Fusion-Core ISA Definition: Revision 0.1

Dylan Wadler

October 21, 2017

Contents

1 Change log

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 set. By use of defining an easy interface with a simple core instruction set, this allows for more freedom in implementation. High end processors and microcontrollers would only have slight varaitions in configuration, as their core would remain identical save for easy to 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 adhere to the RISC philosophy. This allows for more flexibility in design, and possibly faster core clock speeds as the pipeline would depend on smaller amounts of logic. Only the instructions provided in this document are to be implemented in the main processor. The co-processors defined in this document are recommended, but not required for normal function. Co-processor documentation is to be provided by the creator, and should adhere to the standards of clarity and conciseness such that it can be easily implemented 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 co-processors, 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 be denoted with $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 output signals. This is mainly used in the verilog example implementation. If a naming convention is not globally used, it will be stated in the 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", such as the first General Purpose Register File being REGFGP0. Any additional register files require the number after the name of the register file. Register files with additional numbers after them are bank switched to reduce space, hence why the number is required to denote the register file space used.

3.1 Register File List

Th

REGFGP0	
Register	Register Name
$R0	ZERO
$R1	SP0
$R2	FP0
$R3	GP0
$R4	RA0
$R5	ARG00
$R6	ARG01
$R7	ARG02
$R8	ARG03
$R9	ARG04
$R10	ARG05
$R11	RVAL00
$R12	RVAL01
$R13	RVAL02
$R14	RVAL03
$R15	RVAL04
$R16	GPR00
$R17	GPR01
$R18	GPR02
$R19	GPR03
$R20	GPR04
$R21	GPR05
$R22	GPR06
$R23	GPR07
$R24	TMPR00
$R25	TMPR01
$R26	TMPR02
$R27	TMPR03
$R28	TMPR04
$R29	TMPR05
$R30	REGHI0
$R31	REGLOW0

REGFGP1	
Register	Register Name
$R0	ZERO
$R1	SP1
$R2	FP1
$R3	GP1
$R4	RA1
$R5	ARG10
$R6	ARG11
$R7	ARG12
$R8	ARG13
$R9	ARG14
$R10	ARG15
$R11	RVAL10
$R12	RVAL11
$R13	RVAL12
$R14	RVAL13
$R15	RVAL14
$R16	GPR10
$R17	GPR11
$R18	GPR12
$R19	GPR13
$R20	GPR14
$R21	GPR15
$R22	GPR16
$R23	GPR17
$R24	TMPR10
$R25	TMPR11
$R26	TMPR12
$R27	TMPR13
$R28	TMPR14
$R29	TMPR15
$R30	REGHI1
$R31	REGLOW1

REGFEXCP	
Register	Register Name
$R0	ZERO
$R1	SP1
$R2	FP1
$R3	GP1
$R4	RA1
$R5	ARG0
$R6	ARG1
$R7	ARG2
$R8	ARG3
$R9	ARG4
$R10	ARG5
$R11	RVAL0
$R12	RVAL1
$R13	RVAL2
$R14	RVAL3
$R15	RVAL4
$R16	GPR0
$R17	GPR1
$R18	GPR2
$R19	GPR3
$R20	GPR4
$R21	GPR5
$R22	GPR6
$R23	GPR7
$R24	TMPR0
$R25	TMPR1
$R26	TMPR2
$R27	TMPR3
$R28	TMPR4
$R29	TMPR5
$R30	REGHI1
$R31	REGLOW1

REGFSYSCL0	
Register	Register Name
$R0	ZERO
$R1	SP1
$R2	FP1
$R3	GP1
$R4	RA1
$R5	SYSARG00
$R6	SYSARG01
$R7	SYSARG02
$R8	SYSARG03
$R9	SYSARG04
$R10	SYSARG05
$R11	SYSRVAL00
$R12	SYSRVAL01
$R13	SYSRVAL02
$R14	SYSRVAL03
$R15	SYSRVAL04
$R16	SYSGPR00
$R17	SYSGPR01
$R18	SYSGPR02
$R19	SYSGPR03
$R20	SYSGPR04
$R21	SYSGPR05
$R22	SYSGPR06
$R23	SYSGPR07
$R24	SYSTMPR00
$R25	SYSTMPR01
$R26	SYSTMPR02
$R27	SYSTMPR03
$R28	SYSTMPR04
$R29	SYSTMPR05
$R30	SYSREGHI0
$R31	SYSREGLOW0

3.2 General Purpose Registers

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

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

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

3.3 Special Registers

3.3.1 Control Registers

3.3.2 Supervisor Registers

3.4 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

4.2 List of Instructions

4.2.1 Integer

4.2.2 Immediate

4.2.3 Load/Store

4.2.4 Branch/Jump

5 Exceptions and Interrupts

5.1 Exceptions

5.2 Interrupts

5.2.1 User Level

5.2.2 Supervisor Level

6 Co-Processors

6.1 Co-Processor Interface

6.1.1 Co-Processor Conventions

6.1.2 Register Connections

6.1.3 Decode unit Connections

6.2 Interface Connection Definitions

6.3 Adding custom Co-Processor

6.4 List of Co-Processors

6.4.1 Floating Point

6.4.2 System Unit

6.4.3 Atomic Unit

6.4.4 Memory Management Unit

6.4.5

7 Recommended Co-Processors

9 Programming Conventions

9.1 Register Usage

9.2 Memory Locations for Vector Table

9.2.1 Interrupt Vector Table

9.2.2 Exception Vector Table

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

Figure 1: Exception Vector Table