RISC-V SoC Microarchitecture Design & Optimization

Design Review 1 Report

Group 23

Instructor & Sponsor: Weikang Qian

Group Member:

Zhiyuan Liu	angelinaliu@sjtu.edu.cn
Jian Shi	timeshi@sjtu.edu.cn
Li Shi	shili2017@sjtu.edu.cn
Yiqiu Sun	susansun@sjtu.edu.cn
Yichao Yuan	yyc19990826@sjtu.edu.cn

Date: June 17, 2021

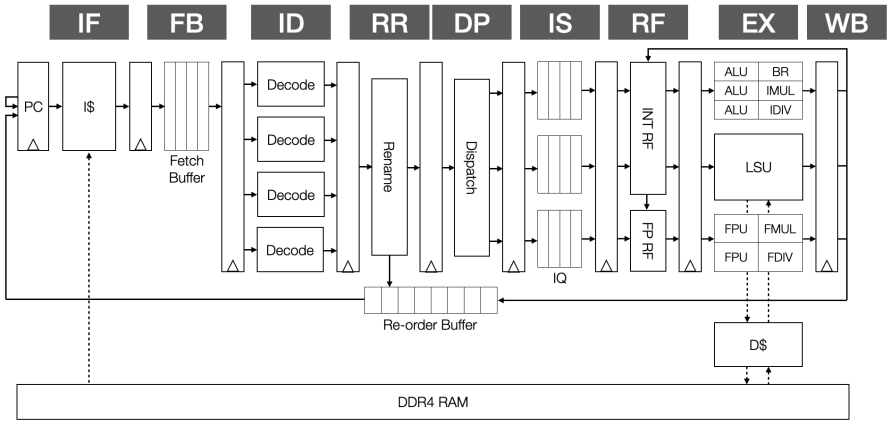


Figure 1: RISC-V core design draft

Abstract

People have continuously pursued higher performance of processors in the past few decades. However, due to physical and material limitations, Moore's Law is reaching its boundary soon, forcing engineers to make new attempts in the field of computer architecture. Domain-specific architecture design and optimization is increasingly popular recently with emerging technology in systems on-chip (SoC) design as well as the need for domain-specific applications. One of the main challenges is raised from the domain of AI-oriented embedded systems, as AI applications in embedded systems usually require high performance, low power consumption and low cost at the same time. In this project, we propose an SoC design which balance performance, power and cost for domain-specific computation-intensive tasks. First, RISC-V is chosen as our instruction set architecture (ISA), which offers simple but reliable design for processor designers to optimize for better performance and lower power. Second, based on RISC-V ISA, we will build an out-of-order processor core to exploit the performance for compute-bound applications. Third, we will apply advanced optimizations, e.g., approximate computing units, to enhance performance and energy efficiency for domain-specific applications. Finally, we will evaluate our SoC design on FPGA test platform in terms of performance and power consumption, and further improve our design according to test results. The outcome will be a complete optimized SoC working on FPGA test platform, running benchmark programs or domain-specific applications.

Keywords: computer architecture, system on-chip, embedded system, domain-specific design

Contents

1 Problem Description & Introduction **3**

2 Benchmark **5**

2.1 MIPS Processor with Classical 5-stage Pipeline 5

2.2 Rocket Core 5

2.3 The Berkeley Out-of-Order Machine 6

2.4 Hummingbird E203 6

3 Customer Requirements & Engineering Specifications **6**

3.1 Customer Requirements (CR) 6

3.1.1 General Requirements 6

3.1.2 Performance Requirements 8

3.1.3 Embedded System Requirements 9

3.1.4 Benchmark Competitions against CR 10

3.2 Engineering Specifications (ES) 10

3.2.1 Cross Correlation of ES 11

3.2.2 Benchmark Competitions against ES & Set Target for ES 11

3.2.3 Correlation from CR to ES 12

3.3 Quality Function Development (QFD) 13

4 Project Plan **14**

5 Conclusion **15**

1 Problem Description & Introduction

During the past decade, people have tried to build machines with better computing performance. Frequency scaling was first used to enhance the performance of the system. Since clock frequency is the "heart rate" of a computer, frequency ramping has traditionally been the dominant force in the improvement of performance for commodity processors from the mid-1980s to about the end of 2004 [11]. However, increases in frequency also lead to increases in power consumption. It was also because of the power wall that Intel canceled its Tejas and Jayhawk processors in 2004 [15]. Fig. 2 illustrates the improvement in clock frequency over time, measured in megahertz (millions of cycles per second). Clearly, clock-frequency improvements have been stalled and limited to about 3-4GHz due to power consumption nowadays [5].

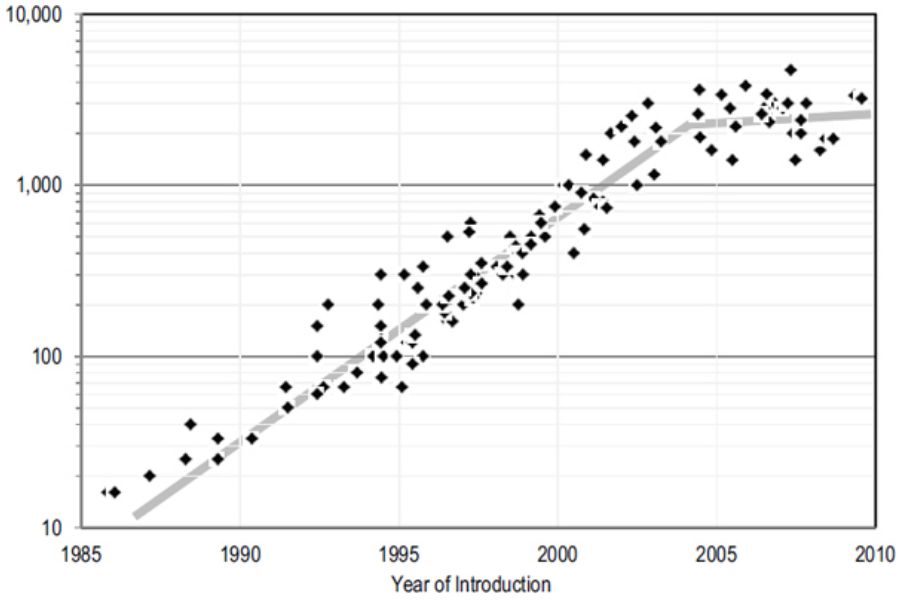


Figure 2: Microprocessor clock frequency (MHz) over time (1985-2010) [5].

Meanwhile, because of physical challenges and material challenges, scaling down the size of CMOS circuits reaches its boundary. The increase in tunneling and leakage currents disables engineers to develop chips with smaller transistors, thus making Moore's law come to an end. Besides, the inability of the dielectric and wiring materials to offer dependable insulation and conduction as the devices are becoming smaller [6] also plays an important role in fading Moore's law. Fig. 3 presents the number of transistors in a chip versus time and its relative performance.

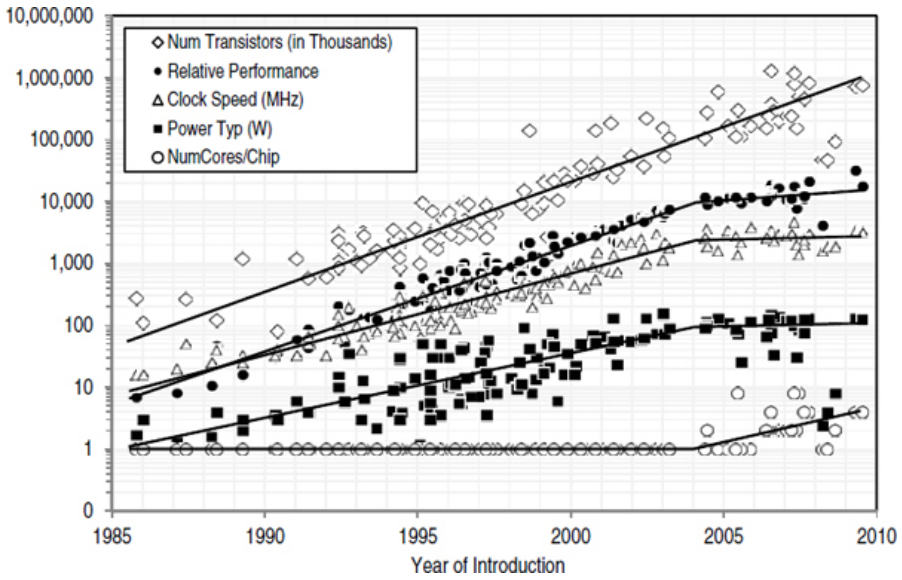


Figure 3: Transistors, frequency, power, performance, and cores over time (1985-2010) [5].

In order to continuously improve the computing performance without depending on Moore's Law, people use parallel computing instead of sequential computing nowadays. What's more, products with parallel hardware and chip multi-processors has become one of the mainstream in the microprocessor industry [5].

In addition to the parallel computing, people also leave the optimization of computer architecture as the future of computing. Domain-specific architecture design and optimization is one of these topics and it is really hot these days because of the need for AI and other applications. A groundbreaking example for domain-specific architecture is Google's tensor processing unit (TPU), which was first deployed in 2015 and now serves more than 1 billion people. Compared with contemporary CPUs and GPUs using similar technologies, it runs deep neural networks (DNN) 15 to 30 times faster and has 30 to 80 times better energy efficiency [10].

The biggest challenge in domain-specific architecture design was raised from the domain of the embedded system. These days embedded system start to adapt algorithms which need very large computing power. For example, Tesla's automotive car needs to process millions of pixels in short latency and feed them to the AI algorithm [2]. Traditional AI computing platform like a CPU cannot provide so much computing power. Not only the computation limits the design, but energy efficiency is also vital for some system. A smart camera usually can be supplied limited1 energy. A camera on the drone, for example, must save energy to enlarge battery life. However, smart embedded systems do need complicated operations to implement neural networks, which usually costs lots of energy which is a dilemma. Finally, embedded systems are limited by cost. Generally, the engine controller under the hood of the

car and the chip in the mobile phone needs to be below $3-5 [5]. The chip in tennis shoes or greeting cards is even cheaper and cannot have a large chip area. Due to these reasons, some techniques like Single instruction, multiple threads (SIMT) may not be appropriate for such kind of system. Therefore, the need for an embedded system really calls for a novel design in computer architecture that can balance performance, energy efficiency, and cost for computation-intensive tasks.

There are some recent achievements in improving computer performance through optimizing of computer architecture. In a study applied in an application specific integrated circuit design (ASIC design), an high-accuracy approximate adder is introduced which can decrease the power and delay with a low relative error [8]. Subsequently, a study introduces approximate operators into DNN hardware accelerators to reduce latency and energy consumption [13]. Another study applies approximate divider and square root operations on their RISC-V design [12]. But all these studies narrow their designs in some specific fields and the approximate adders and approximate operators actually can be used in a wide range of computing components including CPU, GPU and SoC. In this project, we propose an out-of-order processor design based on RISC-V with approximate computing units to reach a general solution for SoC design that can balance performance, energy and cost.

2 Benchmark

In the field of computer architecture, there exist several different processor design to speed up computing from different perspectives.

2.1 MIPS Processor with Classical 5-stage Pipeline

One popular design is MIPS processor with classical 5-stage pipeline. This architecture, proposed by Standford University, uses MIPS Instruction Set Architecture (ISA) and five-stages pipeline, which enables the processor to execute more than one instruction in one clock cycle. However, all instructions in the processor are executed in-order with a single Arithmetic Logic Unit (ALU) [7]. Therefore, the average Instruction per Cycle (IPC) is less than five, which is not efficient enough.

2.2 Rocket Core

Another widely-used architecture is Rocket Core, which is designed by UC Berkeley. Compared to classical 5-stage MIPS processor, Rocket Core has a Floating Point Unit (FPU) and a co-processor, called "ROCC". The specialized FPU speeds up the floating-point computing. Meanwhile, the co-processor enables users to customize their instructions [1].

2.3 The Berkeley Out-of-Order Machine

For further improvement, UC Berkeley releases an architecture for out-of-order execution, called "BOOM". This structure has up to 10 pipeline stages. What's more, instructions are executed out-of-order in this processor [3]. Therefore, it is far more efficient than previous structures.

All architectures mentioned above do not have accelerator for machine learning or approximate computing. In comparison, our final product will have approximate computing unit in Execution stage to speed up machine learning. We believe that with these accelerators, our architecture will performance better in certain situations.

2.4 Hummingbird E203

Humming bird E203 is one of the most popular open-source RISC-V SoC among Chinese community and was designed by Zhenbo Hu's team in China [9]. It is designed for embedded systems with extremely low power consumption and circuit area, and is suitable for research purpose.

3 Customer Requirements & Engineering Specifications

3.1 Customer Requirements (CR)

Since our project is experimental and research-oriented, there do not exist specific customers for our design. However, customer requirements are still necessary, as after the completion of our project, potential embedded system customers may be interested. Meanwhile, mainstream CPU and SoC evaluation indices still apply to our design, e.g., core frequency, performance, energy efficiency, power consumption, etc. Briefly speaking, our target market is artificial intelligence hardware systems, and the consumers include users who want to run machine learning applications in their embedded systems.

We divide our customer requirements into three categories: general (G), performance (P), and embedded system (ES). For the general part, customers may require that the design should be compatible to general RISC-V applications, and attached with clear documentations. Target users of our design may include computer science researchers, embedded system users, or both. The former requires the performance part, e.g., good performance for machine learning applications, and the latter requires the embedded system part, e.g., low power consumption and quick response.

3.1.1 General Requirements

The main concern for the general requirements part is to set up proper constraints so that we can achieve good compatibility as well as usability. This is determined by the characteristics of our target market and consumers.

The target market of our SoC is a sub-division of the traditional computing hardware market, which requires tight constraint on performance, cost and energy efficiency. Thus, the general requirement should be a super-set of current computing market. For example, microprocessors and computing system based RISC-V ISA has solid ecosystem and active community. Tool chains like `gcc` compiler support, Linux kernel support and verification support are well established [14]. The market will welcome a new SoC which is fully compatible with current RISC-V ecosystem because the cost to adopt our product will be low and they will need fewer labors and less time, comparing to the product needs software part from scratch. Also, the competitive computing market requires good usability to win the business. The computing hardware market requires good documentations, as experienced engineers in this high standardized industry can be much more productive when the documentation follows professional protocols. This requirement can also be justified if we consider who is our target consumers. It is hard for startups or small size company to use our product because these companies generally cannot take the risk to adopt a new SoC. Our target consumer will be medium or large size, system level companies who want to use our SoC to increase their current product's efficiency. These companies have strong low-level research and development capability, and their software or application may be architectural specific. Therefore, they will prefer a new SoC with minimum architectural change and requires little effort to deploy their applications. Compatibility and usability are therefore the main points in the General requirements part.

As our SoC is based on the RISC-V architecture, being compatible with the RISC-V architecture is the most important constraint to consider. RISC-V is a family of ISA, so selecting a proper RISC-V ISA subset is critical. RISC-V ISA includes a base integer ISA, like RISC-V 32I, and optional extension to the base ISA. We want our SoC to be a stable target for RISC-V applications so that the application source code written in high level language like C/C++ can be correctly mapped to our SoC. To run general RISC-V applications, a RISC-V 32G is a reasonable target as suggested by the RISC-V specification [16], which includes:

1. RISC-V 32I base ISA
2. RISC-V M extension, for integer multiplication and division
3. RISC-V A extension, for atomic instructions
4. RISC-V F extension, for single-precision floating-point
5. RISC-V D extension, for double-precision floating point

RISC-V 32G applications encode their instructions in 32 bits. However, due to the limited memory size of many embedded systems, we hope that the size of applications can be as small as possible. One of the solutions is to introduce RISC-V C extension for compressed instructions, where instructions are encoded in 16 bits. If time permits, we should add the support for RISC-V C extension to further optimize for embedded systems.

To enhance the usability of our SoC, a clear and clean documentation is needed. Enough documentation should be written for each module, each subsystem and the overall design. If time permits, some performance analysis can also be added in the documentation so the consumers can better unleash the potential of our SoC.

Finally, due to the requirement of cost control in most embedded system designs, our SoC should be as inexpensive as possible.

General Requirements:

1. Is compatible with RISC-V applications (weight: 10)
2. Has good documentation for reference (weight: 6)
3. Is inexpensive (weight: 5)

3.1.2 Performance Requirements

While Gordon Moore predicted the growth in semiconductor, he also foresaw its end 50 years later. In recent days, even the company uses Moore's law as a guideline have to slow their pace to build even smaller transistor. Therefore, our SoC faces a market which require smart design to achieve higher performance under current technology node. The market will first need high performance for normal arithmetic operations. Although nowadays specific AI algorithms rely heavily on the matrix operations, the normal arithmetic operations still take great part in a program to do branching, jumping, indexing, etc. Thus, to achieve an overall high performance, we need to have high performance normal arithmetic units and designs so that this part will not be a bottleneck. For the normal operations, the performance throughput equals to frequency times the Instructions executed per clock cycle (IPC). In order to achieve a high throughput, we need both a high frequency and a high IPC. Designing more pipeline stages may help achieve higher clock frequency, but may result in higher plenty when there is a miss prediction. Dynamic scheduling is a technique to increase the throughput by enabling Out-of-Order execution when instructions have no dependency. This method extract instruction level parallelism and it is always combined with other technique like renaming, speculative execution to maximize the outcome. Also, enough number of function unit is needed so that the pipeline will not stall during the congestion at the execution stage. Our SoC should adopt these traditional techniques to have a high performance on the traditional normal arithmetic operations.

The AI application is a hot topic today. The performance for running AI application is also demanded by our target market. One of the most significant properties of this application is that it is error torrent. During the inference, minor error will not introduce much difference on the final result. During the training, some error may even manually be introduced to increase the robustness of the algorithm. This makes approximate computing a potential way to achieve higher frequency by using circuits with fewer elements.

Another property of AI application is that it is data driven, which means it is usually memory-bounded. Our SoC needs a high performance memory subsystem to make sure the pipeline for AI application will not hungry for data and halt. By using memory hierarchy and introducing cache, we can exploit the temporary locality and spacial locality of memory access to increase the memory subsystem's performance. One of the most important parameter that determines the cache's performance is its size. A trade off between the cache size and other parameter like frequency needs to be made so that the overall performance can be improved.

Before our design is fixed, we want do some architecture exploration. Therefore, we want the parameters of some of the critical modules, like the reorder buffer size and cache size is configurable so that we can change them to evaluate performance.

Performance Requirements:

1. Runs fast for normal arithmetic operations (weight: 8)
2. Runs fast for machine learning applications (weight: 9)
3. Runs fast for memory-bound applications (weight: 7)
4. Can be configured with different parameters (weight: 2)

3.1.3 Embedded System Requirements

Although the ware-house scale computing center still take plenty of the computation nowadays, the market has a trend to offload part of the computation to the edge because of security issue, latency issue, etc. Our SoC will be more competitive at the edge as the concept AI on things has attracted lots of attention. Therefore, we require our SoC satisfy some requirement for embedded system.

The embedded system is often energy-bounded as the power source is usually a battery. High power consumption is not allow because these scenario often require long battery life. The can be evaluated using the meric Operations processed within unit energy, so our SoC should have a limited number at this part. Some embedded system also have requirement on latency. For AR/VR application, this requirement is necessary as longer latency will cause discomfort. We hope our SoC can provide a low average response time to a request for service. The market also requires embedded SoC have low cost, as the embedded system application is normally cost sensitivity. This can be evaluated by the usage of resources on the FPGA, like LUT, BRAM and DSP. The embedded system also has to need to connect to variaous types of peripherals, so have more I/O controllers can make our product more competitive on the market.

Embedded Requirements:

1. Saves power (weight: 6)
2. Responds quickly (weight: 4)

3. Low cost (weight: 3)

4. Has good support for multiple I/O devices (weight: 3)

3.1.4 Benchmark Competitions against CR

In this part, we quickly review some existing open-source solutions against customer requirements. This step gives us a better view of customer requirements as well as offers us some clue on how to design our own products based on existing solutions.

The evaluation is based on a 5-point scale, where 5 means "satisfies perfectly" and 1 means "doesn't satisfy at all". We select four open-source solutions to evaluate: MIPS processor with classical 5-stage pipeline, Rocket Core, Berkeley Out-of-Order Machine (BOOM), and Hummingbird E203. For instance, in terms of "is compatible to general RISC-V applications", we find that Rocket Core and BOOM have the best support, so we give them 5 points. Hummingbird E203 has limited support for RISC-V applications, so we give it 3 points. The MIPS processor with classical 5-stage pipeline is not compatible with RISC-V applications, so we give it 1 point.

The detailed benchmark competitions against CR are shown in the HOQ chart (Fig. 4).

3.2 Engineering Specifications (ES)

Based on the previous customer requirements, we come up with our engineering specifications of our project, which represent for the quantifiable measures to guide us to design the project properly that can meet customers' requirements. Table 1 shows our overall engineering specifications in our project design.

Engineering Specification	**Unit**	**Target Value**
Support RV32G instruction set architecture (ISA)	-	Yes
Core frequency on FPGA test platform	MHz	100
Number of pipeline stages	-	9
Instructions executed per clock cycle (IPC)	-	0.5
Support instruction dynamic scheduling	-	Yes
Typical total cache size	KB	32
Number of function units	-	6
Average response time to a request for service	ms	10
Usage of look-up tables (LUT) on FPGA	k	120
Usage of block RAM (BRAM) on FPGA	-	50
Usage of digital signal processor (DSP) on FPGA	-	30
Power consumption on target FPGA test platform	W	5
Operations processed within unit energy.	MOp/J	25
Number of flexibly-configured modules	-	10
Number of I/O device types	-	3
User guide and programmers manual	-	Yes

Table 1: Engineering Specifications.

This should be a key step to "translate" customers' words into professional terms and specifications. For instance, when customers require that their RISC-V applications can run on our SoC, what we actually need is that our SoC should support RV32G ISA. In this part, we will explain how we check the cross correlation of ES, examine the benchmark competitions against ES, set our own target for ES, and most importantly, correlate CR to ES.

3.2.1 Cross Correlation of ES

In this step, we check the cross correlation of ES, and reveal some internal connection between each specification we just made in the previous step. For example, when we increase the number of function units, the usage of hardware resources also increases. When we want to support instruction dynamic scheduling to support out-of-order instruction execution, we can greatly improve the number of instructions executed per clock cycle, which is a significant mark of performance improvement.

This step also gives us a clear guidance on design trade-offs. The most evident contradiction is that the SoC we want to design can never achieve low cost, high performance, low power consumption at the same time, which exactly corresponds to our three different parts of customer requirements. For example, if we want to improve the performance, possible solutions include increasing the frequency, increasing the number of function units, etc., none of which is good for saving power and reducing cost.

The detailed cross correlation of ES is given in the HOQ chart (Fig. 4).

3.2.2 Benchmark Competitions against ES & Set Target for ES

It is important to perform some benchmark on some existing solutions in terms of our engineering specifications, which enables us to understand better about drawbacks of existing solutions as well as our design targets in the future.

This process sometimes requires "reverse engineering" for commercial products, but should be much easier for open-source products. In our survey, except that Hummingbird E203 is partially open-source, all the other three processors are completely open-source, and we should be able to access the data conveniently. However, we need to pay attention that some data is still difficult to access, e.g., the average response time to a request for service. There are some reasons for this kind of lack of data. First, even though the processor design is open-source, the synthesis tool or the test suites are not open-source, leading to the difficulty to reproduce the test results. Second, even though the test processes are completely public, the results may depend on the specific hardware testing platform, which may be expensive or inaccessible. Third, some of the designs can be flexibly configured, and there is no standard or typical settings, as a result of which the test results under different settings can not be referred or compared directly.

In this step, we try our best to explore the data of these existing solutions. Some data can be easily

accessed, e.g., the number of pipeline stages or whether instruction dynamic scheduling is supported, while some other data cannot. [4] provides us with many useful indicators to evaluate a design and meanwhile gives us lots of data of both Rocket Core and BOOM, including usage of hardware resources, etc.

Based on the data of existing solutions, we come up with our target values for ES. For example, as our aim is to provide good experience for machine learning application users, performance is crucial to some applications like neutral network, and hence we decide to support instruction dynamic scheduling, and set a relatively high value of core frequency of our design. During the process of deciding our own targets, we mainly refer to the data of BOOM, but also look at the data of other three solutions.

The detailed benchmark competitions against ES and our targets for ES are shown in the HOQ chart (Fig. 4).

3.2.3 Correlation from CR to ES

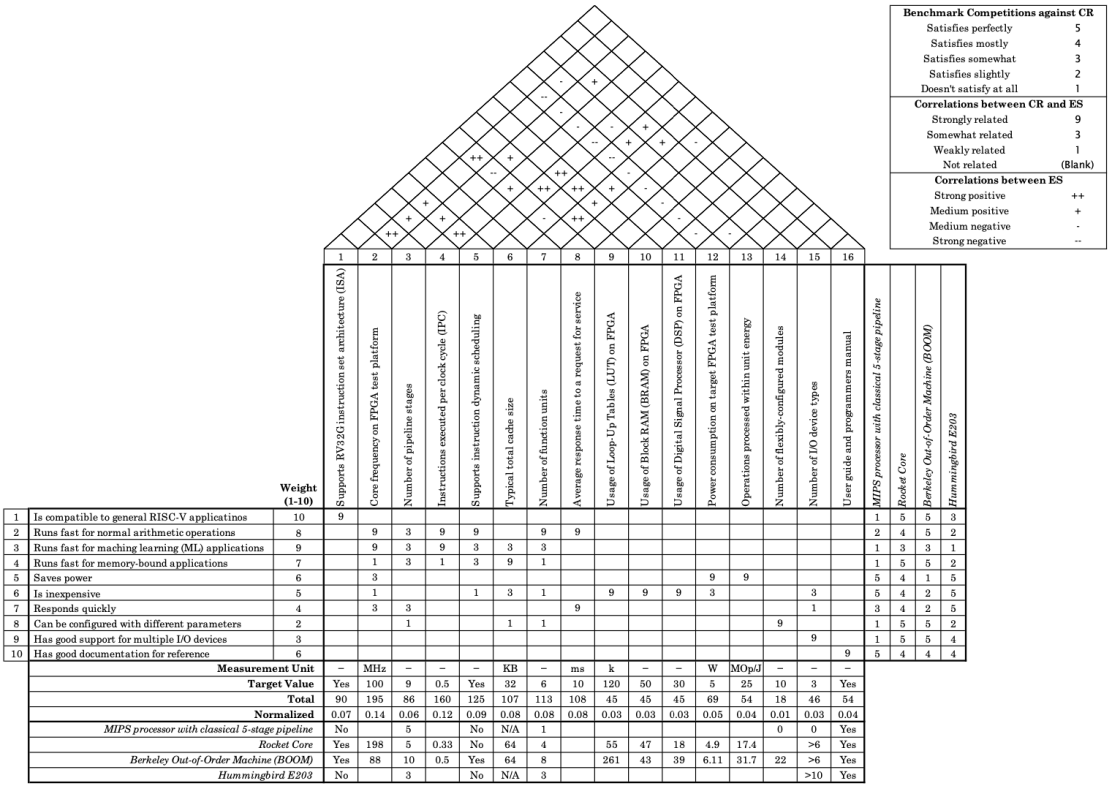
In this step, we not only make our engineering specifications from scratch, but also check whether all the customer requirements can be covered in our specifications. While the following paragraphs give a short explanation on how we transform CR into ES.

1. **General part**: As customers require that our SoC should be compatible to general RISC-V applications, we require that our SoC should support RV32G ISA. As customers need good and clear documentation, we require that we need to deliver SoC user guide and programmer manual. To control the cost, we need to control our circuit size by limiting our usage of hardware resources, e.g., usage of look-up tables (LUT), digital signal processors (DSP), and block RAMs (BRAM).
2. **Performance part**: We want to optimize our SoC for three different customer scenarios: normal arithmetic operations, machine learning applications, and memory-bound applications. They share some common characteristics, e.g., they all require that our SoC should support instruction dynamic scheduling and as high core frequency as possible. Meanwhile, they differ in some specific aspects. For instance, as normal arithmetic applications and machine learning applications are mostly compute-bound, they will benefit more from the improvement on core frequency, instructions executed per clock cycle, and number of function units, while memory-bound applications will benefit more from the improvement on total cache size, as cache is used to accelerate the access to memory.
3. **Embedded system part**: Customers require that our SoC works well in embedded systems. To save power, we need to monitor and control the power consumption and energy efficiency of our SoC. To achieve quick response, we need to measure the average response time to a request for service. To configure our SoC with different parameters easily, we need to increase the number of flexibly-configured modules in our digital design. To have good support for multiple I/O devices,

we need to support multiple kinds of I/O types and test on various I/O devices.

3.3 Quality Function Development (QFD)

We develop our House of Quality (HOQ) chart with QFD method. The HOQ chart is shown in Fig. 4.



Benchmark Competitions against CR	
Satisfies perfectly	5
Satisfies mostly	4
Satisfies somewhat	3
Satisfies slightly	2
Doesn't satisfy at all	1
Correlations between CR and ES	
Strongly related	9
Somewhat related	3
Weakly related	1
Not related	(Blank)
Correlations between ES	
Strong positive	++
Medium positive	+
Medium negative	-
Strong negative	--

		Weight (1-10)	1 Supports RV32G instruction set architecture (ISA)	2 Core frequency on FPGA test platform	3 Number of pipeline stages	4 Instructions executed per clock cycle (IPC)	5 Supports instruction dynamic scheduling	6 Typical total cache size	7 Number of function units	8 Average response time to a request for service	9 Usage of Loop-Up Tables (LUT) on FPGA	10 Usage of Block RAM (BRAM) on FPGA	11 Usage of Digital Signal Processor (DSP) on FPGA	12 Power consumption on target FPGA test platform	13 Operations processed within unit energy	14 Number of flexibly-configured modules	15 Number of I/O device types	16 User guide and programmers manual	*MIPS processor with classical 5-stage pipeline*	*Rocket Core*	*Berkeley Out-of-Order Machine (BOOM)*	*Hummingbird E203*
1	Is compatible to general RISC-V applicatinos	10	9																1	5	5	3
2	Runs fast for normal arithmetic operations	8		9	3	9	9		9	9									2	4	5	2
3	Runs fast for maching learning (ML) applications	9		9	3	9	3	3	3										1	3	3	1
4	Runs fast for memory-bound applications	7		1	3	1	3	9	1										1	5	5	2
5	Saves power	6		3										9	9				5	4	1	5
6	Is inexpensive	5		1			1	3	1		9	9	9	3			3		5	4	2	5
7	Responds quickly	4		3	3					9							1		3	4	2	5
8	Can be configured with different parameters	2			1			1	1							9			1	5	5	2
9	Has good support for multiple I/O devices	3															9		1	5	5	4
10	Has good documentation for reference	6																9	5	4	4	4
	Measurement Unit		–	MHz	–	–	–	KB	–	ms	k	–	–	W	MOp/J	–	–	–				
	Target Value		Yes	100	9	0.5	Yes	32	6	10	120	50	30	5	25	10	3	Yes				
	Total		90	195	86	160	125	107	113	108	45	45	45	69	54	18	46	54				
	Normalized		0.07	0.14	0.06	0.12	0.09	0.08	0.08	0.08	0.03	0.03	0.03	0.05	0.04	0.01	0.03	0.04				
	MIPS processor with classical 5-stage pipeline		No		5		No	N/A	1							0	0	Yes				
	Rocket Core		Yes	198	5	0.33	No	64	4		55	47	18	4.9	17.4		>6	Yes				
	Berkeley Out-of-Order Machine (BOOM)		Yes	88	10	0.5	Yes	64	8		261	43	39	6.11	31.7	22	>6	Yes				
	Hummingbird E203		No		3		No	N/A	3								>10	Yes				

Figure 4: House of Quality (HOQ) chart.

The main steps of QFD method has been expanded in the previous sections. Finally we are able to examine our design priority according to "Total" and "Normalized" fields. The top 5 engineering specifications we need to consider are core frequency, instructions executed per clock cycle, support for instruction dynamic scheduling, number of function units, and average response time. The result generally matches our expectation that performance is still among the most important factors in embedded system SoC design. Meanwhile, some specifications like number of flexibly-configured modules are relatively not as important as previous ones.

4 Project Plan

Based on the project specification, the overall workload can be divided into three milestones. We arrange the timeline of each milestones according to the deadline of three design reviews and final prototype review. Meanwhile, we classify our work into four categories: **memory related units, computation units, pipeline components** and **testing**. **Memory related units** range from the memory block on FPGA board to the cache design of the processor. **Computation units** covers Integer Arithmetic Logic Unit(ALU), Multiplier, Divider, Floating Point Unit (FPU) and Approximate Computing Units. **Pipeline components** includes every core components like Mapping Table, Re-order Buffer and Issue Queue. **Testing** is basically to verify the correctness of every module we write.

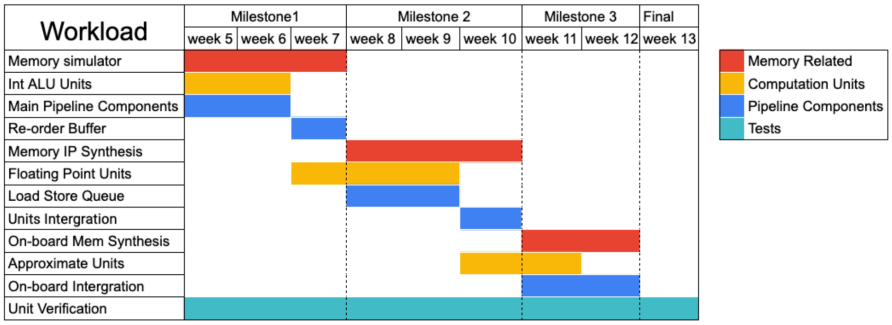


Figure 5: Project Gantt Chart

Fig. 5 shows a simplified Gantt chart of the project. The testing part should be conducted throughout the developing process. The rest three workloads will be distributed evenly into every milestone to make sure we can always balance our pace.

From the chart we can observe that Week 2-5 is not included because what we have mainly done is preparation. We familiarize ourselves with the memory blocks on FPGA board and start to write main components in RISC-V out-of-order (O3) pipeline.

1. By Milestone 1 (week 5 - week 7), we will build a naive simulation-correct and synthesizable RISC-V O3 core.
2. By Milestone 1.5 (week 7 - week 8), we will finish building a complete simulation-correct and synthesis-correct RISC-V O3 core, with a complete memory workflow.
3. By Milestone 2 (week 8 - week 10), we will complete extra components, e.g., approximate units, I/O devices of FPGA.
4. By Milestone 3 (week 10 - week 12), we will run various research experiments on the architecture to test out the performance, power and area balance under different scenarios.

We assign different workloads to different teammates based on their respective expertise. Jian Shi and Yichao Yuan will be in charge of memory related units. Yiqiu Sun will be responsible for computation unit. Li Shi and Zhiyuan Liu will be responsible for pipeline related units.

The only hardware we need is an FPGA board, which is provided by one of our teammates Jian Shi. Besides that, every workload can be done on software. In conclusion, the overall project is expected to be finished without any funding.

5 Conclusion

As Moore's law is reaching its end, the market calls for specialized SoC with domain specific architecture. However, current vendors failed to provide such kinds of product. In the domain of embedded system, such phenomenon is even more prominent. Although general purpose processors like Rocket Cores and BOOM provides efficient implementation for general applications under RISC-V architecture, they do not provide optimized units and designs for applications like AI. On the other hand, approximate computing may provide desirable speedup for them. Based on this fact, we analyze the customer requirements, which can be categorized into three big parts: General Requirements, Performance Requirements and Embedded System Requirements. As a comparison, processors on the market like the Rocket core and BOOM cannot fully satisfy these requirements. These requirements are then detailed and a QFD was given in this report. We outline the project plan with gantt chart. Our project plan includes four milestones to make sure our development proceeds steadily. Work is divided reasonable among all teammates so that our team can be more productive.

References

[1] Krste Asanović et al. The rocket chip generator. Technical report, 2016.

[2] Yarrow Bouchard. How tesla is using artificial intelligence and big data, 2019.

[3] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolic, David A. Patterson, and Krste Asanović. Boom v2: An open-source out-of-order risc-v core. Technical report, 2017.

[4] Alexander Dörflinger et al. A comparative survey of open-source application-class risc-v processor implementations. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, CF '21, page 12–20, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Samuel H. Fuller and Lynette I. Millett. *The Future of Computing Performance: Game Over or Next Level?* National Academy Press, 2011.

[6] Nor Zaidi Haron and Said Hamdioui. Why is cmos scaling coming to an end?, 2008.

[7] John L. Hennessy and David A. Patterson. *Computer Organization and Design MIPS Edition: The Hardware/Software Interface (Sixth Edition)*. Elsevier Science, 2020.

[8] Junjun Hu, Zhijing Li, Meng Yang, Zixin Huang, and Weikang Qian. A high-accuracy approximate adder with correct sign calculation. *Integration*, 65:370–388, 2019.

[9] Zhenbo Hu. *Teach You How to Design a CPU Hand by Hand: RISC-V Processors*. People's Posts and Telecommunications Press, 2018.

[10] Norman P. Jouppi, Cliff Young, Nishant Patil, and David Patterson. A domain-specific architecture for deep neural networks. *Commun. ACM*, 61(9):50–59, Aug 2018.

[11] Kamran Latif. Parallelism via concurrency at multiple levels, 2014.

[12] Lei Li, Michael Gautschi, and Luca Benini. Approximate div and sqrt instructions for the risc-v isa: An efficiency vs. accuracy analysis. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–8, 2017.

[13] Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. Alwann: Automatic layer-wise approximation of deep neural network accelerators without retraining. *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2019.

[14] RISC-V International. Risc-v exchange: Available software, 2021.

[15] Ashlee Vance. Intel says adios to tejas and jayhawk chips, 2004.

[16] Andrew Waterman and Krste Asanović. *The RISC-V Instruction Set Manual Volume I: User-Level ISA, Document Version 20191213*. 2019.

Bios

Zhiyuan Liu (Figure 6)

My name is Zhiyuan Liu, a senior student in JI major in ECE. I like drawing, photography, playing video games, and basketball. Besides, I like to assemble computers by myself. Before the capstone design project, I studied VE270 Introduction to Logic Design and VE370 Intro to Computer Organization in JI, and I also learned VE470 Computer Architecture by myself. I am very interested in computer architecture. I once worked as an intern at Microsoft and I participated in feature development for products there.

This fall, I will become a regular employee of Microsoft.

Figure 6: Zhiyuan Liu

Jian Shi (Figure 7)

I am Jian Shi, a senior student major in ECE at JI. I like taking photos, analyzing video games and swimming. Also, I am a full stack developer. I build and maintain an online server by myself. Starting from choosing hardware components in the server, to the deamon service written by myself, I learn a lot from this experience. Currently, the server has become my computing center. Whenever I have a task that can be processed or maintained by computer or network, I will write a program and throw it to my server. By the way, the test platform for this project is also based on my server.

This fall, I will continue my Ph.D degree at JI under Prof. Weikang Qian's guidance. I think this project will help a lot in my future Ph.D study.

Figure 7: Jian Shi

Li Shi (Figure 8)

I am Li Shi, a senior undergraduate major in ECE at JI. I like studying photography, watching sci-fi movies, eating Chinese food (especially Sichuan cuisine), and sleeping. Also, I enjoy playing with kittens and puppies.

Figure 8: Li Shi

Before the capstone design project, I have been working on the High-Level Synthesis project instructed by Prof. Weikang Qian at JI for about 2 years, during which I accumulated some experience on digital design, software engineering, compiler design, etc. Meanwhile, I once worked as an intern at Apple, where I participated in developing a CPU simulator and studied computer architecture by myself. My interests include digital design, embedded systems, computer architecture, etc. In 2022, I will start my journey in M.S. in ECE program at Carnegie Mellon University.

Yiqiu Sun (Figure 9)

My name is Yiqiu Sun. I major in ECE at JI and CE at University of Michigan. Interested in Computer Architecture and new computing techniques like approximate computing, I worked with Prof. John Hayes on stochastic computing and with Prof. Trevor Mudge on reconfigurable accelerators. Meanwhile, my course experience of EECS 470 Computer Architecture at Michigan would help me better contribute to this project. This fall, I will pursue my Ph.D. degree in computer science in University of Illinois at Urbana-Champaign. With a research focus on In-Memory Computing, I would like to figure out a better communication scheme between processor and memory in the future.

Figure 9: Yiqiu Sun

Besides academics, I am interested in cooking and hiking. I also adopted two cats. They are Gomez and Kent. These two fluffy creatures are my best life companions.

Yichao Yuan (Figure 10)

My name is Yichao Yuan. I am a senior student in JI major in ECE. I like movies, graphics and various technology topics. I enjoy writing software and building hardware and the beauty of their interaction always amazes me. Before the capstone project, I have solid knowledge at each abstraction layer of hardware. In JI, VE270 and VE370 introduce basics about logic design and computer organization. In 2020 spring, I took CS152 and EECS151 in UC, Berkeley, which gave me advanced knowledge in computing architecture and digital design.

This fall, I will become a ECE M.S. student at University of Michigan, Ann Arbor.

Figure 10: Yichao Yuan