

IoT FINAL PROJECT

REPORT

GUO Xiaofan

YIN Chenghao

Contents

[1. Introduction - 3 -](#_Toc153487368)

[1.1 GAP8 Introduction - 3 -](#_Toc153487369)

[1.1.1 Autonomous Peripherals - 3 -](#_Toc153487370)

[1.1.2 Ultra-low Power Microcontroller - 3 -](#_Toc153487371)

[1.1.3 Compute Engine - 4 -](#_Toc153487372)

[1.1.4 Features of GAP8 - 4 -](#_Toc153487373)

[1.2 The Neural Network - 4 -](#_Toc153487374)

[1.2.1 Input Layer - 6 -](#_Toc153487375)

[1.2.2 Hidden Layer - 6 -](#_Toc153487376)

[1.2.3 Output Layer - 6 -](#_Toc153487377)

[2. Realization - 7 -](#_Toc153487378)

[2.1 Code link. - 7 -](#_Toc153487379)

[2.2 Analysis of implementation process and results. - 7 -](#_Toc153487380)

[2.2.1 Realize a simple neural network. - 7 -](#_Toc153487381)

[2.2.2 Implement the neural network on GAP8. - 8 -](#_Toc153487382)

[2.2.3 Use flex sensor data as training and test data. - 11 -](#_Toc153487383)

[2.2.4 Multi-core asynchronous control - 14 -](#_Toc153487384)

[3. References - 19 -](#_Toc153487385)

# Introduction

In this final IoT project, the objective is to gain proficiency in utilizing the multi-core capabilities of low-power development boards GAP8, to understand the basic structure of neural networks, and to learn how to analyze performance.

The project involves mastering control of the GAP8's multi-core functionality by running different functions on distinct cores, implementing a simple neural network, and deploying this network on the GAP8. The project will use simulated data for training and predictions and will conduct a detailed performance analysis of the GAP8, focusing on aspects such as cycles, frequency, and time, as outlined in the attached document.

## 1.1 GAP8 Introduction

The GAP8 microprocessor, crafted by GreenWaves for edge computing and IoT, features a nona-core 32-bit RISC-V architecture. This innovative processor is structured around three key components: autonomous peripherals, an ultra-low power microcontroller, and the compute engine[1].

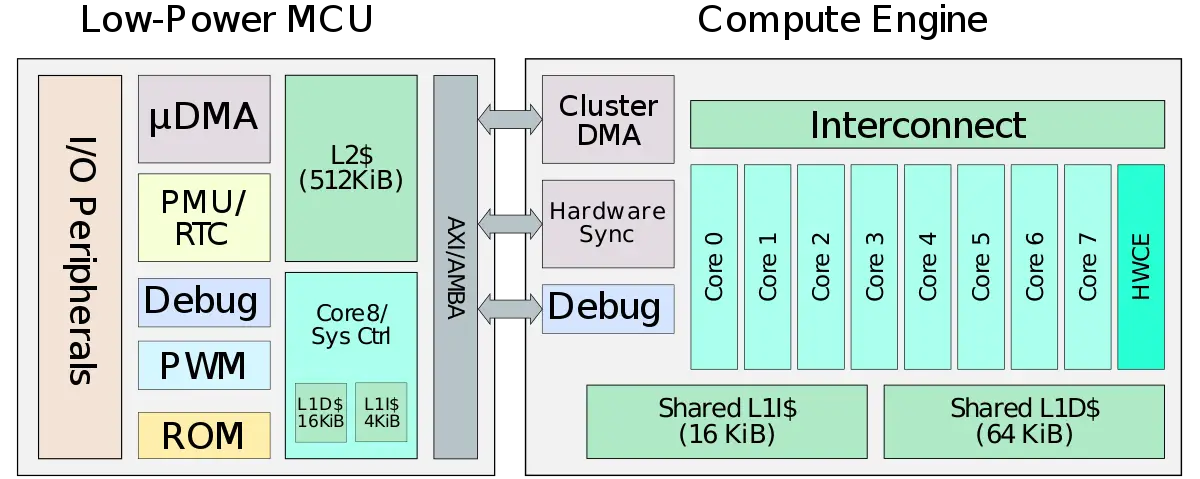


Figure 1 Gap8 Block Diagram[2]

### Autonomous Peripherals

A set of hardware components which designed to operate independently of the main CPU core, enhancing efficiency and reducing power consumption. These peripherals can manage specific tasks without continuous intervention from the central processor, allowing it to remain in a low-power state or focus on other computations.

### Ultra-low Power Microcontroller

The microcontroller block is a standard MCU (Microcontroller Unit) with many of the standard features. The MCU is situated on its own power domain with the peripherals power switchable and configurable along with the clock generator.

The chip features nine cores (1 serving in the MCU + 8 in the compute engine), with support for the integer multiplication and division instructions (M) and compressed instructions (C) standard extensions.

The GAP8 a private L1 cache for the MCU core which consists of a 16 KiB of data cache and 4 KiB of instruction cache. The compute engine has a shared level 1 cache of its own which consists of a 16 KiB instruction cache and a 64 KiB data cache. Additionally, the entire chip shares a 512 KiB level 2 cache consisting of 4 128 KiB cache banks.

### Compute Engine

The compute engine consists of eight additional cores clustered together to form a low power but powerful computational engine. The engine sits on an entirely separate voltage and frequency domains which can be switched off when not operating or downclocked to suit a particular workload more efficiently.

### Features of GAP8

Gap8 is focused on efficient energy management, making it suitable for battery-powered or remotely deployed devices where energy is limited. And Gap8 is focused on efficient energy management, making it suitable for battery-powered or remotely deployed devices where energy is limited.

## The Neural Network

Neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer, rely on training data to learn and improve their accuracy over time[3].

* Feedforward network with a single layer of neurons:

形状, 多边形

描述已自动生成

Figure 2 Feedforward network with a single layer of neurons[4].

* Fully connected feedforward network with one hidden layer and one output layer:

图示

描述已自动生成

Figure 3 Fully connected feedforward network[5].

### Input Layer

The input layer is the first layer of the neural network and is responsible for receiving input data.

* **Function:** Directly process raw input data.
* **Data transformation:** The input layer usually does not make any changes to the data and passes it directly to the next layer.

### Hidden Layer

Hidden layers are located between the input layer and the output layer. They are where the neural network performs calculations and feature extraction. There can be one or more hidden layers.

* **Function:** Processes data and extracts features.
* **Operation:** Each hidden layer contains multiple neurons, and each neuron is connected to all nodes from the previous layer with weights. Each neuron receives inputs from the nodes of the previous layer, applies weights, and produces an output through an activation function. This output becomes the input for the next layer. This process repeats, allowing the network to learn representations of the input data by adjusting the weights.
* **Weights:** Each connection has a weight, which represents the strength of the connection. During training, these weights are optimized according to the algorithm so that the network can perform tasks accurately.
* **Activation Function:** Introduces non-linearity to the network, enabling it to learn more complex patterns. Common activation functions include sigmoid, tanh, and ReLU.

### Output Layer

The output layer is the last layer of the neural network and is responsible for outputting calculation results.

* **Function:** Provides the final output of the network, representing the solution to the problem.
* **Operation:** Each output node corresponds to a possible output category or value.

# Realization

## Code Link.

* Realize a simple neural network.

<https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP1.c>

* Implement the simple neural network on GAP8.

<https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP2.c>

* Implement the neural network on GAP8 with Performance counters.

<https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP2-monitor.c>

* flexSensorData Test.

<https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP3.c>

* flexSensorData Test with Hidden\_Layer.

<https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP3-2.c>

* Multi-core asynchronous control

<https://github.com/heyPetiteF/IOT/blob/main/helloworld/FP4.c>

## Analysis of Implementation Process and Results.

### Realize a Simple Neural Network.

A simple neural network example implemented in C language was run and the following results were obtained:

文本

描述已自动生成

Figure 4 The Output of a Simple NN (1).

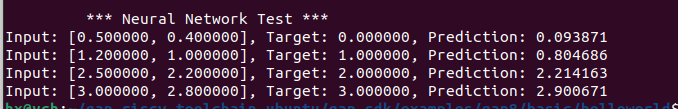


Figure 5 The Output of a Simple NN (2).

Looking at the overall output results, the predicted value of the neural network is quite close to the target value, which indicates that the training process is successful, and the network is able to learn the relationship between the input data and the output data.

### Implement the Neural Network on GAP8.

#### Implement the Simple Neural Network on GAP8.

Running a simple neural network on GAP8 gives the following results:

文本

描述已自动生成

Figure 6 The Output of a Simple Neural Network on GAP8.

Analysis of these results shows that the neural network can produce predictions close to the target value based on the training data, indicating that the learning process is effective.

However, some values of 0.0 for weights may mean that some inputs have very little impact on the output or have been ignored by the network, which can be due to some weights contributing less to error reduction during the optimization process of the network. The final deviation value is negative, which has a small adjustment effect on the final forecast value.

Additionally, "Performance Cycles" and "Timer Cycles" provide information about the efficiency of program execution. A lower number of cycles usually means higher performance.

Finally, successful tests show that the implemented training strategy can effectively implement the forward propagation and back propagation algorithms of neural networks on GAP8 hardware.

#### Implement the NN on GAP8 with Performance Counters.

In order to realize the function of performance counter, based on the code of “Implement the simple neural network on GAP8.”, added the following three functions and run these in main function.

图片包含 文本

描述已自动生成

Figure 7 Three Functions of Performance Counter.

文本

描述已自动生成

Figure 8 Called in Main Function.

However, during the running process, encountered the problem that the time function cannot be called. To capture the data of time tried using the following two functions:

1) Using function of PMSIS “pi\_time\_us()”

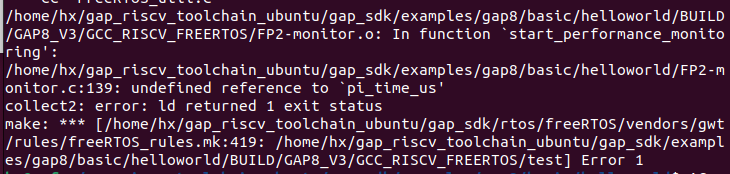


Figure 9 Unable to Call the Function pi\_time\_us().

2) Using function of C language “clock ()”

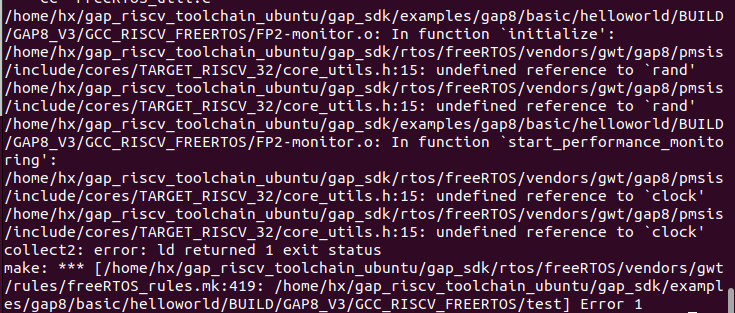


Figure 10 Unable to Call the Function clock().

The code compiles successfully but the result is not outputted due to the target function not being invoked.

### Use flex sensor data as training and test data.

#### flexSensorData Test.

The following procedures were implemented for training the neural network on GAP8 using the data in the flexSensorData file:

1. The flexSensorData.csv was opened with fopen().
2. Each line of the file was read using fscanf().
3. The parsed data were stored in the training\_data and targets arrays.
4. The file was closed with fclose() upon completion of the data import.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure 11 Code to Import .csv Data Using Function.

However, a problem occurred during the reading process, the function could not be called accurately, terminal displayed as follows:

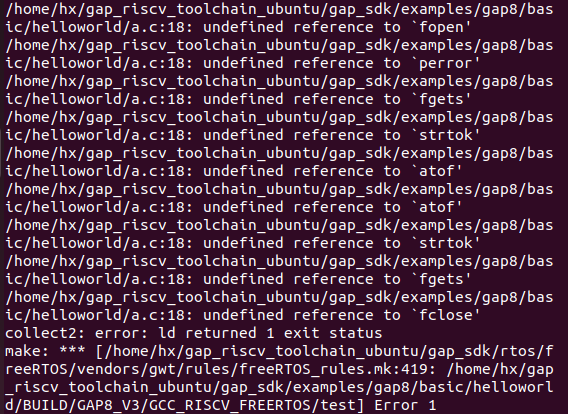


Figure 12 Cannot be Called from Library Function.

Therefore, modifications were made to directly introduce the data into the neural network calculation, the following results are obtained:

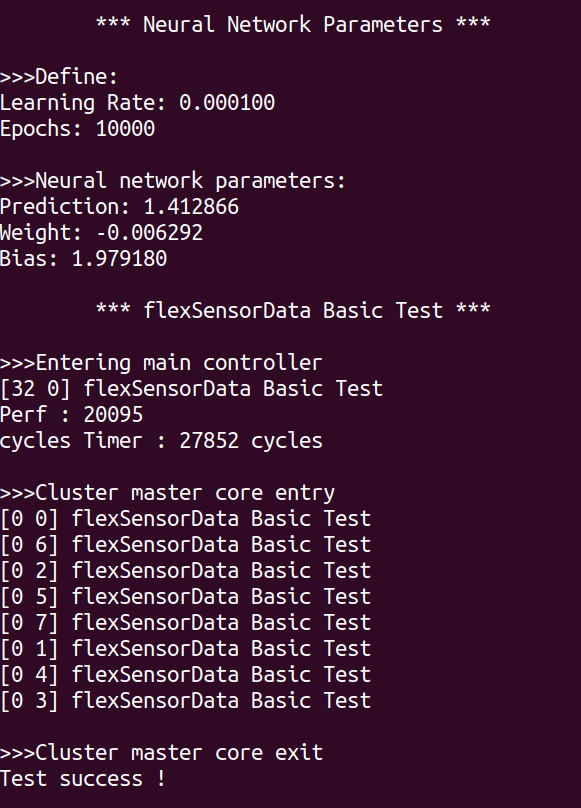


Figure 13 NN Data from flexSensorData.csv (Epochs: 10000).

Considering that the weights are less than zero, the bias is greater than one, and the number of training epochs is set to a relatively high value of 10,000, there may be overfitting in the program.

Therefore, the epochs value is reduced to 100, and the results are as follows:

文本

描述已自动生成

Figure 14 NN Data From flexSensorData.csv (Epochs:100).

The specific values of the weights and biases of the neural network after a certain number of trainings provide some information about the state of the network after training. Performance indicators indicate that the program can run on the cluster and the execution time is reasonable.

#### flexSensorData Test with Hidden Layer.

Considering that neural networks with hidden layers have stronger learning capabilities and can theoretically approximate any continuous function, we made the following attempt.

Based on the data in the flexSensorData.csv file, a neural network with hidden layers was trained on GAP8 and the following results were obtained:

文本

描述已自动生成

Figure 15 NN with Hidde\_Layer.

### Multi-core asynchronous control.

文本

描述已自动生成

Figure 16 Simple NN Run in the 1st Core.

文本

描述已自动生成

Figure 17 Stochastic NN Run in the 2nd Core.

文本

描述已自动生成

Figure 18 NN Forward Propagation with Hidden Layer Run in the 3rd Core.

文本

中度可信度描述已自动生成

Figure 19 NN Back Propagation with Hidden Layer Run in the 4th Core.

文本

中度可信度描述已自动生成

Figure 20 Linear Regression Run in the 5th Core.

文本

描述已自动生成

Figure 21 Logistic Regression Run in the 6th Core.

文本

描述已自动生成

Figure 22 Multi-Layer Perceptron Run in the 7th Core.

文本

描述已自动生成

Figure 23 Simple Classification Run in the 8th Core.

Run the code and get the following results:

文本

描述已自动生成

Figure 24 Results of Function Run in Different Cores

It can be seen from the result diagram that the titles and output results of functions with different functions running in different Cores are mixed and are not output in code order.

Each “function\_core\_n(void \*arg)” in the output results is executed on a different core, but their output is redirected to the same standard output stream: Terminal. Therefore, since all cores share the same output stream, the output results are displayed continuously rather than separated by core. Their output is displayed consecutively in the order of execution, not in code order.

# 3. References

1. GAP8 – GreenWaves, <https://en.wikichip.org/wiki/greenwaves/gap8>
2. File: gap8 block diagram.svg, <https://en.wikichip.org/wiki/File:gap8_block_diagram.svg>
3. What are neural networks? <https://www.ibm.com/topics/neural-networks>
4. Haykin, S. (2009). Neural networks and learning machines (3rd ed., p. 21). Upper Saddle River, NJ: Pearson Education, Inc.
5. Haykin, S. (2009). Neural networks and learning machines (3rd ed., p. 22). Upper Saddle River, NJ: Pearson Education, Inc.
6. https://greenwaves-technologies.com/manuals/BUILD/PULP-OS/html/index.html#section1