National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

F EXTENSION ON RV32I

Semester Project: Embedded System Design

Submitted to Dr. Rehan Ahmed

Submitted by

Naqash Naveed 177386

Safi Majid 174203

https://github.com/naqashnvd/RISC-V

Abstract

Float point operation support allow transcendental functions such as exponential or trigonometric calculations. It adds relative accuracy for granular calculations in comparison to integer operations. The implementation of floating-point extension to RV32I base on FPGA at the execution block allows floating point operations on float data types.

Table of Contents

Chapter 1: Introduction

Overview of Project

Encoding of a number is specified by how it is represented. A string of digits is all it needs to represent a number. However, a number can be represented by various mechanisms. In mathematics, a number can represented by a digit of any length with radix defined by addition of a radix point represented by a dot. The most common representation where the point or dot is not present is integer. For an integer the radix point is at extreme right. Floating point is a mode of representing numbers as two sequences of bits, one representing the digits in the number and the other an exponent which determines the position of the radix point.

Support of transcendental functions such as exponential or trigonometric calculations and added relative accuracy for granular calculations in comparison to integer operations is one of the key advantages of floating-point arithmetic. This project enables floating point operations on our 32-bit RISCV base integer core (RV32I) named as “R3XV”. RISC V is an open source ISA implementation based on reduced instruction set principles enabling hardware design implementations on granular level. Our “R3XV” implementation is five stage pipelined implementation of RISC base ISA enabling integer operations and this project adds floating point extension based on RISC-V manual to our RISC V. This was implemented via Verilog HDL and RTL was generated in Intel Altera Quartus and physically implemented on Cyclone 2 in for “Terasic DE-1” board.

Chapter 2: Design

RV32I (R3XV) Core

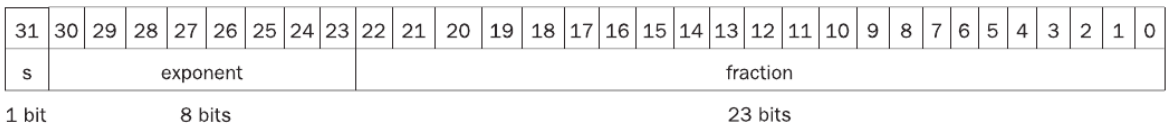
RV32I core previously implemented is a 5 staged pipelined with data forwarding and all integer base instruction support.

RV32I Base Integer Instruction Set

simm[31:12]					rd	0110111	LUI rd, imm
simm[31:12]					rd	0010111	AUIPC rd, offset
simm[20|10:1|11|19:12]					rd	1101111	JAL rd, offset
simm[11:0]			rs1	000	rd	1100111	JALR rd, rs1, offset
simm[12|10:5]		rs2	rs1	000	simm[4:1|11]	1100011	BEQ rs1, rs2, offset
simm[12|10:5]		rs2	rs1	001	simm[4:1|11]	1100011	BNE rs1, rs2, offset
simm[12|10:5]		rs2	rs1	100	simm[4:1|11]	1100011	BLT rs1, rs2, offset
simm[12|10:5]		rs2	rs1	101	simm[4:1|11]	1100011	BGE rs1, rs2, offset
simm[12|10:5]		rs2	rs1	110	simm[4:1|11]	1100011	BLTU rs1, rs2, offset
simm[12|10:5]		rs2	rs1	111	simm[4:1|11]	1100011	BGEU rs1, rs2, offset
simm[11:0]			rs1	000	rd	0000011	LB rd, offset(rs1)
simm[11:0]			rs1	001	rd	0000011	LH rd, offset(rs1)
simm[11:0]			rs1	010	rd	0000011	LW rd, offset(rs1)
simm[11:0]			rs1	100	rd	0000011	LBU rd, offset(rs1)
simm[11:0]			rs1	101	rd	0000011	LHU rd, offset(rs1)
simm[11:5]		rs2	rs1	000	simm[4:0]	0100011	SB rs2, offset(rs1)
simm[11:5]		rs2	rs1	001	simm[4:0]	0100011	SH rs2, offset(rs1)
simm[11:5]		rs2	rs1	010	simm[4:0]	0100011	SW rs2, offset(rs1)
simm[11:0]			rs1	000	rd	0010011	ADDI rd, rs1, imm
simm[11:0]			rs1	010	rd	0010011	SLTI rd, rs1, imm
simm[11:0]			rs1	011	rd	0010011	SLTIU rd, rs1, imm
simm[11:0]			rs1	100	rd	0010011	XORI rd, rs1, imm
simm[11:0]			rs1	110	rd	0010011	ORI rd, rs1, imm
simm[11:0]			rs1	111	rd	0010011	ANDI rd, rs1, imm
00000	00	shamt[4:0]	rs1	001	rd	0010011	SLLI rd, rs1, imm
00000	00	shamt[4:0]	rs1	101	rd	0010011	SRLI rd, rs1, imm
01000	00	shamt[4:0]	rs1	101	rd	0010011	SRAI rd, rs1, imm
00000	00	rs2	rs1	000	rd	0110011	ADD rd, rs1, rs2
01000	00	rs2	rs1	000	rd	0110011	SUB rd, rs1, rs2
00000	00	rs2	rs1	001	rd	0110011	SLL rd, rs1, rs2
00000	00	rs2	rs1	010	rd	0110011	SLT rd, rs1, rs2
00000	00	rs2	rs1	011	rd	0110011	SLTU rd, rs1, rs2
00000	00	rs2	rs1	100	rd	0110011	XOR rd, rs1, rs2
00000	00	rs2	rs1	101	rd	0110011	SRL rd, rs1, rs2
01000	00	rs2	rs1	101	rd	0110011	SRA rd, rs1, rs2
00000	00	rs2	rs1	110	rd	0110011	OR rd, rs1, rs2
00000	00	rs2	rs1	111	rd	0110011	AND rd, rs1, rs2

Floating Point Representation on RISC V

There is always a compromise between the size of the fraction and the size of the exponent, because a fixed word size means you must take a bit from one to add a bit to the other. This tradeoff is between precision and range: increasing the size of the fraction enhances the precision of the fraction, while increasing the size of the exponent increases the range of numbers that can be represented. Floating-point numbers are usually a multiple of the size of a word. [1]



$$(-1)^S \times F \times 2^E$$

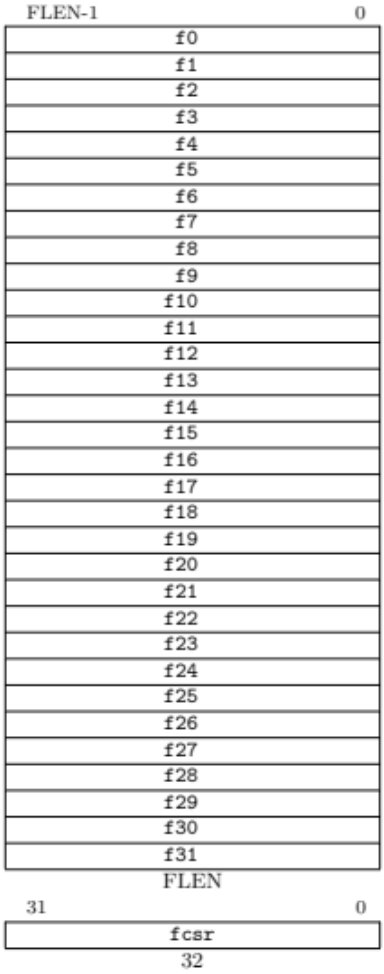
In this representation of a RISC-V floating-point number, s is the sign of the floating-point number (1 meaning negative), exponent is the value of the 8-bit exponent field (including the sign of the exponent), and fraction is the 23-bit number. [1]

This is part of the IEEE 754 floating-point standard, found in virtually every computer invented since 1980. This standard has greatly improved both the ease of porting floating-point programs and the quality of computer arithmetic. To pack even more bits into the number, IEEE 754 makes the leading 1 bit of normalized binary numbers implicit. Hence, the number is actually 24 bits long in single precision (implied 1 and a 23-bit fraction). To be precise, we use the term significand to represent the 24 that is 1 plus the fraction, and fraction when we mean the 23-bit number. Since 0 has no leading 1, it is given the reserved exponent value 0 so that the hardware won't attach a leading 1 to it. Thus 00 ... 00 represents 0; the representation of the rest of the numbers uses the form from before with the hidden 1 added:[1]

$$(-1)^S \times (F+1) \times 2^E$$

Floating Point Operands

The F extension adds 32 floating-point registers, f0–f31, each 32 bits wide. Most floating-point Instructions operate on values in the floating-point register file. Floating-point load and store instructions transfer floating-point values between registers and memory. Instructions to transfer values to and from the integer register file are also provided.[2]



Single Precision Floating Point Instructions

Following instruction support is added via use of Intel Floating Point IP Cores [3] for following Instructions:

Floating Point Addition and Subtraction

Addition and Subtraction of two single precision floating point operands is handled by fadd.s and fsub.s respectively. [4]

00000	00	frs2	frs1	rm	frd	1010011	FADD.S rm, frd, frs1, frs2
00001	00	frs2	frs1	rm	frd	1010011	FSUB.S rm, frd, frs1, frs2

Floating Point Multiplication and Division

Multiplication and Division of two single precision floating point operands handled by fmul.s and fdiv.s respectively. [4]

00010	00	frs2	frs1	rm	frd	1010011	FMUL.S rm, frd, frs1, frs2
00011	00	frs2	frs1	rm	frd	1010011	FDIV.S rm, frd, frs1, frs2

Floating Point Combined Arithmetic

Instructions with combined multiplication and addition and subtraction are handled by: [4]

frs3	00	frs2	frs1	rm	frd	1000011	FMADD.S rm, frd, frs1, frs2, frs3
frs3	00	frs2	frs1	rm	frd	1000111	FMSUB.S rm, frd, frs1, frs2, frs3
frs3	00	frs2	frs1	rm	frd	1001011	FNMSUB.S rm, frd, frs1, frs2, frs3
frs3	00	frs2	frs1	rm	frd	1001111	FNMADD.S rm, frd, frs1, frs2, frs3

Floating Point Square root

Square root of a single precision floating point operand is done by fsqrt.s. [4]

01011	00	00000	frs1	rm	frd	1010011	FSQRT.S rm, frd, frs1

Floating Point Conversions

Conversion from single Precision floating point to integer and vice versa is done by: [4]

11000	00	00000	frs1	rm	rd	1010011	FCVT.W.S rm, rd, frs1
11000	00	00001	frs1	rm	rd	1010011	FCVT.WU.S rm, rd, frs1
11010	00	00000	rs1	rm	frd	1010011	FCVT.S.W rm, frd, rs1
11010	00	00001	rs1	rm	frd	1010011	FCVT.S.WU rm, frd, rs1

Floating Point Comparison or Relational Operations

Less than, more than and equal to operations are governed by these instructions: [4]

10100	00	frs2	frs1	000	rd	1010011	FLE.S rd, frs1, frs2
10100	00	frs2	frs1	001	rd	1010011	FLT.S rd, frs1, frs2
10100	00	frs2	frs1	010	rd	1010011	FEQ.S rd, frs1, frs2

Following instruction support is added via Verilog HDL for following Instructions:

Signed Operations

Following signed operations [4] are implemented by Verilog HDL code:

00100	00	frs2	frs1	000	frd	1010011	FSGNJ.S frd, frs1, frs2
00100	00	frs2	frs1	001	frd	1010011	FSGNJN.S frd, frs1, frs2
00100	00	frs2	frs1	010	frd	1010011	FSGNJX.S frd, frs1, frs2

Maximum and Minimum operations

00101	00	frs2	frs1	000	frd	1010011	FMIN.S frd, frs1, frs2
00101	00	frs2	frs1	001	frd	1010011	FMAX.S frd, frs1, frs2

Following Floating point instructions were implemented using slight modifications in implementation of RV32I.

Load and Store Instructions

Loading from Memory to floating point reg_file and storing back is represented below: [4]

simm[11:0]		rs1	010	frd	0000111	FLW frd, offset(rs1)
simm[11:5]	frs2	rs1	010	simm[4:0]	0100111	FSW frs2, offset(rs1)

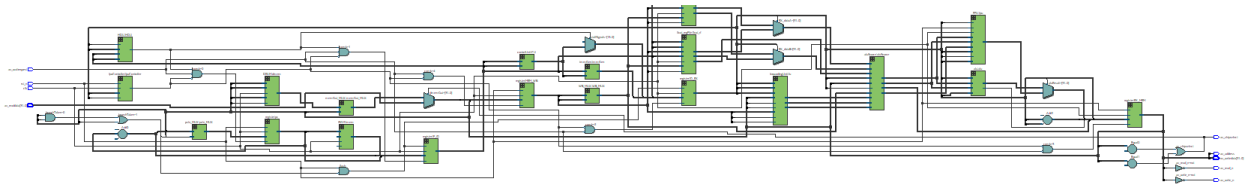
Move Instruction

Moving from and to Integer to and from single precision Floating point is done outside the FPU implementation.

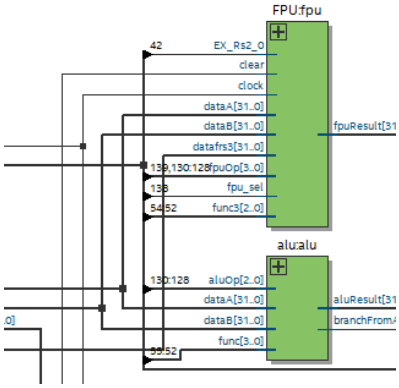
11110	00	00000	rs1	000	frd	1010011	FMV.S.X frd, rs1
11100	00	00000	frs1	000	rd	1010011	FMV.X.S rd, frs1

RTL Diagram

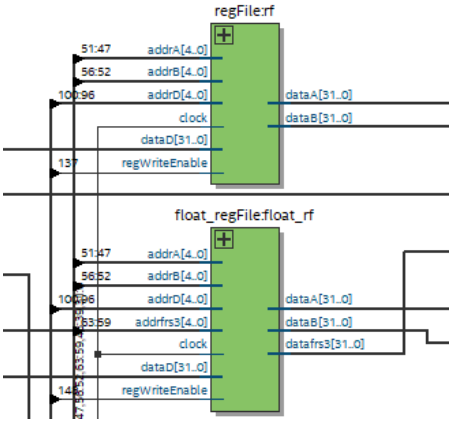
R3VX core with F extension

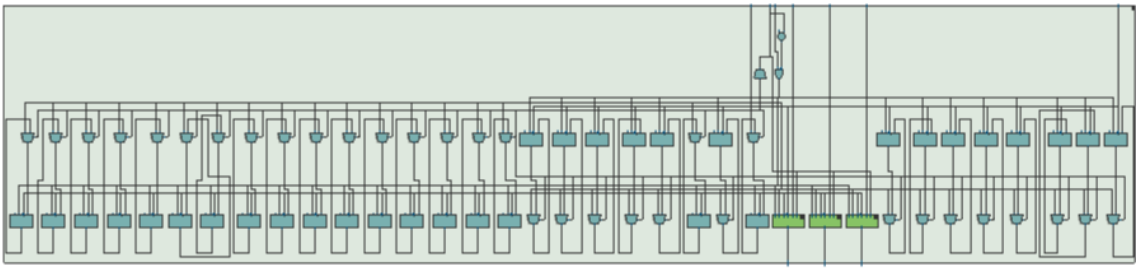


Magnified view of FPU with ALU

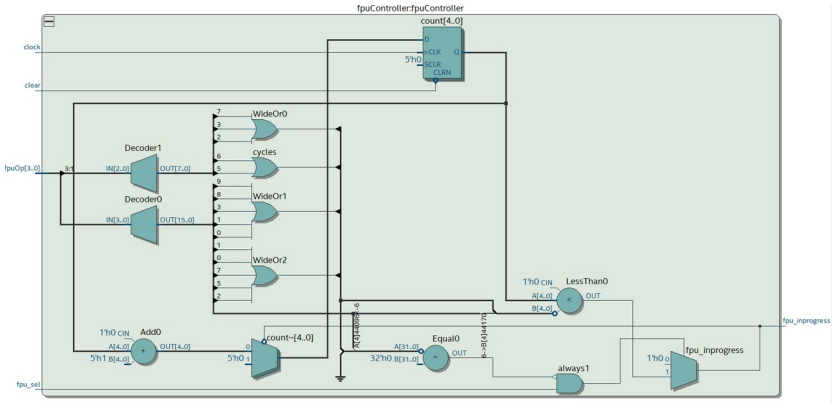


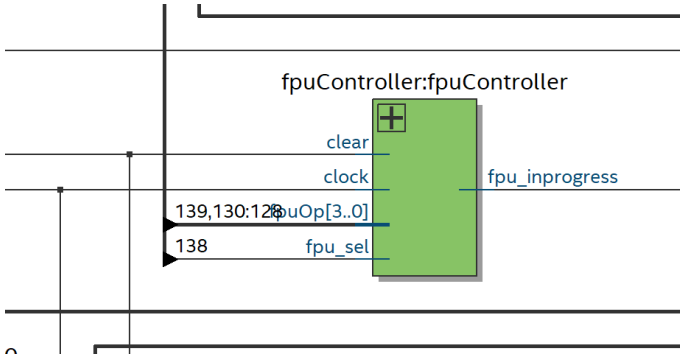
Magnified View of Floating-Point Reg_File





Magnified View of FPU Controller



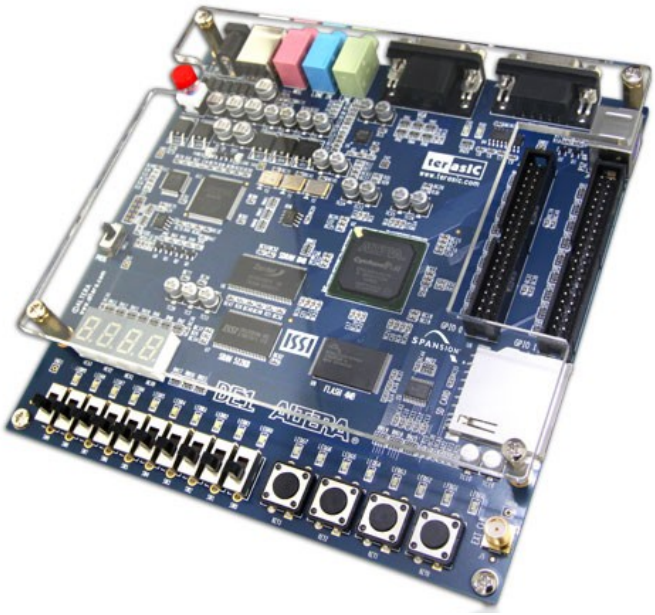


Chapter 3: Demonstration

Hardware Used

Terasic Altera DE-1 FPGA board with key specs as: [5]”

- Altera Cyclone II 2C20 FPGA with 20000 Les
- Altera Serial Configuration deivices (EPCS4) for Cyclone II 2C20
- USB Blaster built in on board for programming and user API controlling
- JTAG Mode and AS Mode are supported
- 8Mbyte (1M x 4 x 16) SDRAM
- 4Mbyte Flash Memory
- 512Kbyte(256Kx16) SRAM
- SD Card Socket
- 4 Push-button switches
- 10 DPDT switches
- 8 Green User LEDs
- 10 Red User LEDs
- 4 Seven-segment LED displays
- 50MHz oscillator ,24MHz oscillator ,27MHz oscillator and external clock sources
- 24-bit CD-Quality Audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (4-bit R-2R per channel) with VGA out connector
- RS-232 Transceiver and 9-pin connector
- PS/2 mouse/keyboard connector
- Two 40-pin Expansion Headers (GPIO: voltage levels: 3.3V)
- DE1 Lab CD-ROM which contains many examples with source code”



Code used

The following C code was used for demonstration. It was compiled into RV assembly by GCC and then to RV machine code and fed into Imem and Dmem via a Python script.

```
void _actual_start()
```

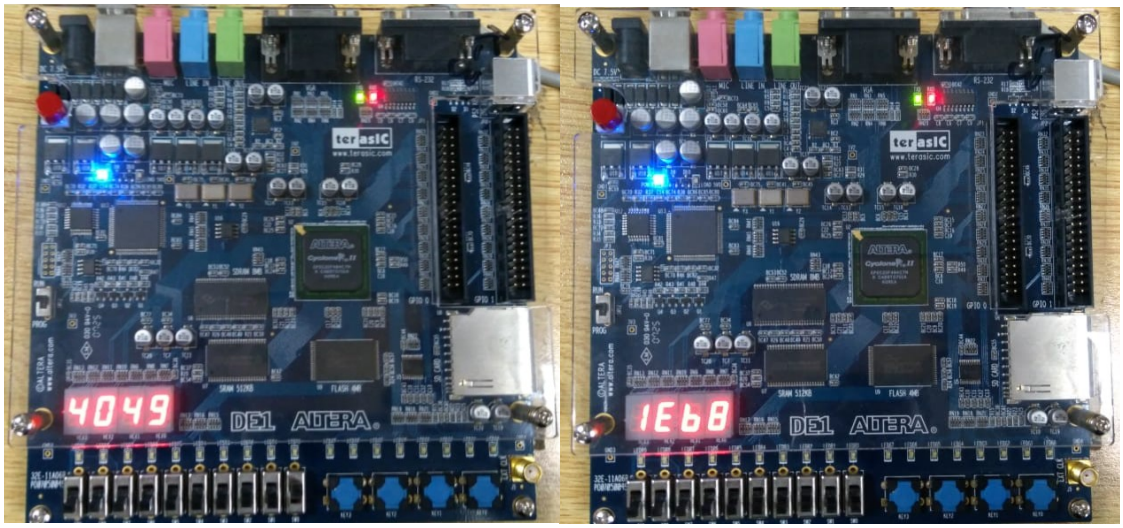
```
{
   volatile float a,b,c;
   a=12.57;
   b=2;
   c=a/(2*b);

   volatile float ptr  = (volatile float)0x108;
   *ptr =  c;

    while(1);
}
```

Output

The code produces the result 3.1425 i.e. approximately pi. The result produced is in IEEE 754 format and is displayed on 7 segment LCD display of DE-1 board where upper 4 bits and lower 4 bits mapped to ON and OFF state of SW0 to display. The output 0x40491eb8 is the representation of 3.1425 i.e. Our answer and proves the authenticity of our implementation



Future Improvement

- Increasing operating frequency of 50 MHz by optimization
- Addition of FCLASS instruction and CSR related instructions.
- Inclusion of interrupts and timers
- Full IEEE-754 implementation for FPU

References

1. David A. Patterson and John L. Hennessy, Computer Organization and Design RISC-V Edition: The Hardware Software Interface, Elsevier Science, 2017.
2. Andrew Waterman, Krste Asanović, SiFive Inc., The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Document Version 20190608-Base-Ratified, June 8 2019
3. Floating-Point IP Cores User Guide, Intel Altera. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_altfp_mfug.pdf .
4. RISC V Instructions guide for rv8 simulator for RISC V on x86-64. https://github.com/rv8-io/rv8/blob/master/doc/pdf/riscv-instructions.pdf .
5. Altera DE-1 board specs. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=83&PartNo=2