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

Proposal Response: RISC-V FPGA-based CPU and Language

Proposal Sponsor: Steven Knudsen, Electrical and Computer Engineering Department

Objective

Design and implement a RISC-V compliant CPU with custom extensions on a Field-Programmable Gate Array (FPGA). Define a high-level language (code name: Fysh) and implement the toolchain necessary to run programs on the custom CPU.

Introduction

The realm of processor design has been primarily dominated by proprietary architectures. However, with the rise of RISC-V instruction set architecture, it's time for an open standard to gain popularity in the industry. It is becoming the basis of custom processor design because of its open and extensible nature. It is also gaining popularity in education for the same reasons.

Having an example implementation of a RISC-V CPU will be beneficial to UofA ECE students' education and exploration. By demonstrating a custom open source RISC-V extension, we aim to equip students with practical insights and readiness for real-world scenarios involving various processor architectures.

Alongside the custom CPU we intend to use the RISC-V architecture to implement our own custom language (Fysh). Making a language that is commercially useful is likely out of our scope, but for Fysh we are hoping for an interesting style, with a unique functionality that sets it apart from a standard language.

Proposed Solution

RISC-V CPU from Programmable Logic (PL)

For the RISC-V CPU, we intend to build it from scratch, referencing the example VHDL implementation of a CPU from the last ECE 410 Lab. The custom instruction shall be to generate a random 32-bit integer using hardware.

The hardware random number generator (RNG) component will be reading values from the 12-bit ADC on the Zybo Development board every clock cycle. Every time the ADC value is read, the 32 bit seed value shifts by 12 bits, effectively updating the seed every clock cycle.

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

The instruction simply then writes the current 32-bit seed of the “Hardware RNG Component” into the register file. A stretch goal for the softcore microprocessor would be to support additional standard extensions like Integer Multiplication (RV32M) and single-precision floating points (RV32F).

For the CPU’s physical memory, we will be utilizing the block RAM (BRAM) inside the FPGA. It is the easiest to implement because the hardware synthesizer is able to infer BRAM from VHDL code. A stretch goal for memory would be to interface with the development board’s DDR3L RAM, expanding the total physical memory up to 1 GB.

For the CPU firmware, we intend to “hardcode” it by directly synthesizing it into one of the available 128 KB blocks of memory. We believe that this is the simplest solution because it does not require any integration with external off-chip storage. This may pose risk to development time for the demo project as firmware compilation also requires synthesizing hardware. It also limits the physical memory that could be used as RAM. A stretch goal would be to read firmware from a micro SD card, removing the need to re-synthesize the hardware to change the firmware. Another stretch goal would be to be able to directly download the firmware through UART, potentially speeding up firmware development.

For the input and output ports, we intend to use Memory-mapped I/O, representing the I/O devices with a specific location in the address space. The initial goal would be to map the Pmod pins on the Zybo Z7-010 with a stretch goal to map the other I/O devices such as the audio jacks and HDMI.

Fysh Programming Language

The Fysh programming language is a statically-typed, procedural programming language that supports basic programming constructs like variables, integer arithmetic, bit manipulation, pointers, arrays, loops, and functions. Minimally, the language should support 8-bit, 16-bit, and 32-bit signed and unsigned integers as primitive types. Some stretch goals would be to have a module system, support type aliases, and create composite types such as structs and tuples.

Here is a sample representation of what the Fysh language may look like (the syntax is subject to change)

```
C/C++
// the fysh's body represents the value of the fysh
//  ( = 0, { = 1

><fysh> ≈ ><{{{{{o>    // assignment: fysh = b'10010' (normal equal signs
should work too)
><fysh> ♡ ><{{{{{o>   // fysh * b'110110' (<3 can also be used instead of ♡)
><fysh> 💔 ><{{{{{o>  // fysh / b'110110' (</3 can also be used instead of 💔)
><fysh>   ><{{{{{o>   // fysh + b'110110'
><fysh>   <o}}}}}><   // fysh - b'110110' (fysh direction represents its sign)
```

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

```
<fysh><                 // -fysh (unary negative)
>><fysh>                // fysh++
<fysh><<                // fysh--

><fysh> o~ ><{((o>      // fysh > b'100'   (tadpole swims to the bigger fysh)
><fysh> ~o ><{((o>      // fysh < b'100'   (tadpole swims to the bigger fysh)
><fysh> ≈≈ ><{((o>      // fysh == b'100'  (normal equal signs should work too)
><fysh> ~≈ ><{((o>      // fysh != b'100'  (normal equal sign should work too)

// bit manipulation (subject to change in final version cuz this is feels
uninspired)

><fysh> & ><{((o>       // fysh and b'101'
><fysh> | ><{((o>       // fysh or b'101'
><fysh> ^ ><{((o>       // fysh xor b'101'
^ ><fysh>               // bitwise not fysh
><fysh> << ><{((o>      // logical left shift
><fysh> >> ><{((o>      // logical right shift

><(((@> ~ ><fysh> o~ ><{((o> ~  // while fysh > b'100' (~ are brackets)
><>                             // open curly bracket
      <fysh><<                  // fysh--
<><                             // closing curly  bracket


><(((^> // if
><>
 // code here
<><

><(((*> ><(((^> // else if
><>
 // code here
<><

><(((*> // else
><>
 // code here
<><
><###> // random 32 bit number
```

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

```
><!@#$> // Error code
This shouldn't happen
<!@#$><
```

```
C/C++
// Factorial Example

><fysh>   ≈ ><{({o>    // b'101' (5 in binary)
><result> ≈ ><({{o>    // b'001' (1 in binary)

// While loop: Continue while fysh is greater than b'1' (1 in binary)
><(((@> ~ ><fysh> o~ ><({{o> ~
><>                                    // Opening curly bracket for while loop
       // Multiply result by fysh
       ><result> ≈ ><result> ♡ ><fysh>

       // Decrement fysh
       <fysh><<
<><                                   // Closing curly bracket for while loop
```

Fysh Compyler

The compiler for the Fysh programming language will likely be implemented in C++ because of its familiarity and native library support for LLVM and GCC. Because of the language's esoteric nature, it would be difficult to use an existing parser generator. To get around this, we intend to create a full implementation of the parser in either C++ or python. Minimally, the parser should be able to successfully parse a Fysh source file and output an abstract syntax tree. A stretch goal would be to display locations of syntax errors within a file.

For the middle end of the compiler, semantic analysis should be able to detect type errors in the program with a stretch goal to point out its location in the source code. We may also try to optimize for code size because of the limited available ROM in the PL.

For the machine code generation, we may use open-source compiler backends like GCC and LLVM to take advantage of their optimizations. We may have to write our own assembler to support the custom RISC-V CPU extensions and the custom execution environment.

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

LED Matrix Digital Aquarium (Demo Project)

To demonstrate the full integration between the programming language and the FPGA, we intend to create a project written in Fysh and download it to the board. Because of the complexity of the two other project components, we intend to keep the demo project simple.

To keep up with the fish theme, we will create a “digital aquarium” using a 32x32 or 64x64 RGB LED matrix. The “fish” shall move randomly, using the custom RISC-V “random” instruction to generate the pseudorandom number generator seed. Minimally, the digital aquarium should show a single multicolour fish swimming around the LED matrix. A stretch goal would be to add inputs that change the number of fish and their colours.

Project Overview Statement (last page)

Intellectual Property Statement

Note: Public presentation and, hence, disclosure is a course requirement.

Intellectual Property includes the HDL source code for the FPGA-based CPU and the source code for the programming language compiler. The HDL source code shall be released under the CERN Open Hardware Licence Version 2 - Strongly Reciprocal license while the compiler source code shall be released under the GNU Affero General Public License.

All source code is currently hosted on GitHub.

All hardware provided by the sponsor remains their property and will be returned.

Contacts

Charles Ancheta - cancheta@ualberta.ca
Kyle Prince - kjprince@ualberta.ca
Yahya Al-Shamali - yalshama@ualberta.ca

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

Project Overview Statement (POS)

Table 1 Project Overview Statement

Project Overview Statement	Project Name RISC-V FPGA-based CPU and Fysh Programming Language	Project Sponsor Steven Knudsen Project Number FYS-001	Project Manager Charles Ancheta

Problem/Opportunity

Address the need for a hands-on example of a RISC-V compliant CPU accompanied by a custom programming language readily available to UofA ECE Students.

Goal

Develop an FPGA-based RISC-V CPU and develop a compiler for a custom defined programming language. Program an example project that interfaces with a sensor using both components.

Objectives

1. CPU processes RV32I instructions along with a custom RISC-V extension.
2. Compiler outputs RV32I code and also supports custom RISC-V extension.
3. Example project compiles and runs on the Zybo development board.

Success Criteria

1. CPU capable of running RV32I machine code in 3 months.
2. Fysh programming language supports basic programming constructs.
3. Fysh compiler capable of parsing Fysh source code and outputting RV32I machine code.
4. Display single fish in the LED matrix.

Assumptions, Risks, Obstacles

1. Hardware (RISC-V CPU from Programmable Logic)
    a. Assumption: The execution environment can be freely defined
    b. Assumption: Hardware exceptions are not required to be implemented
    c. Risk: FPGA might not have enough logic blocks to support a full RISC-V compliant CPU and custom extension
    d. Risk: FPGA might not have direct access to the development board's EEPROM and RAM
2. Software (Compiler)
    a. Assumption: Compiler is not required to give extensive error messages
    b. Obstacle: Lack of experience creating a compiler
    c. Obstacle: Integration with compiler backend
    d. Obstacle: Writing a custom assembler because of custom instructions and execution environment
    e. Risk: Unoptimized compiler may produce firmware that would not fit in ROM
3. Firmware (Demo Project)
    a. Assumption: User input is not required for the final project
    b. Obstacle: Hard dependency on both the FPGA CPU and Fysh compiler
    c. Obstacle: Lack of experience programming in Fysh

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

d. Risk: Shipping time on the required parts of the project since we are only ordering the LED matrix initially

Potential Social, Environmental, Economic impacts

1. Provides an example of a fully-compliant RISC-V CPU for educational purposes

Prepared by	Date	Approved by	Date
Charles Ancheta, Kyle Prince, Yahya Al-Shamali	2024/01/25	<name>	<YYYY/MM/DD>