CABARE: AN EDUCATIONAL RECONFIGURABLE GENERAL PURPOSE PROCESSOR

Tadeu Ferreira Oliveira, Ivan Saraiva Silva

Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte, Natal, Brasil
tadeu@lasic.ufrn.br, ivan@dimap.ufrn.br

ABSTRACT

Reconfigurable architectures combine the flexibility of general purpose processors and the efficiency of ASIC (Application Specific Integrated Circuit), this paper first discuss the current reconfiguration related works then it describes a general purpose processor that uses control unit reconfiguration to achieve flexibility and efficiency. The architecture was completely prototyped and tested on FPGA (Field Programmable Gate Array) along with a memory module and an assembler tool that generates code to the processor, results show that there are minimum area requirements and the architecture may expand to add new instructions or to be used as part of another SoC.

1. INTRODUCTION

The systems using general purpose processors has clearly presented as the standard on desktop computers specially for its flexibility of application, on the other hand embedded devices industry has developed ASIC as a low energy highly efficiency platform since those ASIC can perform specific tasks with minimum resource use. The reconfigurable architecture offers the advantage of using a specialized instruction set, as in an ASIC architecture, and the flexibility of general purpose processors.

Cabare (Computer Architecture BAsed on REconfiguration) uses a control unit based reconfiguration, so the developer may create his own instructions based on control of the signals inside the processor. Among reconfigurable processor classes, the RISP (Reconfigurable Instruction Set Processors) began, recently, to be more explored, due it allows dynamic reconfiguration of the instructions set as the instruction pattern (format, operands, etc.).

The RISP processors can be used in multimedia applications, digital signal processing, cryptography, etc. Besides that, in consequence of its particular features, related to internal logic and change of instructions, they become issue to research and academic works involving computer architectures and microprocessors in computing engineering and computing science graduation courses.

Next section we discuss the current efforts on processor reconfiguration, section 3 describes deeply our approach and the architecture of Cabare section 4 shows results of prototyping and test, finally section 5 shows our conclusions and future works.

2. RELATED WORKS

During last years many Reconfigurable processor were developed and evolved with the available technology, trying to solve new challenges on industrial and academic areas.

The first called reconfigurable processor was PRISC (Programmable Instruction Set Computer), in 1994 in [8]. PRISC was the first to use a RFU on a MIPS datapath. Reconfiguration based processor development leaded to the creation of DISC (Dynamic Instruction Set Computer) by [9] using partial reconfigurable FPGAs. A Hybrid architecture called GARP developed by [6] using a FPGA as a slave functional unit placed on the same location of the processor supporting multicycle operations. In 2001 [4] created the OneChip architecture, this architecture integrates a reconfigurable unit inside a superscalar RISC pipeline processor. The VISC (Variable Instruction Set Communications Architecture) developed by [7] uses a dictionary to define the type of each instruction allowing the complier to better configure the instruction set for a particular program.

A RISP processor, aimed to validate the proof of concept of [3], was developed by [5] adding a reconfigurable unit to new instructions reconfiguration and a control unit where different instruction sets were implemented, those sets could be chosen using privileged instructions according to applications purpose.

3. CABARE ARCHITECTURE

Cabare is a reconfigurable general purpose processor designed with a few instructions set and capable of interpreting new instructions. The design of Cabare is based on the Altera's didactic processor [1], [2]. Being developed thinking on the Altera DE2 Development Board our processor has all the elements seen on Figure 1 and has a 7 Segment decoder, with a multiplexer attached to it, so the user can choose, using the switches on the board, which value is showed on the set of eight 7-segment displays available on the DE2 Board.

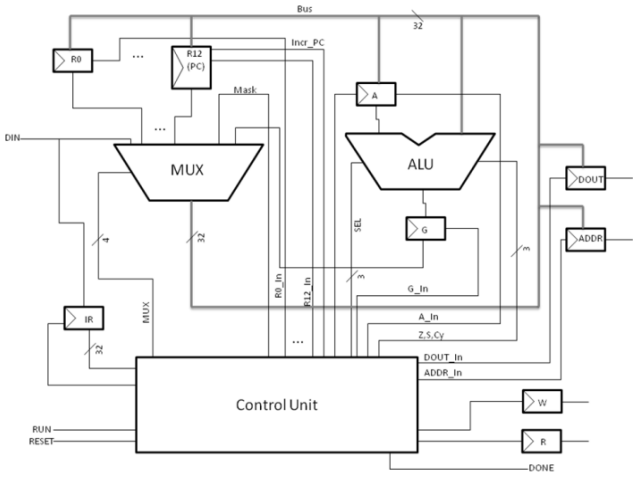

Figure 1 - Cabare Architecture

The processor has 11 general purpose registers (R0 to R11), a program counter register (R12), an ALU with its Accumulator Register (A) and a Register G that holds ALU operations results. Communication with the memory is done using the signals from DIN, DOUT, ADDR, W and R where: DIN (Data In) is the signal through which the processor receives data from memory; DOUT (Data Out) is the registered data output signal to the memory; ADDR (Address) is the registered memory read and write address; the W and R signals are respectively the write and read control signal to the memory. The IR register stores the current instruction being executed on the Control Unit, the final element is a multiplexer to choose what goes on the bus, its inputs are: All the registers (R0 – R12), DIN, G and Mask. Mask is a 32 bits signal, the Control Unit can use it to send constants to the bus. The ALU has 3 flags Z, S and CY to indicate the occurrence of Zero, Signal or Carry conditions on the ALU last operation.

There are two signals that control the start of operation on the processor, those signals are: RUN and RESET each one of those signals is connected to a board's push button. The RESET signal sets the registers to a start value of 0 and the PC (Program Counter) to point to the first instruction. The RUN signal is used to resume the processor after a breakpoint.

Cabare uses a RISP architecture, there are 20 different operations supported by the CPU and 11 reserved Operation Codes for future instructions there are 32 operations code for reconfigurable instructions. This especial kind of operation code is not hardwired defined on the Control Unit they are instead, defined as a set of micro-instructions created by the user (programmer or compiler) and stored on the main memory. An operation code is divided in two parts, the first (3 bits) indicates if this is a common or a reconfigurable instruction. If these bits are all 0 this is a common instruction if they are all 1 this means it is a reconfigurable instruction. The second part is the code that specifies the instruction.

The execution of a reconfigurable instruction can be divided in 3 steps, they are: (i) fetch the reconfiguration table, find the address of the first micro-instruction on the main memory; (ii) fetch the micro-instruction and execute it; (iii) fetch next micro-instructions and execute them until the bit end is set. The reconfiguration table is always located at the address 0xFFFXXXXX where XXXXX are the current operation code five less significant bits.

A reconfigurable instruction is a set of n 32 bit words. Each word describes a micro-instruction. Bit fields on this 32 bits word describe a set of options: ALU operations; BUS selection; Write Enable of each addressable register; memory communication and so on.

Using the reconfiguration function is possible to create conditional and unconditional jumps inside the states of a reconfigurable instruction, this is done by the control unit using the first 4 bits field to create a jump micro-instruction, this jump may be related to the state of the last ALU operation such as zero, carry or negative results, allowing the programmer to create complex new reconfigurable instructions. The 4 bits are interpreted by the control unit as Table 1 shows:

Table 1 - Jump Microinstruction

4 First Bits	Action
0000	No Jump
0001	Jump if ALU flag zero = 0
0010	Jump if ALU flag zero = 1
0011	Jump if ALU flag signal = 0
0100	Jump if ALU flag signal = 1
0101	Jump if ALU flag carry = 0
0110	Jump if ALU flag carry = 1
0111	Always jump

If the 4 first bits are different from 0000 the others 28 bits are interpreted as the memory address to jump for the next micro-instruction, this enables a greater level of reconfiguration to the processor.

On a reconfiguration word there is a 3-bits field that specifies the ALU operation. The operations currently supported are shown on Table 2.

Table 2 - ALU Operations

Signal	Operation
000	ADD
001	SUB
010	AND
011	OR
100	NOT A
101	Transparency
110	Shift Right
111	Shift Left

The bits 25 to 12 are called R0In to R12In they write-enable of each register on the processor. The MUX bits are the selection of the multiplexer, used as shown in Table 3

Table 3 - Multiplexer Options

MUX	Multiplex Value Out
0000	R0
0001	R1
...	...
1100	R12
1101	Mask
1110	G
1111	DIN

The IR, A, G, Dout and ADDR bits, on a reconfiguration word, are write-enable for each of this registers. The WD and RD enable writing and reading to the memory respectively.

The processor also has a special instruction BRK to set a breakpoint; this instruction specifies a memory address where a breakpoint should occur when the processor reaches the instruction on this address it pauses waiting for a key to be pressed to resume the execution. This instruction is especially useful on debugging applications written to Cabare since it allows the developer to pause the execution and to see the contents of every register on the register bank and the registers A and G using the board's 7-segment display.

From the set of instructions an assembly language and an assembler tool were developed. The assembler generates the .mif (memory initialization file) which is used by the synthesis tool to create the first data resident on memory. The current language supports: comments, every instruction, labels, and data initialization. Through data initialization it is possible the definition of reconfigurable instructions. The definition of a reconfigurable instruction is made setting the micro-instructions code on a memory location (or on several memory words).

Since Cabare reads its instructions from a memory module, initially all programming was done writing a mif (memory initialization file) with every instruction to be executed. This kind of programming is too hard, time consuming and error prone, even the debugging and correction of a simple application may imply on hours of code writing. To minimize this problem an assembler was developed, so the programmer can use common low level program techniques to create programs such as labels.

The assembler tool allows the programmer to write using mnemonics which ease the development and understanding of programs to Cabare processors, currently this assembler can produce code using all cabare instruction set and provides means to the programmer to describe the reconfiguration instructions, for now each micro-instruction, of a reconfigurable instruction, still needs to be described as a 32 bits hexadecimal value.

Using the assembler tool the programmer can write code and describe it own reconfigurable instructions the definition of a reconfigurable instruction is made using a special section on assembler code called .data, this way the programmer declares a data region and put it a name so he can call this new instruction using the & operator as shown on the example below:

```
.data reconf01 #0A000140
.data reconf02 #00000820
.data reconf03 #0A800E00
.data reconf04 #0A000140
.data reconf05 #00000020
.data reconf06 #0A800E01

mvi r1,#5
mvi r8,#4
mvi r0,#2
&reconf01
halt
```

3.1 Memory Operations

The memory was implemented as a RAM memory using Altera's megafunction, the memory is organized as 8192 words of 32 bits each. The connection scheme of this component is shown on Figure 2.

From Figure 2 one can see the read address and write address are the same. The processor works activating the READ or the WRITE signals on the rising edge of the clock, on a write operation the data to be written on the memory is sent by the DOUT signal on a read operation the DIN is connected to the processor to send the data.

There are two types of memory operations on the Cabare processor the LOAD, STORE operations, so the communication with the memory module is very simple as shown on Figure 2

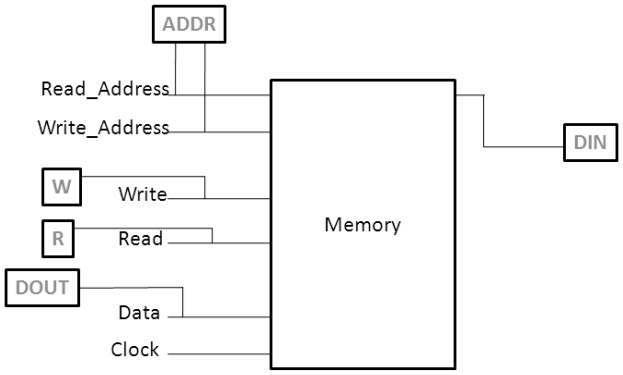

Figure 2 - Memory module

The ADDR, W, R and DIN are signals from the Cabare processor. A LOAD operation is triggered by the R signal as the STORE operation is triggered by the W signal.

4. RESULTS

The elements on this processor were prototyped and tested on a FPGA using the Altera DE2 Development Board with EP2C35F672C6 from Cyclone II family. The complete architecture that includes the processor, a 7-segment decoder and the memory used 1460 Logic Cells meaning 4% of Cyclone's FPGA. The max Frequency our design may run is 50.63 MHz.

The distribution of logic cells can be seen on Table 4 the MUX element act as a bus on controlled by the control unit as can be seen the area is dominated by the register bank which occupies 29% of our architecture.

Table 4 - Distribution of Logic Cells

Element	Logic Cells (% of full design)
Control Unit	222 (15%)
ALU	218 (15%)
Memory	35 (2%)
Registers Bank	423 (29%)
Mux	360 (25%)
7-segment decoder	56 (3%)

Sample applications were developed and tested using Cabare and the assembler tool, those applications included integer square root calculation and matrix multiplication.

5. CONCLUSIONS AND FUTURE WORK

As an educational processor designed and prototyped on FPGA the CABARE processor not only showed as a complete example of a reconfigurable processor, but also as a good platform for developing applications that uses the paradigm of reconfigurable instruction set.

The area used by the full design is small enough (only 4% of Cyclone II FPGA) to use more than one processor as a processing element on an MPSoC. An MPSoC with Cabare using a Network on Chip as interconnection mechanism is being developed which can provide a tool to develop and to analyze parallel applications development.

There is still space for new standard instructions which can be implemented on the processor datapath as well as may be implemented on the ALU new operations improve the processor capability to accept more general purpose applications. New modules to interfaces with other DE2 peripherals as the LCD display and others are being developed.

6. REFERENCES

[1] ALTERA. Altera Tutorials and Lab Exercises. Available at: http://www.altera.com/education/univ/materials/manual/unv-lab-manual.html. Accessed: sep. 2008.

[2] ALTERA. DE2 Development and Education Board. Available at: http://www.altera.com/education/univ/materials/boards/unv-de2-board.html. Accessed: sep. 2008.

[3] BARAT, F.; LAUWEREINS, R. Reconfigurable Instruction Set Processors: A Survey. In: RSP '00: Proceedings of the 11th IEEE International Workshop on Rapid System Prototyping (RSP 2000). Washington, DC, USA: IEEE Computer Society, 2000. p. 168. ISBN 0-7695-0668-2.

[4] CARRILLO, J. E.; CHOW, P. The effect of reconfigurable units in superscalar processors. In: FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays. New York, NY, USA: ACM Press, 2001. p. 141–150. ISBN 1-58113-341-3.

[5] CASILLO. L. A. Design and Implementation in FPGA of a processor with reconfigurable instruction set using VHDL. Master. Informatics and Applied Mathematics Department. Federal University of Rio Grande do Norte, Natal, 2005, 143 p.

[6] HAUSER, J. R.;WAWRZYNEK, J. Garp: a MIPS processor with a reconfigurable coprocessor. In: FCCM '97: Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM '97). Washington, DC, USA: IEEE Computer Society, 1997. p. 12. ISBN 0-8186-8159-4.

[7] LIU, J. et al. Variable Instruction Set Architecture and Its Compiler Support. IEEE Trans. Comput., IEEE Computer Society, Washington, DC, USA, v. 52, n. 7, p. 881–895, 2003. ISSN 0018-9340.

[8] RAZDAN, R.; SMITH, M. D. A high-performance microarchitecture with hardware programmable functional units. In: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture. New York, NY, USA: ACM Press, 1994. p. 172–180. ISBN 0-89791-707-3.

[9] WIRTHLIN, M. J. A dynamic instruction set computer. In: FCCM '95: Proceedings of the IEEE Symposium on FPGA's for Custom Computing Machines. Washington, DC, USA: IEEE Computer Society, 1995. p. 99. ISBN 0-8186-7086-X.