DEPARTMENT OF
Electrical and Computer Engineering

11-203 Donadeo Innovation Centre for Engineering
9211-116 Street NW
University of Alberta
Edmonton, Alberta
Canada T6G 1H9

Preliminary Design: FPGA-Based RISC-V CPU and Fysh Programming Language and Compiler

Revisions

Revision	Author	Changes	Date
001	Charles Ancheta, Yahya Al-Shamali, Kyle Prince	Initial Release	2024-02-16

DEPARTMENT OF
Electrical and Computer Engineering

11-203 Donadeo Innovation Centre for Engineering
9211-116 Street NW
University of Alberta
Edmonton, Alberta
Canada T6G 1H9

Table of Contents

Acronyms

Acronym	Full Description
ALU	Arithmetic Logic Unit
AST	Abstract Syntax Tree
CPU	Central Processing Unit
ECE	Electrical and Computer Engineering
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GPIO	General Purpose Input/Output
IR	Intermediate Representation
ISA	Instruction Set Architecture
LLVM	Low Level Virtual Machine
MCU	Microcontroller Unit
PC	Program Counter
PL	Programmable Logic, the FPGA part of the Zybo development Board
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
ROM	Read-only Memory
RV32I	RISC-V 32-bit Integer, the base 32-bit RISC-V instruction set

References

[1] S. Knudsen, "RISC-V FPGA-based CPU and Language." Dec. 2023. Available: `https://fysh-fyve.github.io/ECE492_RISCV_PP.pdf`

[2] "RISC-V International – RISC-V: The Open Standard RISC Instruction Set Architecture." Feb. 2024. Available: `https://riscv.org/`

[3] C. Ancheta, Y. Al-Shamali, and K. Prince, "Proposal Response: RISC-V FPGA-based CPU and Language." Jan. 2024. Available: `https://fysh-fyve.github.io/ECE492_RISCV_PPR_V01.pdf`

[4] "Esoteric programming language - Esolang." Feb. 2024. Available: `https://esolangs.org/wiki/Esoteric_programming_language`

[5] "Introducing ESP32-C3 | Espressif Systems." Nov. 2020. Available: `https://www.espressif.com/en/news/ESP32_C3`

1 Purpose

This document describes the preliminary design for the RISC-V FPGA-based CPU and Language [1].

2 Concept of Operation

The high-level operation of the RISC-V FPGA-based CPU and Language is illustrated by the following user stories and use cases.

2.1 User Stories

2.1.1 Writing a Compiler

Upon receiving the project list for this semester's capstone project, we were excited to see that there is a project described as "Likely the most difficult project on our list": a native C compiler for a single board computer with a custom CPU and instruction set. It was to our disappointment that the computer did not yet have an operating system, rendering the project infeasible and impractical.
However, we insisted on working on a compiler. This time, we are targeting RISC-V[2], an open standard ISA that is gaining popularity in the industry. It was initially stated in the proposal that we are taking inspiration from modern languages like Rust, Zig, or Go[3], but we have also decided to unleash our creativity and make our programming language esoteric[4].

2.1.2 Learning about RISC-V ISA

We are also curious about the internals of real-world CPUs. We have only encountered a "toy" VHDL implementation of a CPU in ECE 410, our digital logic design course. Our initial idea was to run the compiled program written in our esoteric programming language on an ESP-32 C3 MCU[5]. However, we are required to have a hardware aspect to our project.
Because of this, we decided to implement a RISC-V CPU ourselves with a custom CPU instruction to showcase the extensibility of RISC-V. Implementing the CPU in VHDL and programming a Zybo board means that the project can be used by all ECE students. With a softcore microprocessor and a compiler, we have a real full-stack computer engineering project that we can be proud of and can potentially aid in the learning of other computer engineering students.

2.2 Use Cases

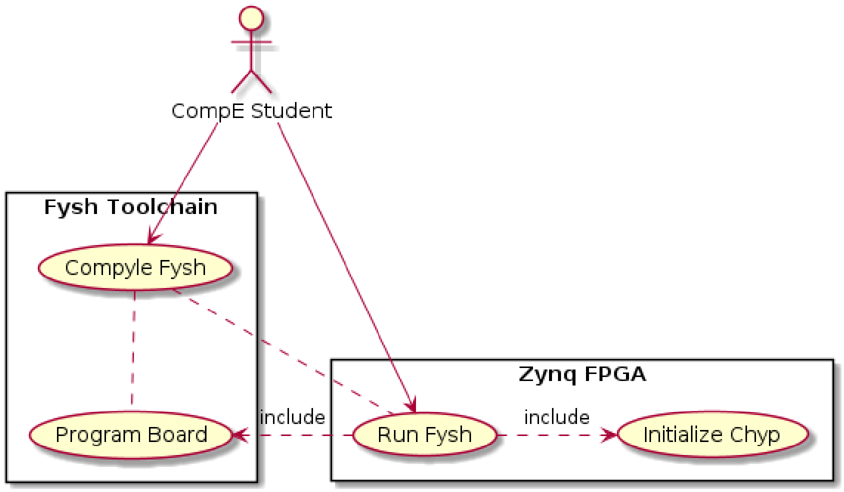

Figure 1: Use cases for the Fysh Toolchain

Use Case Description and Details	
Number	UC-001
Name (action)	Initialize Chyp
System	Zynq FPGA
Actor	CompE Student
Use Case Goal	Initialize Zynq FPGA with the Fysh-Fyve CPU.
Primary Actor	CompE Student
Preconditions	CompE Student has a development board.
Postconditions	FPGA is programmed with the RISC-V CPU.
Basic Flow	1. CompE student initializes the Fysh-Fyve project in Vivado. 2. CompE student runs the hardware synthesis and implementation. 3. CompE student generates a bitstream and programs the FPGA.
Alternate Flows	None

Table 1: UC-001 - Initialize Chyp

Use Case Description and Details	
Number	UC-002
Name (action)	Compyle Fysh
System	Fysh Toolchain
Actor	CompE Student
Use Case Goal	Compile Fysh source code into an executable format.
Primary Actor	CompE Student
Preconditions	Fysh source file exists.
Postconditions	RV32I machine code describing the Fysh source code is outputted.
Basic Flow	1. CompE student runs FyshSea, the Fysh Compyler, giving the Fysh source code as input. 2. FyshSea program exits and a binary file with RV32I format is outputted.
Alternate Flows	A. Fysh source code has a syntax or type error. 1. FyshSea outputs an error message describing the error. 2. FyshSea exits.

Table 2: UC-002 - Compyle Fysh

Use Case Description and Details	
Number	UC-003
Name (action)	Program Board
System	Fysh Toolchain
Actor	CompE Student
Use Case Goal	Download Fysh fyrmware into the Fysh-Fyve chyp.
Primary Actor	CompE Student
Preconditions	1. Fysh program is successfully compiled into RV32I machine code. 2. Zynq FPGA is initialized with the Fysh-Fyve CPU.
Postconditions	The Zynq FPGA has the firmware loaded into the Fysh-Fyve CPU.
Basic Flow	1. CompE student transforms the firmware binary into a format that is usable by the Fysh-Fyve CPU. 2. CompE student downloads the FPGA-usable format into the FPGA.
Alternate Flows	A. Firmware size is too large. 1. FyshDude outputs an error message describing the error. 1. FyshDude exits.

Table 3: UC-003 - Program Board

Use Case Description and Details	
Number	UC-004
Name (action)	Run Fysh
System	Zynq FPGA
Actor	CompE Student
Use Case Goal	Execute program written in Fysh.
Primary Actor	CompE Student
Preconditions	1. FPGA is initialized with Fysh-Fyve CPU. 2. Fysh program is compiled into a binary. 3. Fysh program is programmed into the development board.
Postconditions	Fysh program is running on a device.
Basic Flow	1. CompE student depresses the reset button on the development board. 2. CPU runs the instructions one by one.
Alternate Flows	A. CPU encounters an exception. 1. CPU halts execution. 2. Error LED lights up.

Table 4: UC-004 - Run Fysh

3 Functional and Performance Requirements

In the table below, the "CPU" refers to the Fysh-Fyve CPU, the FPGA-Based RISC-V CPU for the Zybo development board. "Fysh" refers to the Fysh programming language, the custom-made esoteric programming language for this project. "FyshSea" refers to the Fysh Compyler, the compiler that targets RV32I with native support for the custom RISC-V instruction.

FR #	Functional Requirement Description
FR-01	The CPU shall be compliant with the RV32I instruction set
FR-02	The CPU shall have general-purpose input/output pins
FR-03	The CPU shall have access to memory
FR-04	The CPU shall have a custom instruction to generate a random 32-bit integer
FR-05	The CPU's memory shall be programmable with RV32I firmware
FR-06	Fysh shall support basic integer operations
FR-07	Fysh shall support bit manipulation
FR-08	Fysh shall support memory addressing
FR-09	Fysh shall have a programming construct for random number generation
FR-10	Fysh shall be statically typed
FR-11	FyshSea shall convert Fysh source code to RV32I instructions
FR-12	FyshSea shall support the basic features of Fysh

Table 5: Functional Requirements

PR #	Performance Requirement Description	Related FRs
PR-01	The CPU shall perform 1 instruction every 2 clock cycles on average	01
PR-02	The CPU's hardware RNG shall periodically generate a unique number	04

Table 6: Performance Requirements

4 System Design

The overall system will have two parts as shown in Figure 2, the Fysh toolchain in the development machine, and the RISC-V CPU in the Programmable Logic (PL) of the Zybo board. The development machine is usually not part of the "deployment", but the compiler is part of the project and should be shown in this case.

4.1 System Architecture

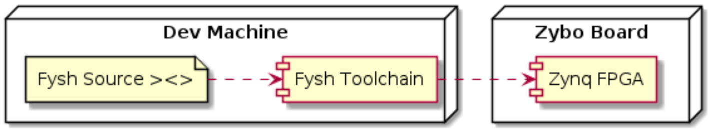

Figure 2: Deployment diagram for Fysh firmware.

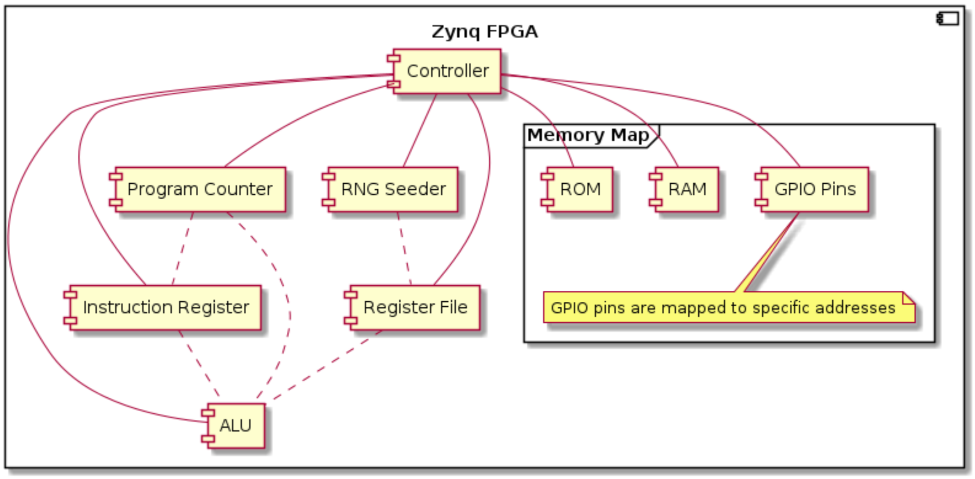

Figure 3: Simplified Fysh-Fyve Architecture.

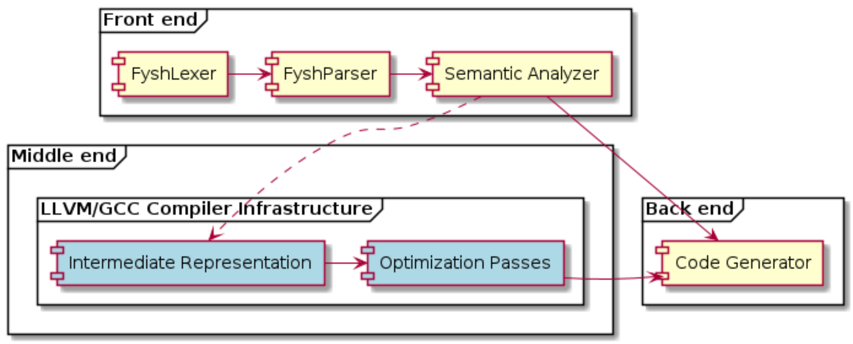

Figure 4: FyshSea Architecture. Components in light blue are optional.

4.1.1 Hardware Components

Component	Description
Controller	The brain of the brain of the computer (CPU).
Instruction Register	Contains the current instruction.
Program Counter	Contains the address of the current instruction.
Register File	Fast intermediate storage. Used for holding both data and addresses.
ALU	Performs computations on various data, mostly from registers.
GPIO Pins	Hardware Pins for off-chip interaction.
RAM	Main memory for the computer.
RNG Seeder	Hardware component for generating random numbers.

Table 7: Hardware Components

4.1.2 Software Components

Component	Description
FyshLexer	Transforms source code into a list of tokens.
FyshParser	Transforms a list of tokens into an AST.
Semantic Analyzer	Checks for correctness of the program by ensuring valid types, identifiers, operations, etc.
Intermediate Representation	Language-agnostic representation of the program that is subject to multiple optimization passes.
Optimization Passes	Performs various optimizations on the IR such as combining instructions, dead-code elimination, control flow analysis, etc.
Code Generation	Compiles the AST or IR to machine code.

Table 8: Software Components

5 System Requirements

SR #	System Requirement Description	FR #	PR #	Notes
SR-01	The Zybo development board shall have 128KB of RAM and 128KB or ROM	03, 05	03	Zybo Z7-010 has 270KB of Block RAM
SR-02	The Fysh toolchain must be supported on Linux			May also support Windows and macOS
SR-03	The Fysh toolchain and CPU must be open source			

Table 9: System Requirements

6 Minimum Design

In this section a minimum design, the “walking skeleton” is described. The purpose is to define the functionality to be implemented in the first development iteration. The outcome is reported to the client providing an opportunity for early feedback.

6.0.1 Fysh-Fyve (RISC-V Processor)

For the minimum design, our first iteration will focus on executing a single instruction on the FPGA CPU. The purpose is to demonstrate that the processor can be synthesized.

6.0.2 Fysh (Esoteric Language Compiler)

As for the compiler, our first iteration would be to compile a single instruction and upload it to the board. The purpose is to demonstrate that the compiler can generate RISC-V assembly code and that the code can be executed on the Fysh-Fyve processor.

6.0.3 Stretch Project (Digital Aquarium)

As a stretch goal, we would want to implement a digital aquarium using RGB LED matrix. This would showcase that the compiler can handle basic arithmetic, bit manipulation, memory addressing and loops. For the hardware, it will showcase Hardware RNG and GPIO.

7 High-Level Hardware Design

The hardware design as described in Section 4.1.1 follows a typical computer architecture with registers, program counter (PC), arithmetic logic unit (ALU), controller, and memory.
The **controller** is a finite state machine that drives all the other components of the CPU. The controller decides which components are selected on the datapath by decoding the instructions from the **instruction register**. The instruction register holds the instruction stored in the address that is held by the **PC**.

The **ALU** performs all the computations using values from various sources. The **register file** is the most common data source and destination of operations but the PC and immediate values from the instruction can also be used as a source.

The **RNG seeder** is a hardware component that uses hardware noise to generate a 32-bit number for random number generation. This makes it more difficult to predict the randomly-generated values unlike using other physical properties like the current time or the MAC address of a machine.

The memory address space of the CPU is comprised of the **ROM**, the **RAM**, and the **GPIO pins**. The ROM contains the firmware that is pre-programmed into the CPU before boot time. The RAM is volatile memory that can be used for temporary data storage. The GPIO pins are mapped to specific addresses and are read from and written to using load and store instructions, respectively.

8 High-Level Software/Firmware Design

The most abstract explanation of the software aspect of the project would be that a Fysh program is compiled by the custom Fysh compiler and executed on our hardware. More in depth, the compiler consists of multiple components which break down the Fysh program so that it can be understood and executed using C++.

The first piece of this compiler is the front end, which is made up of the lexer (**FyshLexer**), the parser (**FyshParser**), and the **semantic analyzer**. The lexer takes in Fysh code as input, and turns it into a series of tokens, i.e tokenizes, which the parser can then interpret. The parser takes a sequence of tokens from the lexer and produces an AST of the program while checking its syntax. Next the semantic analyzer traverses the syntax tree generated by the parser and checks if the program is semantically consistent.

From here we move on to the middle end of the compiler, which includes the **intermediate representation** and **optimization passes**, which are considered optional for our implementation. An intermediate representation is essentially a data structure representing source code, which may then be subjected to several optimization passes, some for analyzing, some for transforming, and other additional utility optimizations. Before moving on to the back end, we note once again that since the middle end is listed as optional, the compiler may move straight from semantic analysis to the back end.

The back end of the compiler contains the **code generation** aspect of our compiler, which processes source code into machine code so that it can be executed by the user.

9 Prototype Budget

Component	Mfr P/N	Mfr	Qty	Unit Price	Extended Price
FPGA Development Board	Zybo Z-7010	Xilinx/AMD	1	$299.00	$299.00
64x64 RGB LED Matrix	RGB-Matrix-P2.5-64x64	Waveshare	2	$36.10	$72.20
128x32 Monochromatic OLED Display	410-222	Digilent	1	$14.99	$14.99
				Total Cost	$386.19

Table 10: System Budget

Appendix: The Fysh Programming Language

This is fysh

```
><>
```

Fysh have to be terminated ~

```
><> ~
```

This is Steven

```
><steven> ~
```

Steven has binary scales. `}` represents 1 and `)` represents 0. Steve is valued at `b101` (5 in decimal)

```
><steven> = ><})}> ~
```

Steve doesn't give a flying fysh about their scale direction. ≈ for variable assignment is cool with them too

```
><steven>   ><}({> ~
```

Steven is blind. You have the power to bless them with sight, but it's completely optional

```
><steven>   ><{({°> ~
><steven> = ><{({o> ~
```

When there is a school of fysh, their collective value is equal to the sum of each individual member. This gives steven a value of `b101 = b100+ b001`

```
><steven>   ><{((°> ><(({°> ~
```

Sometimes fysh want to be different and swim the other direction. This takes away from the school's value. This gives Steven a value of `b101 = b111 - b010`

```
><steven>   ><{{{°> <°)})>< ~
```

Fysh often get lonely. This loneliness causes fysh to meet new fysh and proliferate. This gives Steven a value of `b101010 = b110 * b111`

```
><steven>   ><{{(°>   ><{{{°> ~
```

Since the fysh are lonely, they aren't too picky. They'll make do with a lesser form of love <3

```
><steven> = ><{{(o> <3 ><{{{o> ~
```

Not every fysh story is a happy one. At times, separation is unavoidable, and their division is symbolized by a heartbreak

```
><steven>   ><{{(({°>   ><{({°> ~
><steven> = ><{{(({o> </3 ><{({o> ~
```

As life goes on, we learn from our mistakes and improve. Steven's self help journey allowed them to grow an extra tail, incrementing their value by 1

```
>><steven> ~
```

Sometimes we feel like a fyshup, a failure. And that's ok, it's a part of being fysh. However for some fysh, this feeling is too much to handle and is internalized. They haven't received the emotional support they need and have gone on a downward spiral, causing them to feel worthless. They begin to retreat and try to swim away in the opposite direction causing their value to decrement by 1.

```
<steven><< ~
```

Sometimes Steven F*shs up and throws an error. This can be done using two WTF (What The Fysh) encompassing a string

```
><!@#$>
    What The Fysh?!
<!@#$><
```

Not all fysh are created equal. But sometimes they are. We can check this using or ==

```
><steven>   ><theFysh> ~
><steven> == ><theFysh> ~
```

But sometimes we want to ensure that two fysh are different. This can be done using ~ or ~=

```
><steven> ~  ><theFysh> ~
><steven> ~= ><theFysh> ~
```

Tadpoles, curious by nature, gravitate towards larger fysh, perhaps seeking guidance or a new destiny. This tadpole ponders if Steven has more than six fins. (Steven > sixFins, Steven >= sixFins)

```
><steven> o~ ><sixFins> ~
><steven> o~  ><sixFins> ~
```

Or, with a twist of fate, they explore if Steven is the lesser, a journey of humility and discovery. (Steven < sixFins, Steven <= sixFins)

```
><steven> ~o ><sixFins> ~
><steven> ~o  ><sixFins> ~
```

Steven is a curious fysh and often wonders if the world is as it seems. They ponder the meaning of life, the universe, and everything. They seek the truth, and if both steven and theTruth are true, they find it. (Steven && theTruth). Or if steven or theTruth is true, they find it. (Steven || theTruth). or if steven is not real and this is all a simulation, they find it. (!!Steven)

11-203 Donadeo Innovation Centre for Engineering
9211-116 Street NW
University of Alberta
Edmonton, Alberta
Canada T6G 1H9

```
><steven> && ><theTruth> ~
><steven> || ><theTruth> ~
!! ><steven> ~
```

Steven is looking for a change and is interested in having their bits rearranged. Given it's 2024, a time when open-mindedness prevails, Steven has a variety of bitwise manipulation techniques at his disposal: `AND (&)`, `OR (|)`, `XOR (^)`, `NOT (!)`. **For those moments when he's feeling particularly adventurous, he might even consider the** `logical shift left (<<)` **or** `logical shift right (>>)` **operators.**

```
><steven> &  ><{((°> ~
><steven> |  ><{((°> ~
><steven> ^  ><{((°> ~
><steven> !  ><{((°> ~
><steven> << ><{((°> ~
><steven> >> ><{((°> ~
```

In the whirlpool of fysh logic, the while loop, symbolized by `><(((@>`**, ensnares conditions within** `[ ]`**, with** `><>` **and** `<><` **encapsulating the iterative heart.**

```
><(((@> [ ><steven> o~ ><{((°> ]
><>
    <steven><< ~
<><
```

Navigating through decision streams, if and else statements flow naturally. `><(((^>` **marks the if,** `><(((*> ><(((^>` **for else if, and** `><(((*>` **for the uncharted else.**

```
><(((^> [ ><steven> o~ ><{((°> ]
><>
    <steven><< ~
<><

><(((*> ><(((^> [ ><steven> ~o ><{((°> ]
><>
    >><steven> ~
<><

><(((*>
><>
    ><steven>  ><(((°> ~
<><
```

For those seeking the fortune of randomness, `><###>` **unveils a 32-bit treasure, while** `><//>` **whispers single-line comment and** `></*>`**,** `<*\><` **span multiline scriptures.**

```
><###> ~

><//> Comment

></*>
Comments
Comments 2
<*/><
```

A fysh tank `[ ]` is an array of fysh separated by fysh food –

```
><steven>   [><({(°> - ><({(°>] ~
```

These fysh tanks can be traversed using a fysh

```
><steven>[><(({°>] ~
```

Factorial Example

```
><//> Comment

><number>     ><{({°>   ><//> b101 = 5
><factorial>  ><(({°>   ><//> b001 = 1

><//> while number > 1
><(((@> [><number> o~ ><(({°>]
><>
    ><//> factorial = factorial * number
    ><factorial>   ><factorial>   ><number> ~

    ><//> number -= 1
    <number><< ~
<><
```