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

Thesis proposal

DESIGN OF RISV-V BASED EDGE-AI DEVICE

COUNCIL:	COMPUTER ENGINEERING
INSTRUCTOR:	Assoc.Prof TRẦN NGỌC THỊNH
REVIEWER:	

Student 1: Văn Ngọc Thanh Tùng

Ho Chi Minh City, November 2023

COMMITMENT

We pledge that this project is the result of my own research and work with the advises from our group's supervisors, Assof.Prof. Trần Ngọc Thịnh and Mr. Huỳnh Phúc Nghị. All the reference are cited properly and all the numbers and statistics are reliable and honest. Our group has completed the requirements for the thesis set out by the Faculty of Computer Science and Engineering.

Sincerely,
Project's members:
Văn Ngọc Thanh Tùng

ACKNOWLEDGEMENT

We would like to express our appreciation to our supervisors, Assof.Prof. Trần Ngọc Thịnh and Mr. Huỳnh Phúc Nghị for helping us in this project. Thanks to the knowledge from Mr. Thịnh and Mr. Nghị, our group was able to complete this project and achieve the requirements set out by the Faculty. We also appreciate the knowledge that we have learnt during the time in the university from all the teachers in the school, especially the teachers from Faculty of Computer Science and Engineering

We also want to express our sincere to Mr. Ngô Minh Hồng Thái, the author of the 2023 thesis "Tích hợp AI trên thiết bị biên dựa trên RISC-V cho các ứng dụng IoT"from the Ho Chi Minh University of Technology. His thesis set the foundation for the main purpose of our thesis, which is the improvements and application implementation of the AlPicoSoC system. Without his work and advices, we cannot complete this project as well as this thesis.

Last but not least, we would like to thanks our friends and family. With their help and support, we are able to overcome the difficulties in the process of researching for the thesis while studying in the university. While not directly helping us in the thesis, our friends and family provide us with all the support they can in order for us to complete this thesis with the best result possible.

Sincerely,
Project’s members:
Văn Ngọc Thanh Tùng

Table of contents

List of figures

List of tables

CHAPTER 1

INTRODUCTION

1.1 Abstract:

Nowadays, AI and its applications plays an important role in our everyday life. The release of ChatGPT marks the start of a new era where the impact of AI, especially generative AI is more significant that ever. Therefore, we can see that the rise of AI can open up some new interesting application that can drastically changed our everyday life. But the rapid improvement of AI technologies demands faster and more capable hardware, which have pushed the semiconductor industry to its limit.

Fortunately, beside the breakthrough of AI technology, the semiconductor industry also saw a new trend that is quite popular, that is the rise of open-source Instruction Set Architecture (ISA), especially RISC-V. What make RISC-V special is the fact that this is an open-source ISA, which means everyone can use as well as freely modified it, thus provides advantages that some proprietary ISA can not. With this in mind, we can design a custom chip easily with the help of FPGA (Field Programmable Field Array).

With the rise of AI and RISC-V as well as FPGA, it is now possible that we can design an AI Processing Unit (APU) with an extensible ISA for a relatively cheap price. This can open up a new opportunity for AI technology that traditional and proprietary ISA cannot provide, that is unmatched accessibility and extensibility.

1.2 Project's goal and scope:

To solve the problems that AI is having right now, we try to investigate, implement, and if possible, improve an AI application chip with the help of a CPU core that support the RISC-V instruction set.

Our goal is utilizing the new technologies in order to create an improvable, extensible and affordable AI device. From there, we hope that this project can help improving the AI as well as the semiconductor industries.

Because our project is quite ambitious and with the lack of tool as well as man power, we limit the scope of this project to just improve a previous design. The design that we will try to improve is the AIPicoSoC, implemented by Mr. Ngô Minh Hồng Thái from his graduated thesis.

To complete this project, we divide it into 2 phases:

- **Phase 1:** We will investigate various RISC-V core as well as some neural network model to have a basic understanding. Then we will investigate the design of AIPicoSoC to find out its flaws. From there, we will select some feasible one to try and improve it.
- **Phase 2:** Based on our selections, we will implement our solution. The goal of this phase is to successfully implement the features proposed from Phase 1 and compare the modified version with the original.

CHAPTER 2

BACKGROUND

2.1 Artificial Intelligence

Artificial Intelligence (AI for short), according to Encyclopedia Britannica, is "*the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings*"[Cop23]. Just like a human mind, in order to gain "intelligence", a machine needs to gather information through **Learning**, uses those information to perform a decision by **Reasoning**, and finally, knows to correct itself with **Self-correct**.

The rise of artificial intelligence started in the year 1958 with the implementation of the Perceptron by Frank Rosenblatt. The Perceptron is a simple neural network that took inspiration from the human neural system. The Perceptron "*is designed to illustrate some of the fundamental properties of intelligent system in general*"[RP21]. Since then, with the development of the computer infrastructure as well as AI algorithm, more and more larger and more complex artificial neural networks have been created. Today, with the introduction of ChatGPT and other generative AI based on large language model, the immense potential of AI has been discovered and utilized.

Today, there are some popular artificial neural network that have many application in our life, such as: Feedforward Neural Network (FNN), Convoluted Neural Network (CNN), Recurrent Neural Network (RNN), etc

2.1.1 Perceptron

The Perceptron is an algorithm in supervised learning that is used for binary classifier. The concept was first implemented in 1957 by Frank Rosenblatt as the Alpha Perceptron. In the Alpha Perceptron, there are 3 systems, which is shown in the Figure 2.1:[Ros57]

- **S-system (Sensory system):** This is the input for the system, connected to the Association system by a series of connection that can be positive (carry signals that "excite"activities to the A-system) or negative (carry signals that suppress activities to the A-system).
- **A-system (Association system):** This system performs the switching function between the input and the output. The system takes the signals from the Sensory system, then, through a "weight"parameter that determine the necessary sum value of the input to activate the output, it will "decide"the output value.
- **R-system (Response system):** This system will take then output from the Association system and compare it to a "bias", which is a parameter that define the critical value that the sum or mean of the outputs needs to exceed in order for the output to be fired. If the sum or mean of the output is higher than the "bias", the output will be 1, otherwise it will be 0.

In Figure 2.2 and Figure 2.3, we can see the Perceptron with 3 independent and 3 binary output sets respectively.[Ros57]

Although simple, the Perceptron had create the foundation of artificial neural network and with that, the development of various neural networks and machine learning methods that are the backbone of the AI technology.

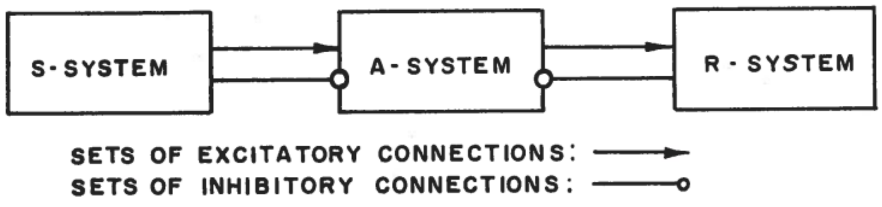
S-SYSTEM
A-SYSTEM
R-SYSTEM
SETS OF EXCITATORY CONNECTIONS:
SETS OF INHIBITORY CONNECTIONS:

Figure 2.1: General organization of the perceptron

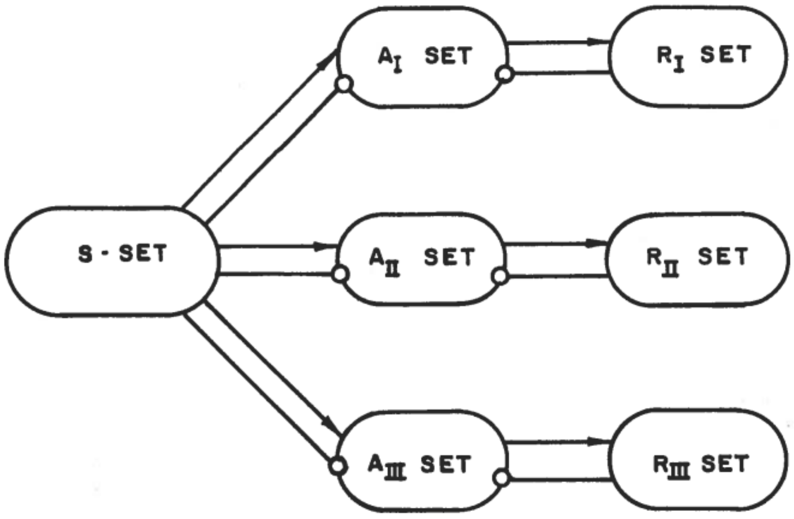

Figure 2.2: Organization of a preceptron with three independent output sets

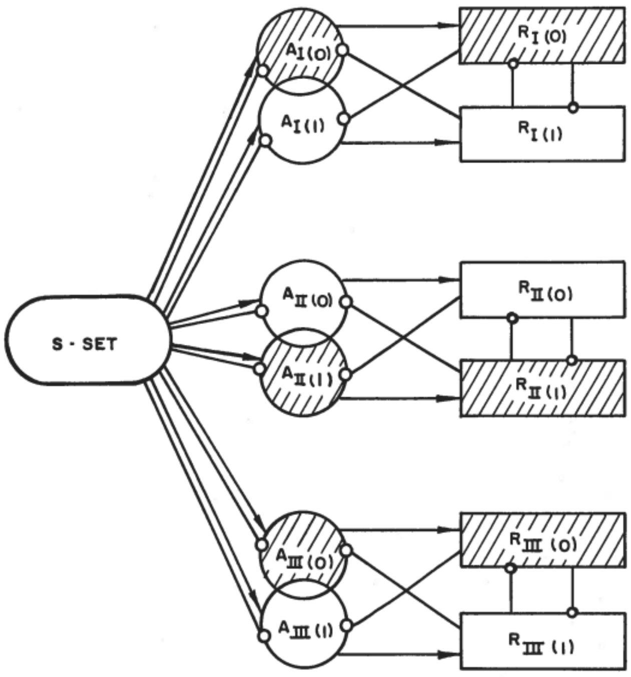

Figure 2.3: Organization of a preceptron with three binary output sets

2.1.2 Artificial neural network

Artificial neural network (sometime called neural network for short), first proposed in 1944 by Warren McCullough and Walter Pitts, is one of the most popular algorithm in machine learning. It is usually visualized as a graph with nodes and connected edges that formed a network simillar to a human nervous system. Nodes in this network are like a Perceptron, each node contained a weight that the input data will multiply with, then if the result is larger than the threshold, the nodes "fired".

There are several neural networks that are still see heavy usage in today's application. We can list several notable ones such as:

- **Feedforward Neural Network (FNN):** This is the first and the most simple type of neural network. The data will be transferred from one layer to another in one direction from the input through the hidden layers to the output without forming any cycles or loops.
- **Convolutional Neural Network (CNN):** This is a class of neural network, characterized by the presence of one or more convolutional layers in it.
- **Reccurent Neural Network (RNN):** This is a type of neural network that is quite simillar to the Feedforward neural network. The key difference between the two is that RNN have bi-directional dataflow while FNN's dataflow is uni-directional.

2.1.3 Machine learning

Machine learning is a field of study in artificial intelligence that concerns about development and implementation of several statistical algorithm that can help a machine to create a decision based on the pattern in a given input. Machine learning has been used in many application in our everyday life, such as computer vision, email filtering, speech recognition, etc. In artificial neural network however, we usually apply a special kind of machine learning, that is Deep learning, which is a machine learning method that is based on neural network.

Just like in programming, we also have many paradigms in machine learning. Some notable ones are:

- **Supervised learning:** A paradigm in machine learning where a model is trained using a combination of input and expected output. In supervised learning, we use a technique called backpropagation, which will constantly update the weights in the model until the model converged. This paradigm can help us in applications where we need to categorize the input data.
- **Unsupervised learning:** A paradigm in machine learning that, in contrast to supervised learning, train a model using non-labeled data. Instead of relied heavily in backpropagation like supervised learning, unsupervised learning also use some other learning rules like Hopfield, Boltzmann, Gibbs samplings, etc.
- **Semi-supervised learning:** Also called Weak-supervised learning, this paradigm is a combination of both supervised and unsupervised learning paradigm. In this learning paradigm, only some of the output is labeled, while the others are unlabeled or imprecisely labeled. This paradigm sees the most used in some large language model in generative AI.
- **Reinforcement learning:** A paradigm that aims to train a model to choose the most optimized solution for a problem. This paradigm is used when the mathematical model is infeasible.

Beside the ones that we mentioned above, we also have some other learning paradigms such as: Self-learning, Feature learning, etc.

2.2 RISC-V

RISC-V is an open-source instruction set architechture (ISA) based on reduced instruction set computer (RISC) principle. Unlike complex instruction set computer (CISC) whose every instructions contain many low-level operation, RISC instruction only do 1 low-level operation per instruction. Because of this, to complete a task, a RISC computer may need more than 1 instructions while a CISC computer can complete the task only in 1 instruction.

	RISC	CISC
Register	16-32 general purpose	8-16 general purpose
Data types	Usually two (integer and float)	Usually more than 2
Instructions	Load-store, operations on datatypes in registers	Corresponse to datatypes
Instruction format	Fixed length	Variable length
Encoding	1 instruction = 1 operand or operation	1 instruction = 1 statement
Design objective	Trade off program length for time to execute	Minimize program length for work/instruction
Implementation	Hard-wired Processor, instructions take 1 cycle, simple pipeline	Micro-programmed processor, instructions take variable time, complex pipeline
Caching	Essential	Useful
Compiler design	Should stress best ordering	Should stress finding right instruction
Philosophy	Move all function to software	Move any useful function from software to hardware

Table 2.1: The comparision between RISC and CISC[Jam95]

Although it look like RISC computer is an inferior to CISC computer, RISC computer can optimize the speed for each instruction. Studied have shown that in CISC: "*25% of the instructions in the instructions' set make up 95% of the execution time.*"[Jam95]. From here, we can see that around 75% of the hardware-supported instructions are often not used. Another study conducted on an IBM 360 compiler that only 10 instructions accounted for 80% of all the instruction executed. For 16 instructions that number is 90%, 95% for 21 instructions and 99% for 30 instructions[Jam95]. If we can eliminate the extra instructions that are never used, we can improve the performance as well as the area in a chip. However, RISC also have some disadvantages when compare to the CISC. Since one instruction only accounted for 1 operation in RISC, in order to execute a task, a RISC computer can take more instructions than CISC, increasing the program code as well as memory usage. Moreover, since RISC use more register than CISC, we need a more complicated register's address decoder.

Beside the adventages and disadvantages of the RISC principle, RISC-V also have another advantage that we can consider, that is the fact that this is an open-source ISA, which mean we can freely use and modify the design. This accessibility as well as extensibility allow RISC-V to be highly customizable. Moreover, RISC-V also provide us with an ISA that avoids "over-architecting", allow efficient implementation to ASIC or FPGA; be simple and affordable for educational purposes as well as reduce complexity in new implementations. Because of this, the popularity of RISC-V have exploded over the year, became one of the most popular open-source ISA available as well as open the door of chip design to everyone. For that reason, we have the opportunity to access a lot of RISC-V cores and extensions, such as:

- Base integer instruction set with RV32I, RV64I
- "M"extension for multiplication and division
- "A"extension for atomic instructions
- "F"and "D"extensions for single and double precision floating point numbers

Beside these cores and extensions, there are also numerous custom cores made by various developers and organizations. Some notable examples are VexRiscv, CVA6, Ibex, Rocket, etc. All of these custom cores are all base on the base core and its extensions.

CHAPTER 3

RELATED WORK AND RESEARCH

3.1 About the AIPicoSoC:

As we mentioned in the last section, there are several RISC-V custom cores and SoC that are readily available for us to use. But in this project we mostly care about the AIPicoSoC, a System-on-chip that is based on the PicoRV32 core.

3.1.1 AIPicoSoC:

The AIPicoSoC is a System-on-Chip implemented by Mr. Ngô Minh Hồng Thái in 2023. It is based on the PicoSoC with the addition of an AI Accelerator. The components of AIPicoSoC can be found in Figure 3.3:[Thá23]

- **PicoRV32:** This is a RISC-V core that is the main control unit in this system. It is responsible for computation operations, memory management as well as peripherals communication control.
- **Alpha Accelerator:** This is a Deep Learning Processor, which is a specialized computational unit. Its role is to help running the deep learning algorithm more efficiently.
- **Alpha Accelerator Pico Controller:** This module helps simplify the communication between the PicoRV32 core and the Alpha Accelerator.
- **IMEM:** This is a read-only instruction memory where we store the instructions for the PicoRV32.
- **DMEM:** This is a volatile memory, serve as the temporary memory

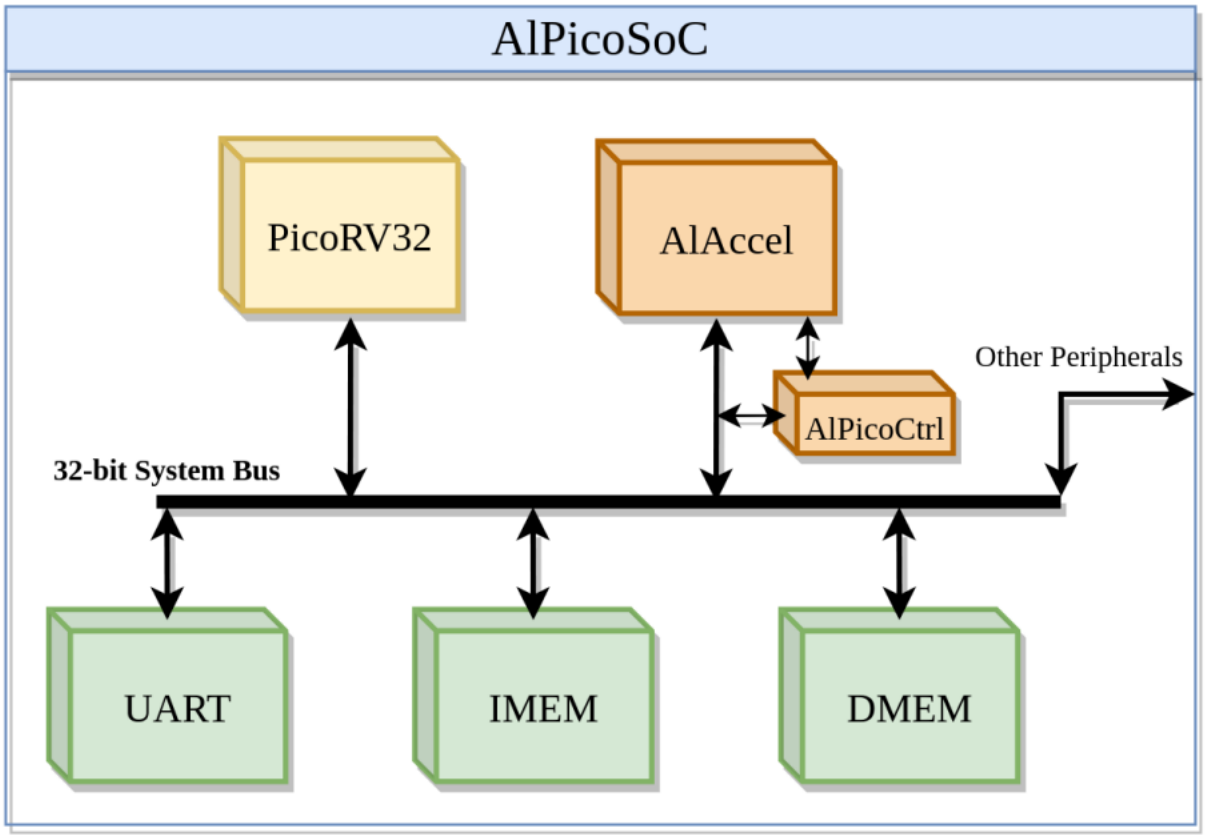

Figure 3.1: Overview of the AIPicoSoC

for the PicoRV32 when it execute the instructions. It is designed as a Dual-port RAM which allow read and write data simultaneously.

- **UART:** This is the only peripheral for the AIPicoSoC. It allows the system to communicate with other hardware components such as sensors, micro-controler, etc
- **32-bit System Bus:** This bus is a simple bus that connects all the module of the ALPicoSoC together.

3.1.2 PicoRV32 and PicoSoC:

PicoRV32 is a CPU core that is implemented by using RV32IMC instruction set. The core can be configured into RV32E, RV32I, RV32IC, RV32IM, or RV32IMC core. The features of this core include:[Wol]

- Small (750-2000 LUTs in 7-Series Xilinx Architecture)
- High f_max (250-450 MHz on 7-Series Xilinx FPGAs)
- Selectable native memory interface or AXI4-Lite master interface. The native memory interface includes:

  - A 1-bit output *mem_valid* signal
  - A 1-bit output *mem_instr* signal
  - A 1-bit input *mem_ready* signal
  - A 32-bit output *mem_addr* signal
  - A 32-bit output *mem_wdata* signal
  - A 4-bit output *mem_wstrb* signal
  - A 32-bit input *mem_rdata* signal
- Optional IRQ support (using a simple custom ISA)
- Optional Co-Processor Interface
- Have 3 variation, *picorv32*, *picorv32_axi* and *picorv32_wb*. The first one provide a simple native memory interface, while the second and third ones provide AXI-4 Lite and Wishbone master interface respectively.
- Dhrystone benchmark is 0.516 DMIPS/MHz

From the features mentioned above, it is clear that this core is a size optimized core that can work with high frequency clock signal. Moreover, the core is highly configurable with multiple extensions and interfaces. However, the biggest drawback of this core is the performance. With the Dhrystone benchmark of only 0.516 DMIPS/MHz at the highest performance setting, lower than most popular RISC-V core nowadays, it is obvious that this core is not build for high performance application in mind. Because of this, the PicoRV32 core is suitable for application where intense calculation operations are not needed and size is preferred.

The PicoRV32 core does not have a lot of SoC based on it, but one of the most used one is PicoSoC. PicoSoC is a simple SoC (System-on-chip) that can run instructions directly from a SPI flash chip. The SoC includes:

- PicoRV32 with RV32IMC ISA
- Execute-in-place(XIP) SPI flash controller
- A single UART interface
- A single 1kB SRAM

Figure 3.1 demonstrate the PicoSoC system. PicoSoC can be connected to another module through a 32bit native system bus, which can help us extend its application.

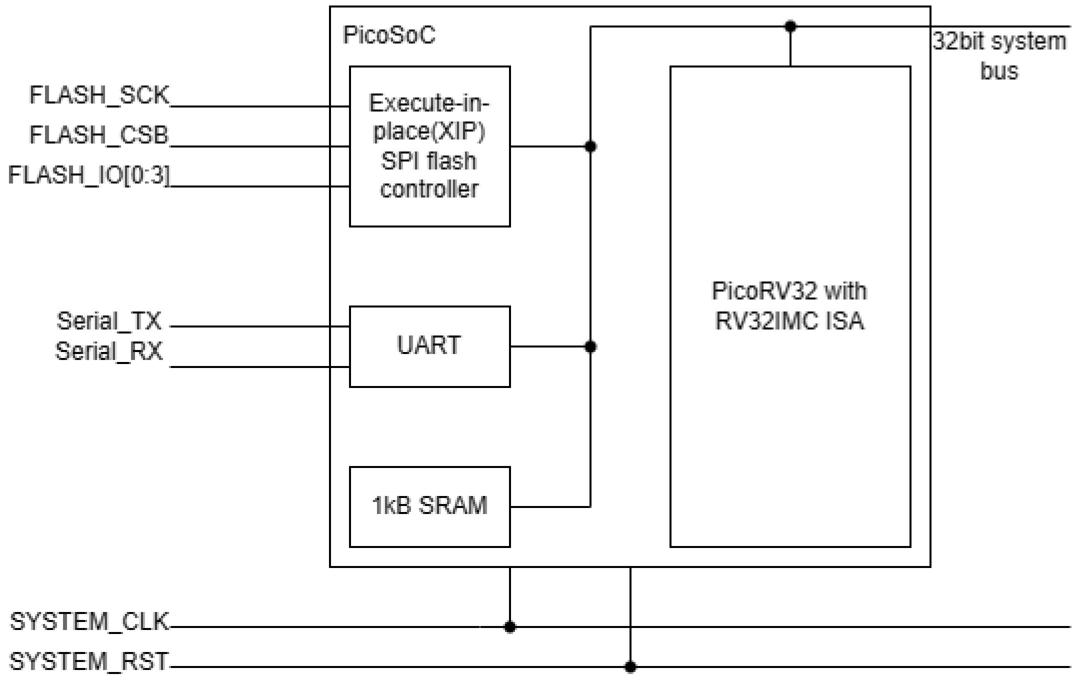

Figure 3.2: The overview of the PicoSoC system

3.1.3 Alpha Accelerator:

The Alpha Accelerator is a Deep Learning Processor, serves as the AI accelerator for the AIPicoSoC system. Some features of the Alpha Accelerator include:[Thá23]

- Calculate Deep learning algorithm at Layer level, which mean it will get the input off a layer and it will output the output of that layer. The accelerator can support 3 types of layer (Fully connected, Convolution and Max-pooling layer) and 2 type of activation functions (ReLU and ReLU6)
- Can support integer quantization to 8-bit integer.
- Can perform read data, compute and write back operations in a 3-stage pipeline: RDATA-COMP-WBACK

The overall architecture for the Alpha Accelerator can be found in Figure 3.2

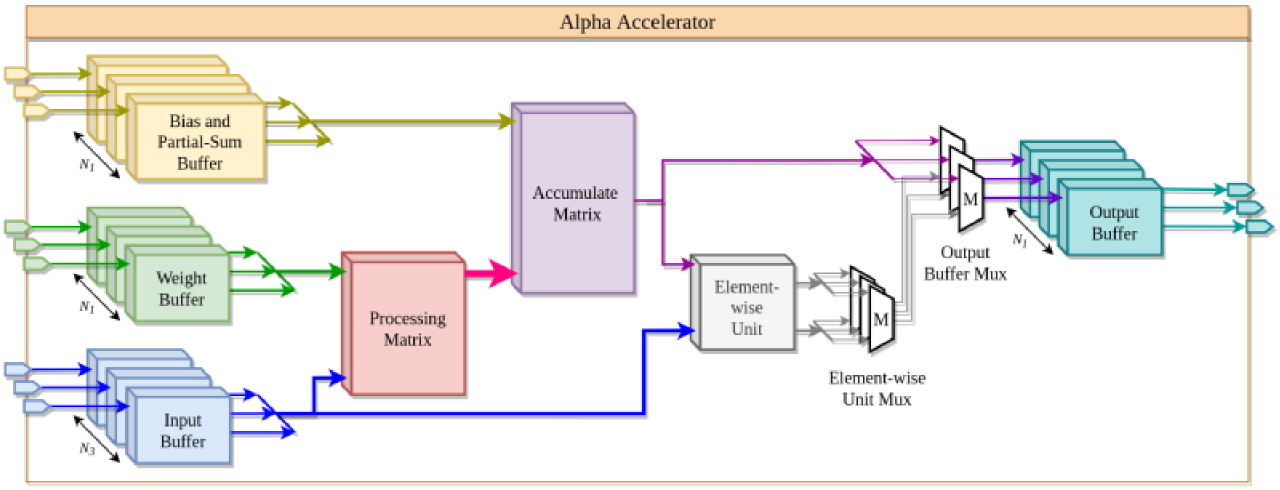

Figure 3.3: Overview of the Alpha Accelerator

Based on their functions, we can divide the Alpha Accelerator into 3 blocks:

- **Computation block:** This block consists of the following modules: Processing Matrix, Accumulate Matrix, Elemental-wise Unit.

- *Processing Matrix:* The most important component of the accelerator. This module is responsible for the computation of the multiply accumulate (MAC) operations.

- *Accumulate Matrix:* This module is used to accumulate the results from the Processing Matrix.

- *Elemental-wise Unit:* This module is used for single-operation computation such as rounding, activation functions, etc.

- **Data reading block:** This block include the following modules: Bias and Partial-sum Buffer, Weight Buffer and Input Buffer

- *Bias and Partial-sum Buffer:* A buffer that store the Bias value as well as the Partial sum value used for the computation cycle.

- *Weight Buffer:* A buffer that stores the weight value (for the Fully-connected layer) or the kernel (for the Convolutional layer)

- *Input Buffer:* A buffer that stores the input values of the system.

- **Data writing block:** This block only have an Output Buffer module, which is responsible for storing the output value of the Alpha Accelerator.

- Other components, include: Output Buffer Mux and Elemental-wise Buffer Mux.

3.2 About the Tensorflow Lite as well as other software tools:

Tensorflow is a platform for machine learning application. It specialized in multidimensional array (Tensors) based computation as well as allow the user to model, train and export any machine learning model. Tensorflow Lite is a set of tool based on Tensorflow that enables developers to run their model on the mobile, embedded as well as edge device. Some key features of Tensorflow Lite include:

- Optimized for on-device machine learning by addressing 5 key constraints: latency, privacy, connectivity, size and power consumption
- Multiple platform as well as diverse programming language support
- High performance

In order to train the AI model, beside Tensorflow, we also need a dataset. The dataset used for the training of the AI model in AIPicoSoC is the MNIST dataset, which contains many black and white image of handwritten number digits commonly used for training various image processing system.

Once we have the library and the dataset, we can start the trawining process for our Deep learning model. But in order to compile the model into machine code for the RISC-V core to run, we need another specialized tool. That tool is the RISC-V GNU Compiler Toolchain, which can compile C/C++ code into machine codes that can run on RISC-V core.

CHAPTER 4

EXISTING PROBLEMS AND SOLUTION PROPOSAL

4.1 Existing problems in the AIPicoSoC system:

Based on the information we have gathered in Section 3, we can notice some flaws in the design of the AlPicoSoC. Those flaws are:

- **Lack of a standardized bus system:** The bus system for the AlPicoSoC system is just a simple valid-ready interface databus. Although it can help transmit the data properly, this bus does not follow any standard bus protocol, thus can make it difficult for us to integrate other modules or systems into AlPicoSoc. Moreover, this bus can only make single 32-bit data transfer at a time, which can limit the potential of our system.
- **Low performance core:** The AlPicoSoC system only contains one PicoRV32 core, which is not well known for its performance. Below is a table that compare the performance by of PicoRV32 core with two other very popular RISC-V core, VexRiscv[Spib] and Ibex[low]. For ease of comparison, we will use Dhrystone and CoreMark benchmark:

Core	**Dhrystone (Highest performance)**	**CoreMark (Highest performance)**
PicoRV32	0.516 DMIPS/MHz	N/A
VexRiscv	1.57 DMIPS/MHz	2.57 CoreMark/MHz
Ibex	N/A	3.13 CoreMark/MHz

Table 4.1: Comparision between cores configured at the highest performance

From the table, it is clear that PicoRV32 is not a high performance core when compared with two other popular RISC-V cores such as VexRiscv and Ibex. This lack of performance, combine with the fact that there is only one RISC-V core in the AlPicoSoC system, can negatively affect the performance of the AlPicoSoC in AI applications.

- **Low number of peripherals:** Right now, the AlPicoSoC system only support 1 peripheral connection which is UART. This lack of peripheral support can affect our system's utility.
- **Small dataset for Deep learning:** The Deep laerning model for the Alpha Accelerator in AlPicoSoC only use the MNIST dataset for training. The MNIST dataset is not a very large dataset, consisted of only a few black-and-white picture of hand-written number digits. With this small dataset, we cannot have a good Deep learning model for any application.

4.2 Solutions for improvement of AlPicoSoC:

With the weaknesses of the AlPicoSoC system discussed in the previous section in mind, in order to address those and to implement some practical implementation for the system, we propose a few solutions for improvement. These improvements are:

4.2.1 Improve the Deep learning model as well as the Alpha Accelerator:

4.2.1.1 Deep Learning model:

Since our Deep learning model is trained using only MNIST data set, which is not a very large data set, our model can only recognize hand-written digit. This is simply not enough if we want to run any practical application on our system. In order to improve this model, we suggest using another data set for training. For this purpose, we propose a few data set that are available:

- **EMNIST:** The EMNIST, stand for Extended MNIST, is a data set that derived from the NIST Special Database 19 which contains handwritten digits and characters from over 500 writers.
- **ImageNet:** An image data set organized according to the WordNet hierarchy. Each meaningful concept in WordNet is call a "synonym set"and in WordNet, there are more that 100.000 synonym sets. For each syn-

onym set in WordNet, ImageNet provide around 1000 images and label.

- **Google's Open Image data set:** An image data set from Google. This data set has around 9 millions URLs to images which has been labeled, spanning over 6000 categories. Today, Open Image has developed to Open Image V6, which expands the amount of image to around 125,000 images.
- **MS COCO (Common Objects in Context):** A large scale data set used for object detection, segmentation and captioning. This data set contains a wide variety of common object image, which can help a lot in training our model for pratical application.

4.2.1.2 Alpha Accelerator:

For the Alpha Accelerator, there are some improvement point that can be considered. Right now, we consider adding more pipeline stages to the Alpha Accelerator. Right now the Alpha Accelerator has 3 pipeline stages: *RDATA*, *COMPS* and *WBACK*. The *RDATA* stage will read and load the necessary parameters into the computational blocks. Then, the *COMPS* stage will execute the MAC computational operations and finally, the *WBACK* stage will compare the values for the pooling layers, quantize the output and feed it into an activation function. Then the data will be stored in the output buffer. We suggest turning this into a 5-stage pipeline model *RDATA - LOAD - COMPS - ACTIVE - WBACK* to reduce the amount of task the Accelerator must do in some stages. Each stage of the model will responsible for the following task:

- *RDATA:* Read the input, weight as well as the bias and partial sum values.
- *LOAD:* Load the read value into the computational blocks
- *COMPS:* Calculate the MAC for the Fully-connected and Convolution layers
- *ACTIVE:* Pooling and quantize the MAC results and feed it into the activation function
- *WBACK:* Write the output result into the output buffer

Beside increasing the number of pipeline stages, we can also consider increas-

ing the number of Processing Unit (PU) in the system. The system currently support 3 MAC units per PU, 3 PUs per Processing Array and 3 Processing Arrays per Processing Matrix. The figures below illustrate the components of the Processing Matrix, Processing Array as well as Processing Unit:
With this in mind, we can consider increasing the amount of Processing

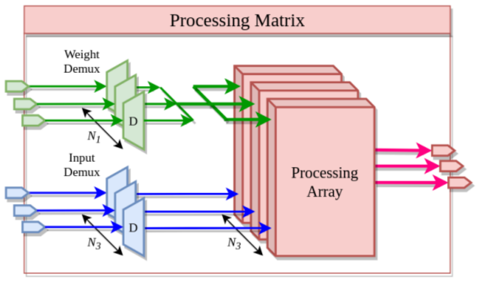

Figure 4.1: The overview of the Processing Matrix

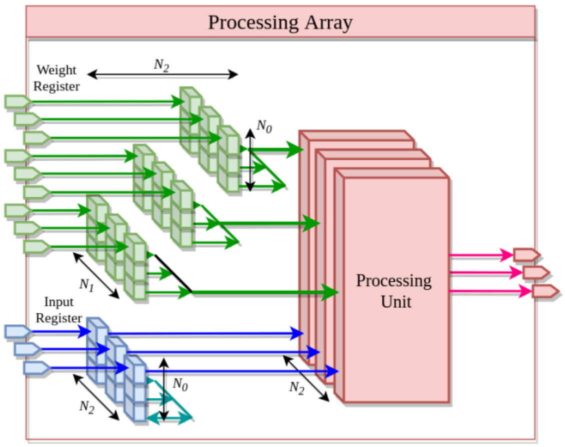

Figure 4.2: The overview of the Processing Array

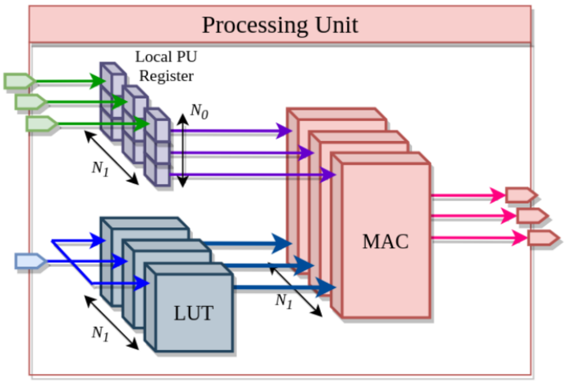

Figure 4.3: The overview of the Processing Unit

Array, Processing Unit as well as MAC components to improve the computational performance of the accelerator.

4.2.2 Improve the data bus of the AlPicoSoC to AXI-4 Lite bus:

The bus of the AlPicoSoC system is not a standardized bus, which can cause some difficulties in integrating new modules or system into the AlPicoSoC. Because of it, we will try to standardize the bus system by replacing the current bus with the AXI4-Lite bus, a subset of AXI bus protocol, one of the most popular bus protocol for communication between modules in a chip.

Introduced by Arm in 2003, the Advanced eXtensible Interface (AXI) protocol is a bus protocol targeted for high-frequency, high-performance system design. The objectives of AXI protocol are:

- To be suitable for high-bandwidth low-latency designs
- Enable high-frequency operation without using complex bridges
- Meet the interface requirements of a wide range of components
- Be suitable for memory controllers with high initial access latency
- Provide flexibility in implementation of interconnect architectures
- Be backward-compatible with previous AMBA bus protocols such as AHB and APB

Some key features of the AXI bus protocol include:

- Separated address phase and data phase
- Supports unaligned data transfer with the help of byte strobes
- Burst-based transactions using only start address
- Separated read and write data channel

The AXI4-Lite is a subset of AXI4 protocol, with some key features:

- All transactions are burst length of one.
- All data accesses are the same size as the width of the data bus.
- Exclusive accesses are not supported.

The AXI4-Lite protocol is intended for communication with simpler, smaller control register-style interfaces in components

The reason we choose AXI4-Lite is because AXI4-Lite is a standardized bus which saw a lot of usage in the design of chips and SoCs. With a standardized bus, we can communicate with other peripheral much more easier since we do not need to implement a bridge between our system and the peripherals. Moreover, with the AXI4-Lite, we can communicate much easier with ARM's system, most notably the ARM Cortex A9 processor which can be found on the Zynq-7000 FPGA. This can help us integrating our system into other already available system, increasing our system's utility

4.2.3 Add another RISC-V core in addition to the PicoRV32:

As we have mentioned in the previous sections, the PicoRV32 is not a very good RISC-V core in terms of performance. And in order to have any practical application for our system, we need another core with much higher performance. Luckily, PicoRV32 is a size-optimized core so we can treat the PicoRV32 core as a co-processor to a main core in our system. Some RISC-V cores that we are considering:

4.2.3.1 VexRiscv:

VexRiscv is a RISC-V CPU core implemented in SpinalHDL, which is a library for Scala programming language that is used to describe digital hardware. What make VexRiscv special is that it is done with a very software-oriented. This allow most users who are not very familiar with hardware description language an easier time getting used to the CPU core. The features of this core include:[Spib]

- RV32I[M][A][F[D]][C] instruction set, is configurable
- Pipelined from 2 to 5 or more stages
- High performance, with up to 1.57 DMIPS/MHz for Dhrystone benchmark or 2.57 CoreMark/MHz for CoreMark benchmark
- Have AXI-4, Avalon and Wishbone interface
- FreeRTOS port available
- Optional instruction and data caches
- Optional interrupts and exception handling with *Machine*, *Supervisor* and *User* modes

It is clear that VexRiscv is a more versatile core than PicoRV32 with better performance while keeping the same high configurability. The smallest version of VexRiscv while can be smaller that PicoRV32 (504 LUTs compared with 750 LUTs), provide mostly the same performance result (0.52 DMIPS/MHz with 0.516 DMIPS/MHz). However, one of the drawback of this core is the amount of sotfwares and tools needed to implement this core into the board, but nonetheless, this is a great core that can be used in a wide range of application.

4.2.3.2 Ibex:

Ibex (also known as Zero-riscy) is a open source 32-bit RISC-V CPU core written in SystemVerilog. The CPU core is designed to be highly parametrizable and suited for embedded control applications. Ibex supports the following extensions:

- Integer(I) or Embedded(E)
- Integer Multiplication and Division(M)
- Compressed instructions(C)
- Bit manipulation(B)

Beside that, the Ibex core also have some other features such as:

- Can be configured into 4 versions: *micro*, *small*, *maxperf*, *maxperf-pmp-bmfull*
- 3-stage pipeline (Instruction fetch, Decode and execute, Writeback)
- High performance (0.904 CoreMark/MHz for *micro*, 3.13 CoreMark/MHz for *maxperf* and *maxperf-pmp-bmfull*)

It is clear that this core is more powerful than VexRiscv and PicoRV32 while having a little less option for configuration. With its high performance in mind, it can be used in applications where we need strong computation power instead of optimal size.

4.2.4 Increase the number of peripheral for more utility:

To increase the utility of our system for further application, we suggest adding more peripheral support. Nowadays, there are a lot of serial communication protocols that are crucial in IoT application. Because our aim is to integrate our system into our daily life, we need to extend the amount of peripherals that our system support. Some serial communication protocols that we are considering is SPI and I2C. After that, based on the application, we can consider others peripherals if needed.

If we also improve the system bus to the AXI4-Lite protocol, we suggest connecting the peripheral with a APB bus, which will be interfaced to the AXI4-Lite bus through a bridge. The Advance Peripheral Bus (APB) is a part of the Advanced Micro-controller Bus Architecture (AMBA) just like AXI4-Lite. APB protocol is optimized for minimal power consumption and is used to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol.

4.3 Solution proposal:

4.3.1 System proposal: AlPicoSoC v2.0 and its application:

Based on the factors for improvements that we have mentioned in the previous sections, we propose a modified AlPicoSoC system, called AlPicoSoC v2.0. Below is the overview of our proposed system:

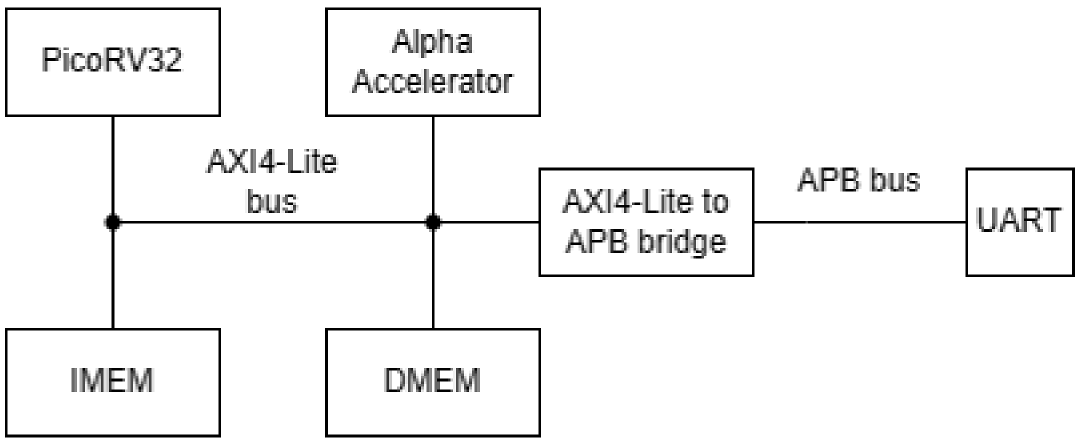

Figure 4.4: Overview of AlPicoSoC v2.0 system

In this system, we will replace the simple bus in AlPicoSoC to AXI4-Lite bus. The peripherals like UART will be connected to an APB bus, which will be connected to the main system through an AXI4-Lite to APB bridge. Moreover, we will also improve the Alpha Accelerator by increasing the number of pipeline stages from 3 stages (RDATA - COMPS - WBACK) to 5 stages (RDATA - LOAD - COMPS - ACTIVE - WBACK) as well as training our model with EMNIST data set to allow our model to recognize number digits as well as texts. These improvements are necessary for the implementation of our system's application.

For the application of the AlPicoSoC system, we suggest utilizing our system in the image recognition applications. To be more specific, we will develop an image recognition application where the system can detect handwriting characters as well as number digits captured from a camera. After that, we can consider upgrade it into text recognition and consider turning the AlPicoSoC v2.0 system into an ASIC (Application Specific Integrating Circuit) design.

4.3.2 Timeline for the next phase:

Below is the timeline that we are considering for the implementation of the proposed system. Note that this timeline is not fixed and can be changed in the future if needed:

Task	**Time expected**
Implementing the AXI4-Lite and APB bus system	3 weeks
Testing the system with the bus implemented	1 week
Increasing the number of pipeline stages in the Accelerator	3 weeks
Testing the system with the improved Accelerator	1 week
Train our Deep learning model with EMNIST data set	1 week
Testing our system with the new Deep Learning model	1 week
Implementing the application for the AlPicoSoC	3 weeks
Testing the application	1 week
Reserve weeks	2 weeks
Total time	**16 weeks**

Table 4.2: The estimated timeline for our next phase

List of reference

[Ros57] Frank Rosenblatt. *"The Perceptron—a perceiving and recognizing automaton"*. Cornell Aeronautical Laboratory., 1957.

[Pat85] David A Patterson. "Reduced instruction set computers". **in***Communications of the ACM*: 28.1 (1985), **pages** 8–21.

[Jam95] Tariq Jamil. "Risc versus cisc". **in***Ieee Potentials*: 14.3 (1995), **pages** 13–16.

[Fre99] R.E. Freund Y.;Schapire. *Large Margin Classification Using the Perceptron Algorithm.* Accessed 23 November 2023. 1999.

[CP16] Tony Chen **and** David A Patterson. "Risc-v geneology". **in***EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-6*: (2016).

[Kra+16] Ivan Krasin **andothers**. "OpenImages: A public dataset for large-scale multi-label and multi-class image classification." **in***Dataset available from https://github.com/openimages*: (2016).

[Coh+17] Gregory Cohen **andothers**. "EMNIST: an extension of MNIST to handwritten letters". **in**(2017).

[Har17] Larry Hardesty. "Explained: Neural networks". **in***MIT News*: (2017).

[Ert18] Wolfgang Ertel. *Introduction to artificial intelligence.* Springer, 2018.

[Sch+18] Pasquale Davide Schiavone **andothers**. "Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX". **in***2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*: 2018, **pages** 1–3. DOI: `10.1109/S3S.2018.8640145`.

[Shi+18] Dongjoo Shin **andothers**. "DNPU: An energy-efficient deep-learning processor with heterogeneous multi-core architecture". **in***IEEE Micro*: 38.5 (2018), **pages** 85–93.

[Pul+19] Antonio Pullini **andothers**. "Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing". **in***IEEE Journal of Solid-State Circuits*: 54.7 (2019), **pages** 1970–1981. DOI: `10.1109/JSSC.2019.2912307`.

[Alp20] Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2020.

[RP21] F Rosenblatt **and** S Papert. *Perceptron.* **volume** 9. April, 2021.

[Cop23] B.J Copeland. *"artificial intelligence"*. Accessed 22 November 2023. 2023. URL: `https://www.britannica.com/technology/artificial-intelligence`.

[Thá23] Ngô Minh Hồng Thái. *Tích hợp AI trên thiết bị biên dựa trên RISC-V cho các ứng dụng IoT.* 2023.

[Arma] Arm. *AMBA APB Protocol v2.0 Specification.*

[Armb] Arm. *AMBA AXI Protocol v1.0 Specification.*

[ETHa] Università di Bologna ETH Zurich. *PULP.* URL: `https://github.com/pulp-platform/pulp`.

[ETHb] Università di Bologna ETH Zurich. *PULPino SoC.* URL: `https://github.com/pulp-platform/pulpino`.

[ETHc] Università di Bologna ETH Zurich. *PULPissimo SoC.* URL: `https://github.com/pulp-platform/pulpissimo`.

[low] lowRISC. *Ibex RISC-V Core.* URL: `https://github.com/lowRISC/ibex`.

[Spia] SpinalHDL. *About SpinalHDL.* URL: `https://github.com/SpinalHDL/SpinalHDL`.

[Spib] SpinalHDL. *VexRiscv.* URL: `https://github.com/SpinalHDL/VexRiscv`.

[WA] Andrew Waterman **and** Krste Asanovic. *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA.*

[Wol] Clifford Wolf. *PicoRV32 - A Size-Optimized RISC-V CPU.* URL: `https://github.com/YosysHQ/picorv32`.