A Custom Designed RISC-V ISA Compatible Processor for SoC

Kavya Sharat(✉), Sumeet Bandishte, Kuruvilla Varghese, and Amrutur Bharadwaj

Indian Institute of Science (IISc), Bangalore, India
kavya.sharat@gmail.com, sumeet.bandishte30@gmail.com, edkuru@dese.iisc.ernet.in, amrutur@ece.iisc.ernet.in

Abstract. RISC-V is an open Instruction Set Architecture (ISA) released by Berkeley Architecture Group from the University of California, at Berkeley (UCB) in 2010. This paper presents the architecture, design and complete implementation of a 32-bit customisable processor system containing a mix of features as listed below. The 32-bit processor based on RISC-V ISA, is capable of handling atomic operations in addition to all integer operations supported by the ISA. The design has a priority-based nested interrupt controller, giving the user an added flexibility to program the priority levels of interrupts. In addition, there is a debug unit which provides internal visibility during program execution. An error detection and correction interface to memories, makes the design resilient to radiation induced bit-flips. The on-chip communication interface follows the standard Wishbone specification. The design has been implemented on Xilinx Virtex-7 XC7VX48T FPGA and achieves a peak frequency of 80 MHz, with the processor stand-alone operating at 190 MHz. On a 65 nm technology node, the design operates at a frequency of 170 MHz, while the processor stand-alone, a maximum frequency of 220 MHz. The design occupies a footprint of 1.027 mm^2 with 32-KB on-chip memory.

Keywords: Processor · Pipeline · Cache · Interrupt controller
Error handling · Debug unit

1 Introduction

Processors are found in a plethora of applications from embedded computers in toys to industrial control systems. Most of them have proprietary ISAs which makes customisation difficult and expensive. A custom designed system based on an open ISA is beneficial for fast and variety of development.

1.1 RISC-V

RISC-V is an open-source ISA released by Berkeley Architecture Group from the University of California at Berkeley(UCB) in 2010 [1]. Its aim is to accelerate research and development in the field of computer architecture.

B. K. Kaushik et al. (Eds.): VDAT 2017, CCIS 711, pp. 570–577, 2017.
https://doi.org/10.1007/978-981-10-7470-7_55

1.2 Related Works

System-on-Chip development in the field of RISC-V started with the release of "Rocket Chip" [7]. It is tethered to an ARM core and requires intervention from the core to emulate DRAM and peripherals.

In [5], the designers have released a soft core in chisel [4], called 'ZSCALE', following the RISC-V RV32IM extension. The 'VSCALE' [6] is the Verilog version of 'ZSCALE'. mRISC-V, a RISC-V based micro controller, is described in [8]. Since, none of these implementations till date have a combination cache, atomic instruction support, configurable nested interrupt handling capability, error handling feature, it was decided to build the entire system from scratch. Many more implementations of RISC-V like PULPino [9], lowRISC [10] have been come up lately.

2 Processor System Design

Our system has a 32-bit processor following RV32IMA extensions of RISC-V, Instruction and Data Cache, Wishbone bus interconnect, configurable peripherals, interrupt controller and a Debug Unit.

2.1 Processor

The processor is a 5-stage, in-order execution machine. The 5 stages are fetch (F), decode (D), execute (E), memory (M) and write-back (W). The stages and their functionality is similar to that in conventional pipeline, with a few additions. There is an *amo* block is for atomic instructions. Also, the execute stage has a multiplier and divider module.

2.2 Cache

The cache architecture is chosen to be 2-way set-associative, since it serves as a balance between increase in miss-rate in case of direct mapped cache and increase in hit-time in case of fully associative cache. The Data and Instruction Cache are of size 16KB each, with 32-byte block size. Write-back policy is used to avoid the cost of writing to memory every time change is made to a cache block.

Data Cache: The Data Cache (D-Cache) holds the data section of the program serving load/store requests from the processor. Figure 1 shows the top level view of the data cache.

The command sequencer is the interface to the processor pipeline. It handles requests arriving at both the ports- in D and M stages. It prevents simultaneous read and write to the same cache line. It infers contention from the addresses and gives priority to the M stage request since it is chronologically older.

There are two port control units (PCUs) each handling the request of an access port. After arbitration (if required), the ports control units perform the

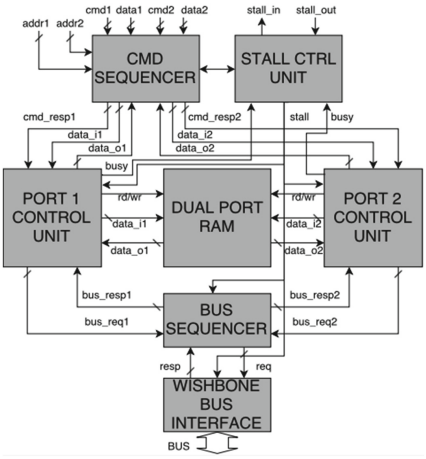

Fig. 1. Data Cache organisation

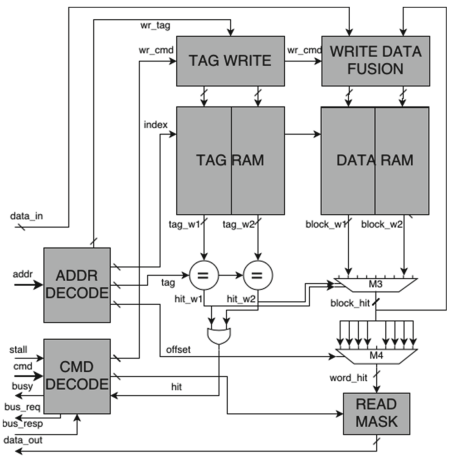

Fig. 2. Port Control Unit + RAM

read/write to the Data and Tag RAMs. Figure 2 illustrates a port control logic along with the Data and Tag RAMs for a 2-way set associative configuration.

The address decode extracts index, offset and tag from a 32-bit address. Index addresses the RAMs, tag is used to check for a match and offset is used to extract the word from the cache line. Multiplexer M3 selects between the 2 ways based on the tag check. M4 selects a word (32-bit) from a cache block. When reading data smaller than a word length (32-bit), Read Mask logic extracts data from a word and performs sign extension.

Instruction Cache: The Instruction Cache (I-cache) stores the instructions of the program. Figure 3 shows the block schematic of the I-cache.

The instruction cache takes in requests from only the Fetch (F) stage. Hence does not require a command sequencer to handle contentions. Also, a single port RAM suffices for the I-cache. The *cmd* signal input to the I-cache is *rd* (read) as the I-cache is read only. The rest of the blocks and working is as explained in the D-Cache section.

2.3 Interrupt Handling

The system currently supports nested interrupts from upto 32 sources. There is a provision to add more interrupts, if required. These interrupts can be dynamically assigned to any of the 4 priority levels.

The handling of interrupts is done by 2 units:

1. Interrupt controller, which is a Wishbone compatible peripheral
2. Interrupt interface towards the processor side.

The interrupt controller receives interrupts from peripherals and sorts them according to priority. It then updates its registers and signals the CPU by

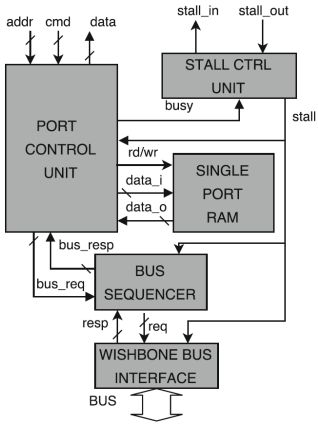

Fig. 3. Instruction Cache organisation

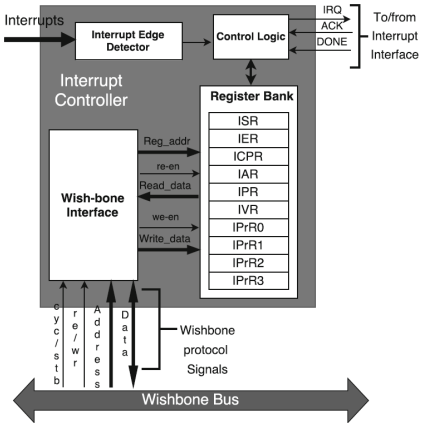

Fig. 4. Handshake between Interrupt Controller and Interrupt interface

sending an *IRQ* request. The interrupt interface towards the processor side saves the current state of the program onto the stack and then sends an *ACK* signal acknowledging the interrupt controller. The processor vectors to a common interrupt handler and the interrupt service routine queries the interrupt controller, by reading its register bank to determine the source of the interrupt.

The encounter of *ERET* instruction, as specified by RISC-V ISA marks the end of the interrupt service routine [11]. The interrupt interface restores the current state of the program from the stack. A *DONE* signal is sent from the RISC-V core. The interrupt controller updates its registers with this signal.

Interrupt Controller. Figure 4 shows the interrupt controller, its internal structure and its interface signals.

Interrupt edge detector: This detects a rising edge on any of the interrupt lines, and asserts a trigger on the Control Logic.

Register bank: The interrupt controller has a set of memory mapped registers: Interrupt Enable Register (IER), Interrupt Pending Register (IPR), Interrupt Active Register (IAR), Interrupt Current Priority Register (ICPR), Interrupt Status Register (ISR), Interrupt Priority Registers (IPrR0-3) and Interrupt Vector Register (IVR)

Control Logic: On a trigger from the Interrupt Edge Detector, the control logic block reads the Interrupt Enable Register (IER) and sends an *IRQ* request, only if the interrupt is enabled. The control logic also updates the Current priority register (ICPR), Status register (ISR), Active (IAR) and Pending registers (IPR). When a second interrupt occurs while an interrupt is in progress, the

control logic reads the Register Bank (ICPR and IPrR0-3) to decide whether it has to pre-empt if it is of higher priority level or be kept pending if it is of lower priority level. On the receipt of a *DONE* signal, it updates its registers by clearing the corresponding pending and active bits (IPR and IAR).

Wishbone Interface: The interrupt controller has a wishbone interface allowing access to specific registers in its Register Bank. Configuring the priority levels of interrupts by writing to IPrR0-3 and enabling/disabling interrupts by writing to IER is done through the bus interface.

Interrupt Interface: The interrupt interface towards the processor side saves the state of the program by injecting a set of stored instructions. When an *IRQ* request signal is received, these instructions are injected to the decode stage. Similarly, an *ERET* signal, triggers injection of instructions to restore the program state.

2.4 Debug Unit

The on-chip Debug Unit assists in debugging software running on the processor providing the following capabilities: halt, resume, single step, reset the system, reading register and memory contents. The communication with the host is done with a two-wire cable following the UART (Universal Asynchronous Receiver/Transmitter) protocol. As shown in Fig. 5, the debug unit has 2 submodules: Debug Support Unit (DSU) and Debug Handle Unit (DHU).

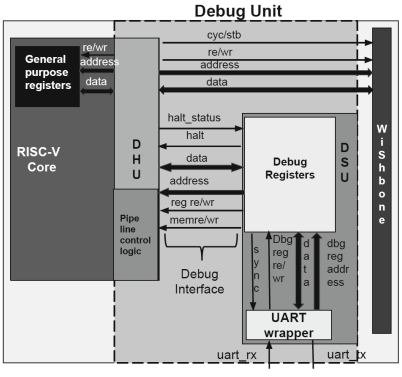

Fig. 5. Debug Unit

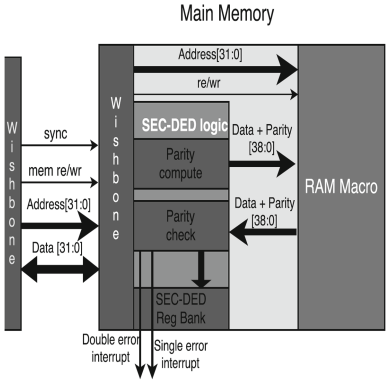

Fig. 6. Error correction interface

Debug Support Unit (DSU). The DSU sends commands to the Debug Handle Unit based on the commands received from the user. The DSU has a set of debug registers, controlling program flow. They are not memory mapped and user-programmed through UART. The DSU also has a UART wrapper controlling the transmission and reception of data, adhering to the UART protocol.

Debug Handle Unit (DHU). The DHU interfaces to the processor core. It communicates with the DSU through the debug interface as shown in Fig. 5. The DHU provides access the general purpose register and memory contents for read/write operations while debugging. Through the Pipeline Control Logic block, it controls data flow through the pipeline stages.

2.5 Error Handling

There is probability of soft errors in memories, due to interference from alpha-particle or cosmic rays, which disrupt program execution. Hence, memories are made fault-redundant using Error Correction Codes (ECCs). The error correction code used in our implementation is Single Error Correction Double Error Detection (SEC-DED) [13].

Figure 6 shows the main memory with error handling interface. Our 32-bit processor, requires 6 bits for single error correction and 1 additional bit for double error detection. The wishbone interface receives synchronisation, memory read/write, address and data signals from the core through the wishbone bus and accordingly asserts signals to the RAM macro. During a memory write operation, the Parity Compute block computes the 7 parity bits, and they are interspersed between the data bits and written to memory [13]. During a memory read, the Parity Check block re-computes the parity bits and compares it with the stored parity bits. Mismatch indicates an error raising single and double error interrupts as shown in Fig. 6. The SEC-DED register bank has memory-mapped Address Register and Correct Data Register which records the erroneous memory address and the corrected data respectively. Single-bit errors in memory are corrected by writing the corrected data to memory in the interrupt service routine. Double-bit errors terminate the program.

3 ASIC Implementation

The processor system is implemented on a 65 nm UMC technology node. The design is synthesized using Cadence RTL Compiler into a gate-level netlist. The RAM macro blocks for the instruction and data cache have been generated using the Standard Memory Compiler, Memaker 201201.1.1, provided by Faraday Technology Corporation. Cadence Conformal LEC tool is used to check the equivalence of the golden RTL and the synthesized netlist. In the backend design, we use Cadence Encounter to do the power distribution, floorplanning, placement, clock tree synthesis, design routing and obtain the GDSII file.

4 Results

4.1 FPGA Implementation

Design and implementation is done in Xilinx Vivado 2015.4 with Virtex-7 FPGA (XC7VX485T) as target. Table 1 shows the resource utilisation after Place and Route. RAM 36/18 refers to SRAM of si ze 36/18 Kilo bits.

Table 1. Resource utilisation

Module	Look-Up tables	Reg	RAM 36	RAM 18
Top	14462	6927	90	2
Processor	4208	840	0	0
D-Cache	6327	3310	18	0
I-Cache	807	923	8	2
Interrupt controller	819	354	0	0
Bus controller	311	156	0	0
Debug	356	229	0	0

Table 2. Comparison of our processor system with Vscale implementation on FPGA

Parameter	VScale	Our processor	SoC
Look-Up tables	2500	4208	14462
Reg	1006	840	6927
Max Freq (MHz)	131	190	121(no-RMW) 80(RMW)

The data cache is shown to have the highest resource utilisation due to hardware support for RMW operations. Table 2 shows a comparison of our processor system with Vscale processor from UCB [6] in Xilinx Vivado 2015.4 Design Suite with Virtex-7 FPGA target.

Our processor system offers a frequency speedup of about 1.4x compared to the VScale processor. The full SoC achieves a maximum frequency of 121 MHz without RMW support and 80 MHz with RMW support.

4.2 ASIC Implementation

Table 3 shows the results of our design synthesized with 65 nm UMC process technology, as compared with some of the commercial implementations. Though pipelined with five stages, our processor occupies lesser area. However, the overall system area is higher due to the 32KB on-chip RAM, for the instruction cache

Table 3. Comparison of ASIC implementation of our processor system with commercial implementations

Parameter	Shakti F-class [15]	MicroRISC-V [14]	Our System
Architecture	5-stage	3-stage	5-stage
Process	55 nm	130 nm	65 nm
Area (mm^2)	0.27	0.12 (Processor) 0.35 (SoC+RAM)	0.0497 (Processor) 0.0794 (SoC), 1.027 (SoC+RAM)
RAM (on-chip)	Not available	4 KB	32 KB
Max freq (MHz)	416	100	220 (Processor), 170 (SoC)
Instance cnt	25176	Not available	24907

and data cache. Process, voltage and temperature conditions for corner cases include: *Best case corner* - fast, 1.32 V, −40° C; *Worst case corner* - slow, 1.08 V, 125° C

5 Conclusion

This paper presents the design and implementation of a RISC-V ISA compatible processor system. The scope of extensions in the design include adding virtual memory support, floating point instruction support and multiple execution modes. Power saving techniques in ASIC flow could be explored.

References

1. RISC-V, The Free and Open RISC Instruction Set Architecture. RISC-V Foundation (2016). https://riscv.org. Accessed 14 Jun 2016
2. Waterman, A., Lee, Y., Patterson, D.A., Asanovic, K.: The RISC-V instruction set manual, volume i: base user-level ISA. EECS Department, UC Berkeley, Technical report UCB/EECS-2011-62 (2011)
3. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach. Elsevier (2011)
4. Chisel. The Regents of the University of California (2015). https://chisel.eecs.berkeley.edu. Accessed 14 Jun 2016
5. Magyar, A., Lee, Y., Ou, A.: Z-Scale: Tiny 32-bit RISC-V Systems with Updates to the Rocket Chip generator. The International House, Berkeley (2015)
6. Verilog version of Z-scale, vscale (2016). https://github.com/ucb-bar/vscale. Accessed 14 Jun 2016
7. Schmidt, C.: "RISC-V" Rocket Chip "Tutorial". UC Berkeley (2015)
8. Duran, L.R.C., et al.: A 32-bit RISC-V AXI4-lite bus-based Microcontroller with 10-bit SAR ADC. In: VII Latin American Symposium on Circuits and Systems (LASCAS) (2016)
9. PULPino. http://www.pulp-platform.org. Accessed 25 May 2017
10. https://en.wikipedia.org/wiki/LowRISC. Accessed 25 May 2017
11. Waterman, A., Lee, Y., et al.: The RISC-V Instruction Set Manual, Volume II: Privileged Architecture. CS Division, EECS Department, University of California, Berkeley (2015)
12. Girard, O.: OpenCores openMSP430, Revision 1.13, 19 May 2015
13. Error Detection and Correction: Supplement to Logic and Computer Design Fundamentals. Pearson Education (2004)
14. Duran, C., Rueda, L., Castillo, G., et al.: A 32-bit 100 MHz RISC-V Microcontroller with 10-bit SAR ADC in 130 nm CMOS GP. In: Third RISC-V Workshop Proceedings (2016)
15. Gupta, S., Gala, N., et al.: SHAKTI-F: a fault tolerant microprocessor architecture. In: IEEE 24th Asian Test Symposium (2015)