Fusion-Core ISA Definition: Revision 0.1

Dylan Wadler

October 18, 2017

Contents

1 Changelog

2 Introduction
- 2.1 About
- 2.2 Goals
- 2.3 Conventions

3 Register File Definitions
- 3.1 General Purpose Registers
- 3.2 Special Registers
  - 3.2.1 Control Registers
  - 3.2.2 Supervisor Registers
- 3.3 Adding Registers

4 Instruction Definitions
- 4.1 Instruction Types
  - 4.1.1 Integer
  - 4.1.2 Immediate
  - 4.1.3 Load/Store
  - 4.1.4 Branch/Jump
  - 4.1.5 Floating Point
  - 4.1.6 Atomic
  - 4.1.7 System
  - 4.1.8 Co-Processor
  - 4.1.9 Custom
- 4.2 List of Instructions
  - 4.2.1 Integer
  - 4.2.2 Immediate
  - 4.2.3 Load/Store
  - 4.2.4 Branch/Jump
  - 4.2.5 Floating Point
  - 4.2.6 Atomic
  - 4.2.7 System
  - 4.2.8 Co-Processor
  - 4.2.9 Custom

5 Exceptions and Interrupts
- 5.1 Exceptions
- 5.2 Interrupts
  - 5.2.1 User Level
  - 5.2.2 Supervisor Level

6 Co-Processor Interface
- 6.1 Interface Connection Definitions
- 6.2 Adding custom Co-Processor
- 6.3 List of Co-Processors

7 Memory Map

8 Programming Conventions

8.1 Register Usage . . . .

8.2 Memory Locations for Vector Table . . . .

8.2.1 Interrupt Vector Table . . . .

8.2.2 Exception Vector Table . . . .

1 Changelog

Version 0.1 Initial Definition of the Instruction Set Architecture

2 Introduction

2.1 About

The Fusion-Core ISA is dedicated to creating an easily expandible architecture without altering the instruction defining an easy interface with a simple core instruction set, this allows for more freedom in implementation. High and microcontrollers would only have slight varaitions in configuration, as their core would remain identical sa maintain and scalable co-processors.

The architecture is Big endian, with a core instruction set that is RISC, but the co-processors do not need to RISC philosophy. This allows for more flexibility in design, and possibly faster core clock speeds as the pipeline on smaller amounts of logic. Only the instructions provided in this document are to be implemented in the main co-processors defined in this document are recommended, but not required for normal function. Co-processor docu be provided by the creator, and should adhere to the standards of clarity and conciseness such that it can be easil from the documentation alone in a HDL.

64 Bit instructions: At this moment in time, the Fusion-Core ISA is only a 32 bit ISA. Due to the focus on older implementations could easily be modified to include 64 bit operations.

2.2 Goals

2.3 Conventions

Document Conventions: Example code will be shown with `monospace` text. General purpose registers will b $R# where # is the number of the register. Special purpose registers will be proceeded by a $ as well.

Naming Conventions: The name of input signals will have "_in" after the signal name, with "_out" after This is mainly used in the verilog example implementation. If a naming convention is not globally used, it will b individual section that it pertains to.

3 Register File Definitions

This section goes over the different registers available in the ISA. Each register file name begins with "REGF", s General Purpose Register File being REGFGP0. Any additional register files require the number after the name file. Register files with additional numbers after them are bank switched to reduce space, hence why the numbe denote the register file space used.

3.1 General Purpose Registers

32 general purpose registers that are 32 bits wide are available, as shown in Figure 1, below.

While it is not defined by the architecture, larger general purpose registers can be used instead of 32 bit wi larger registers are needed, consider using a co-processor to for instructions that require larger operands. This compatibility between different implementations.

Figure 1: General Purpose Registers (REGFGP0)

Register	Secondary use	Function Call State	Width
$R0	Hardcoded value of 0	Static	32 bits
$R1	Stack Pointer	Saved (callee)	32 bits
$R2	Frame Pointer	Saved (callee)	32 bits
$R3	Global Pointer	Saved	32 bits
$R4	Return Address	Saved (caller)	32 bits
$R5	System arg 0	Saved (caller)	32 bits
$R6	System arg 1	Saved (caller)	32 bits
$R7	System arg 2	Saved (caller)	32 bits
$R8	System arg 3	Saved (caller)	32 bits
$R9	System return 0	Saved (callee)	32 bits
$R10	System return 1	Saved (callee)	32 bits
$R11	System return 2	Saved (callee)	32 bits
$R12	System return 3	Saved (callee)	32 bits
$R13	Function arg 0	Saved (caller)	32 bits
$R14	Function arg 1	Saved (caller)	32 bits
$R15	Function arg 2	Saved (caller)	32 bits
$R16	Function arg 3	Saved (caller)	32 bits
$R17	Function return 0	Saved (callee)	32 bits
$R18	Function return 1	Saved (callee)	32 bits
$R19	Function return 2	Saved	32 bits
$R20	Function return 3	Saved (callee)	32 bits
$R21	Global Register 0	Saved (callee)	32 bits
$R22	Global Register 1	Saved (callee)	32 bits
$R23	Global Register 2	Saved (callee)	32 bits
$R24	Global Register 3	Saved (callee)	32 bits
$R25	Temporary Register 0	Volatile	32 bits
$R26	Temporary Register 1	Volatile	32 bits
$R27	Temporary Register 2	Volatile	32 bits
$R28	Temporary Register 3	Volatile	32 bits
$R29	Temporary Register 4	Volatile	32 bits
$R30	High word	Volatile	32 bits
$R31	Low word	Volatile	32 bits

NOTE: In writing programs, register arguments and return values can be used to reduce the number of register processes.

3.2 Special Registers
3.2.1 Control Registers
3.2.2 Supervisor Registers
3.3 Adding Registers
4 Instruction Definitions
4.1 Instruction Types
4.1.1 Integer
4.1.2 Immediate
4.1.3 Load/Store
4.1.4 Branch/Jump
4.1.5 Floating Point
4.1.6 Atomic
4.1.7 System
4.1.8 Co-Processor
4.1.9 Custom
4.2 List of Instructions
4.2.1 Integer
4.2.2 Immediate
4.2.3 Load/Store
4.2.4 Branch/Jump
4.2.5 Floating Point
4.2.6 Atomic
4.2.7 System
4.2.8 Co-Processor
4.2.9 Custom
5 Exceptions and Interrupts
5.1 Exceptions
5.2 Interrupts
5.2.1 User Level
5.2.2 Supervisor Level
6 Co-Processor Interface
6.1 Interface Connection Definitions
6.2 Adding custom Co-Processor
6.3 List of Co-Processors

8 Programming Conventions

8.1 Register Usage

8.2 Memory Locations for Vector Table

8.2.1 Interrupt Vector Table

8.2.2 Exception Vector Table

Address (32 bit)	Definition
0x0000	Reset address
0x0000	
0x0000	
0x0000	
0x0000	
0x0000	

Figure 2: Exception Vector Table