FPGA IMPLEMENTATION OF A RECONFIGURABLE MICROPROCESSOR

Jacob Davidson

University of Quebec in Montreal, P.O.B. 8888, Station A,
Montreal, Quebec, Canada, H3C 3P8

Abstract

This paper describes the implementation of an 8-bits reconfigurable microprocessor (RM) in a memory based FPGA device (XILINX). The RM is designed as an 8 bits microprocessor with a complete instruction set (41), hardware and software interrupts and 2 Kb addressing range (11 bits address bus). The paper presents the trade-offs involved in designing the architecture, the design for performance issues and the possibilities for future development.

Introduction

Memory based Field Programmable Gate Arrays (FPGAs) have the advantage of real-time in-circuit reconfigurability as opposed to other gate arrays of similar gate density. This advantage translates into unlimited, in-circuit flexibility, reconfigurability and reliability, facilitating prototyping of complex electronic designs. Usually, FPGAs are used as LSI replacement on low volume production or prototyping devices which are to be eventually implemented as ASICs. Their 100% testability and the possibility of achieving a high degree of fault coverage makes them increasingly attractive for complex designs with multiple iterations on their design cycles.

Purpose of the reported work

Certain real-time applications require special-purposes, custom-made microprocessors which are difficult to manufacture because of the limited market demand. One of the solutions available today are the memory based FPGAs combining reconfigurability and flexibility with thousands of logic gates. For these reasons, a memory based FPGA (XILINX) was considered for the design of an 8 bits reconfigurable microprocessor with 2 Kb to 4 Kb addressable memory and a comprehensive instruction set. The advantages of an FPGA based reconfigurable microprocessor (RM) are:

- *Increased throughput.* FPGAs can operate at the maximum circuit speed and consequently they can perform several times faster than a programmed general-purpose microprocessor.
- *Low cost implementation.* The programming of the connections between logic gates inside an FPGA is much cheaper for each new application than the manufacturing of a Gate Array, specifically in the case of a limited edition and frequent reconfiguration of the microprocessor.
- *Increased productivity.* In order to reconfigure the microprocessor a new software has to be downloaded into its internal memory (XILINX circuits are memory based FPGAs), which is very similar to downloading a new software program in memory. Consequently, the cost of designing new RMs obeys the dynamics of the software industry rather than the hardware industry.
- *Improved testing.* All the internal registers, internal data bus and address bus can be observed and controlled directly on input/output pins assigned specifically to this task.

Contributions relative to previous work

Several microprocessor designs based on FPGA are reported in the literature (see references 1. to 5.). Previous work was mostly intended for teaching purposes, resulting in very limited capacity microprocessors, with only one or two registers, small address space (256 bytes addressing space - reference 5.) and no subroutine call or interrupt facilities.

The reconfigurable microprocessor has all the characteristics of a complete 8 bits microprocessor, with an accumulator A (8 bits), an index register X (11 bits), a stack register S (11 bits), a program counter PC (11 bits), an instructions register IR (16 bits), an 8 bits arithmetic logic unit (ALU) and a decoder. It also has 41 instructions (see Instructions set), including Jump Subroutine - JSR, Software Interrupt - SWI, Return from Interrupt - RTI (for software and hardware interrupts). Moreover, this

IEEE 1993 CUSTOM INTEGRATED CIRCUITS CONFERENCE

microprocessor can reconfigure itself by starting the process of loading a new configuration into his internal memory.

Microprocessor design

The microprocessor architecture was targeted specifically for implementation in different FPGAs. This was carried out by using multiplexers instead of tristate buffers as bus switching elements.

Fig. 1 shows the block diagram of the reconfigurable microprocessor architecture. The chip has 22 one-byte instructions (#1 to #22) and 19 two-bytes instructions (#23 to #41), based on 5 bits opcodes and 11 bits addresses. The 16 bits instructions register (IR) separates the opcode from the address, allowing the decoder to determine the sequence of operations needed to execute the instructions. The other registers are: accumulator A (8 bits), program counter PC (11 bits), index register X (11 bits) and stack register S (11 bits).

The instructions are divided in 4 address modes: inherent (#1 to #17), indexed (#18 to #22), immediate (#23 to #25) and extended (#26 to #41). From the point of view of their impact on the arithmetic logic unit (ALU), the instructions are further divided into 3 groups: pass data - PD (instructions: pull, lda -x, lda x+, lda #val, lda addr), no_load accumulator - NL (instruction cmp #val) and pass accumulator - PA (the rest of the instructions).

The microprocessor uses 3 flags: zero - Z, negative - N and carry - C, used respectively by the instructions: BEQ (branch if equal), BMI (branch if minus) and BCS (branch if carry set). The carry flag is also used by the instructions: LSL (logical shift left) and LSR (logical shift right).

From the total address space of 2 Kb, two addresses are decoded by the microprocessor internally, representing a 16 bits output register. This register is used for addressing the 32 Kb RAM into which are downloaded the new microprocessor configurations.

The microprocessor has a 16 Mhz external clock, from which are generated 3 other clocks used for instruction decoding and execution. The majority of instructions execute in a "fetch2" cycle because they require two memory fetch cycles to load the 16 bits instruction register - IR. Only the JSR (jump subroutine) instruction requires a double fetch2 cycle, needed to save the PC in the stack and to load the new jump address.

Hardware and software development

The RM was designed with XILINX 4010 device (10,000 gates, 191 pins), using XILINX software tools, Mentor Graphics schematics editor NETED and logic simulator QUICKSIM, on a 400T (12 Mips) Hewlett-Packard workstation. The microprocessor was extensively simulated with QUICKSIM before implementation and after place and route, requiring several modifications of the initial design.

A cross-assembler was developed (using C language), to translate assembler programs written for the microprocessor. Also, a microprocessor prototype board was designed, on which the RM was coupled with an ACIA - Asynchronous Communication Interface Adapter. A software kernel was written (1/2 Kb) for interfacing with an IBM-PC through a serial RS-232C port.

The prototype board has a 32 Kb EPROM, containing the 4010 configuration data (22 Kb), and the kernel software which takes 1/2 Kb from the RM's total addressable space of 2 Kb. A 32 Kb static RAM chip was added on the board, in order to accommodate new configurations, new software kernels and user programs for the microprocessor. At configuration time, the EPROM and RAM are both connected in a "Parallel Master Low" mode to the 4010 device. After power-on/RESET or a high-to-low transition on the DONE/PROG pin, the 4010 device loads the microprocessor configuration from EPROM and starts the kernel.

Reconfiguring the system requires a new configuration and a new kernel to be downloaded from a host computer into the 32 Kb RAM. The new configuration can be loaded at any moment from the external RAM into the internal memory of the 4010 device, creating another version of the microprocessor with different instructions and functions. The kernel contains a minimum of commands: Memory Modify, Execute, Download, Reconfigure and Download Configuration.

The process of reconfiguration is started through the Reconfigure kernel command, which changes the polarity of one of the microprocessor's outputs connected to the DONE/PROG pin of the 4010 device. In this way, the microprocessor reconfigures itself, allowing for easy verification of new architectures.

A second microprocessor configuration and a second cross-assembler and kernel were developed in order to test the reconfigurability of the microprocessor. The Download Configuration command was used to load the new configuration and kernel into the 32 Kb external RAM. The reconfiguration of the 4010 device was started with Reconfigure command, and the new microprocessor was tested immediately and successfully.

Summary

A reconfigurable microprocessor can have a tremendous impact on remote control applications where, not only changes in software but also in hardware, increase flexibility and reliability. Further studies will definitely uncover many new areas of research, development and applications for the reconfigurable microprocessor.

References

1. D.E. Van Den Bout, "AnyBoard: An FPGA Based Reconfigurable System", IEEE Design & Test of Computers, Sept. 1992, pp. 21-30.

2. K. S. Perianayagam, "FPGA Implementation of the BH8000 Wormhole Router", Fourth Annual IEEE International ASIC Conference, Sept. 23-27, 1991, pp. p16/3.1-p16/3.4.

3. Yee-Lu Zhaog & all, "A color video camera using FPGA video processor", Fourth Annual IEEE International ASIC Conference, Sept. 23-27, 1991, pp. p16/7.1-p16/7.4.

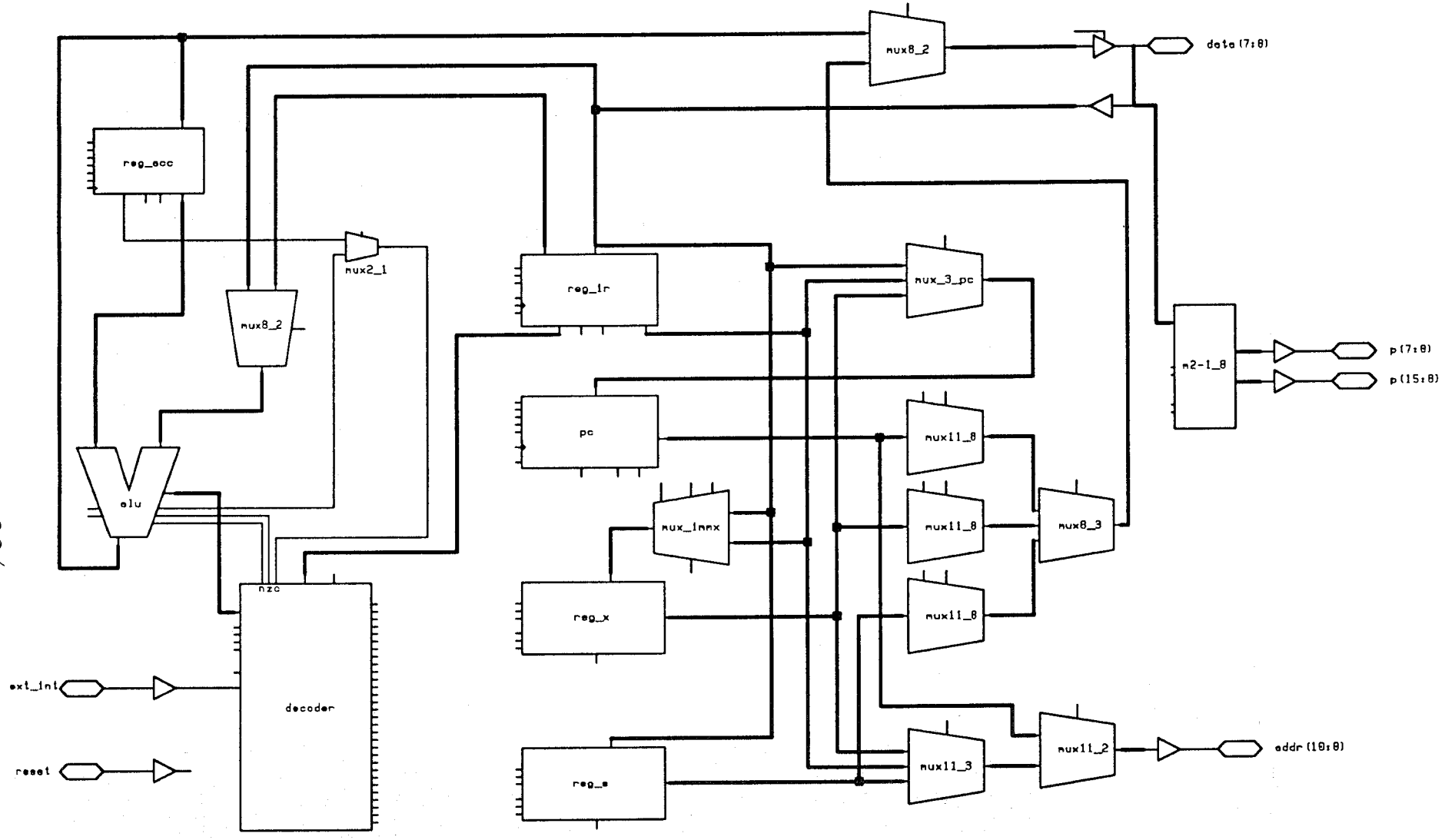
4. R. B. Brown &all, "A Microprocessor Design Project in an Introductory VLSI course", Microelectronic System Education Conference, July 22-25, 1991, pp. 195-205.

5. H. Kanbara, "KUE-CHIP: A Microprocessor for Education of Computer Architecture and LSI Design", The Third Annual IEEE ASIC Seminar, Sept. 17-21, 1990, pp. p10/4.1-p10/4.4.

Instructions set

1.	swi	software interrupt
2.	rti	return from interrupt (software/hardware)
3.	rts	return from subroutine
4.	clr	clear reg_a
5.	com	complement reg_a
6.	pul	pull reg_a
7.	psh	push reg_a
8.	dec	decrement reg_a
9.	inc	increment reg_a
10.	lsl	logical shift left reg_a
11.	lsr	logical shift right reg_a
12.	cli	clear i (unmask hardware interrupt)
13.	sei	set i (mask hardware interrupt)
14.	dex	decrement x
15.	inx	increment x
16.	des	decrement s
17.	ins	increment s
18.	jmp x	indexed jump
19.	lda -x	load reg_a with pre-decrement x
20.	lda x+	load reg_a with post-increment x
21.	sta -x	store reg_a with pre-decrement x
22.	sta x+	store reg_a with post-increment x
23.	ldx #xval	load x immediate
24.	lda #val	load reg_a immediate
25.	cmp #val	compare reg_a with immediate value
26.	sub addr	substract from reg_a
27.	ora addr	logic or with reg_a
28.	add addr	addition with reg_a
29.	and addr	logic and with reg_a
30.	eor addr	logic eor with reg_a
31.	lda addr	load reg_a from address
32.	sta addr	store reg_a at address
33.	jmp addr	jump address
34.	jsr addr	jump subroutine
35.	ldx addr	load x
36.	stx addr	store x
37.	lds addr	load s
38.	sts addr	store s
39.	beq addr	branch if zero
40.	bmi addr	branch if minus
41.	bcs addr	branch if carry set

3.2.4



Microprocessor - architecture

Fig. 1