ECOMIPS: An Economic MIPS CPU Design on FPGA

Xizhi Li
The CKC honored School of Zhejiang University, P.R. China
LiXizhi@zju.edu.cn

Tiecai Li
Department of Electrical Engineering
Harbin Institute of Technology
litc611@public.hr.hl.cn

Abstract

In recently years, many traditional ASIC applications are moving towards a more flexible FPGA based design. To establish a complete application system, it often needs a separate processor to achieve some interactive system functionalities such as I/O operations and control-sensitive tasks. Modern chip producers (Xilinx, Altera, etc) are promoting economic, yet powerful FPGA chips that have the capacity of migrating a general DSP or Microprocessor into one FPGA chip. The trend is that both a general CPU module and the application specific circuit are to coexist on a single chip. This article introduces a MIPs CPU architecture or ECOMIPS to be implemented for that purpose. It focuses on economic resource utilization on modern chips (Xilinx Spartan 3 families). Traditional MIPs architecture was modified to avoid resource conflicts with the ASIC part. The key principles of designing such MIPs HDL IP cores are covered and analyzed with implementation results. As a second objective, ECOMIPS also tries to make itself a customizable and reusable architecture for bridging the gap between microprocessor and ASIC.

1. Introduction

In recently years, many traditional ASIC applications are moving towards a more flexible FPGA based design. To establish a complete application system [3], it often needs a separate co-processor to achieve some interactive system functionalities such as I/O operations and control-sensitive tasks. Modern chip producers (Xilinx Corp, Altera, etc) are promoting economic, yet powerful FPGA chips that have the capacity of migrating a general DSP or Microprocessor into one FPGA chip. The trend is that both a general CPU module and the application specific circuit are to coexist on a single chip. The integrated chip will not only reduce the cost of the product, but also increase the system's overall speed. What's more, by embedding a fast CPU module in the FPGA, it will open up new possibilities of redistributing processing tasks between the CPU and the original ASIC, hence reduce the complexity of the system and ease the design of both.

Yet, a few of obstacles must be resolved. Firstly, FPGA has limited chip resources and a general purpose CPU module will use up too many of them. Secondly, even we manage to accommodate the two units in one chip, if without carefully designed architecture, it will be difficult to reuse the CPU module in other applications or in the improved version of the same system. Thirdly, commercial CPU IP cores rarely meet the system's design specification in its original form and is usually too well encapsulated to be modified freely. Moreover it is usually less customizable than a self-designed CPU core.

This article introduces MIPs CPU architecture or ECOMIPS to be implemented on the same chip as application specific FPGA. It focuses on economic resource utilization on modern chips (Xilinx Spartan 3 families [1]). Traditional MIPs architecture was modified to avoid resource conflicts with the ASIC part. The key principles of designing such MIPs IP cores are covered and analyzed with implementation results. As a second objective, ECOMIPS also tries to make itself a customizable and reusable framework [4, 5, 7, 8] to be used for various applications. Throughout section 2, we give a few hints on how to augment current ECOMIPS into a reconfigurable architecture; another literature will focus on reconfiguring ECOMIPS.

2. ECOMIPS Architecture

ECOMIPS is a compact 32-bit MIPs CPU module to be embedded in a FPGA chip. By taking advantages of modern FPGA chip, the system itself consumes very little chip resources and leaves abundant room for implementing other specialized control and processing modules. A set of software tools have been developed to facilitate the process of redesigning the MIPs processor, so that it would be easy to reuse the CPU core for various applications.

Modern FPGA chips like the Xilinx Spartan 3 families provide many kinds of specialized resources as well as ever increasing gate numbers. Table 1 provides a list of selected features on Spartan 3 chip families [1, 2].

Some of these resources are general purpose ones and used by almost all modules in the chip. However, some resources like multipliers, wide multiplexers, blocked RAM and DCM (with high resolution phase shifting) are not extensively used by most computational modules. When a specialized FPGA chip is at work, many of its dedicated resources are left unused. ECOMIPS takes advantages of those often unused resources and relieves resource racing on other logic cells on the chip.

Table 1. Selected Xilinx Spartan 3 features

- Logic resources
  - Abundant logic cells with shift register capability
  - Wide multiplexers
  - Fast look-ahead carry logic
- SelectRAM™ hierarchical memory
  - Up to 1,872 Kbits of total block RAM
  - Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
  - Frequency synthesis
  - High resolution phase shifting

2.1. Microprogram controller

In CPU, the two modules that consume most gate resources are Arithmetic Logic Unit (ALU) and sequencing control unit (or microprogram controller). For speed concerns, most CPU systems derive sequencing control unit from well designed finite state machines. Another approach is to code all sequencing logic into micro-program and store it into a RAM/ROM block. Both designs have their own advantages over the other. ECOMIPS uses the RAM based microprogram approach for the following reasons: (1) blocked RAM is a rich resource in modern FPGA chips. (2) It is much easier to design new CPU functions such as adding application specific instruction set and custom datapath. (3) The complexity of the sequencing circuits does not increase with the complexity of sequencing logic. (4) By turning sequencing logic into software (microprogram), it's much easier to do a kernel context-switch [9] or update sequencing logic at both design time and run-time.

Microprogram storage unit is implemented as a Blocked RAM primitive in ECOMIPS. If the micro-instruction width is 18bits, a single blocked RAM is able to accommodate 256 micro-instructions on a Xilinx Spartan 3 chip. In most cases, one blocked RAM will be far sufficient for storing the entire microprogram. So the sacrifice of one blocked RAM can replace all the sequencing logic which would otherwise use up a large amount of gate resources in FPGA. Table 2 compares the resource consumption of 32 bits ECOMIPS controller with an 8 bits 8031 controller. The synthesizer used for both HDL modules is Xilinx XST.

We see that sequencing circuits eat up over 50% of resources in a general CPU like 8 bits 8031 core. However, with the controller logic moved to a blocked RAM, ECOMIPS' microprogram controller almost does not need any logic cell resource at all.

Table 2. Controller resource compare

Resource \ module	32 bits ECOMIPS controller	8 bits 8031 controller	8 bits 8031 core
Number of Slices:	10	154	250
Number of Slice Flip Flops:	8	40	129
Number of 4 input LUTs:	17	287	468

ECOMIPS further reduces gate complexities in microprogram controller by using a single dispatch scheme. Every MIP instruction is dispatched immediately after the instruction fetch phase is finished. Dispatch is done directly using the 6 bits Opcode from the instruction; therefore no separate dispatch ROM is needed. In other words, the micro-instructions in the RAM is indexed with Opcode and every instruction is dispatched only once (see Figure 1).

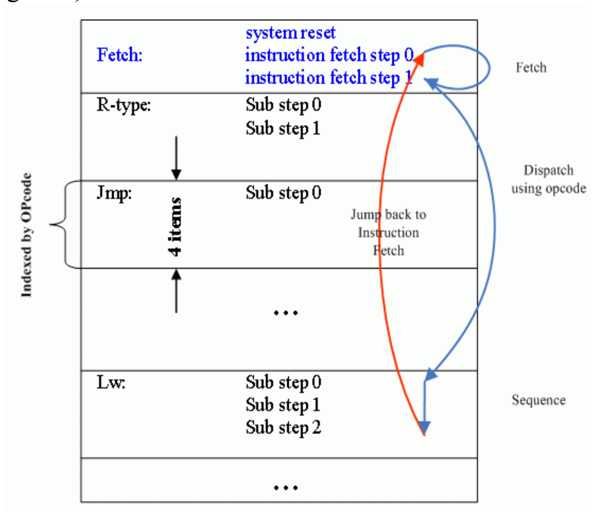


Figure 1. Microprogram execution path

The microcode in ECOMIPS maps directly to control signals in the datapath. Therefore, no translation circuit is needed.

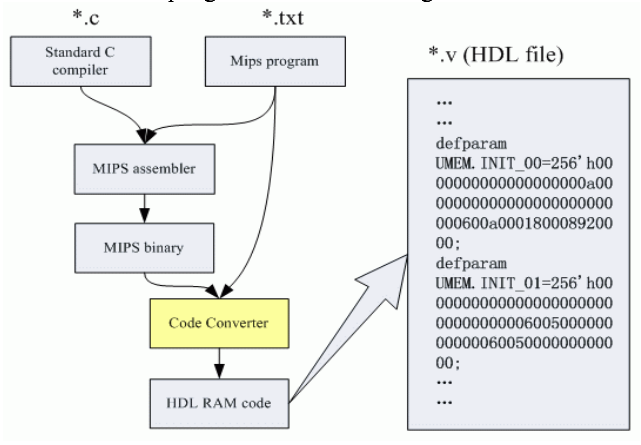
However, the speed of reading one word from a blocked RAM is not as fast as that from a combinatorial circuit and the maximum system clock rate of the ECOMIPS is almost linearly dependent on the speed of blocked RAM in the FPGA. Fortunately the speed of RAM access has been made very fast in modern FPGA chips, currently in less than 15 nanoseconds. If the clocking scheme is properly designed, the maximum CPU clock rate can still be as high as 64MHZ.

2.2. Register files

ECOMIPS has 32 32-bit registers that are directly accessible in its instruction. Again to save flip-flop resources on FPGA, the ECOMIPS uses another blocked RAM for storing all 32 registers. Xilinx Spartan 3 chip supports dual-port synchronous access [1] to its blocked RAM which is the ideal solution to register file in MIPs architecture.

2.3. Memory

The time to read one instruction from the memory directly affects the overall speed of the CPU. So memory must be kept close to the CPU. In ECOMIPS, it is implemented as a bunch of blocked RAMs. The speed to fetch one instruction is as fast as accessing register file or microprogram. However, the number of blocked RAM varies dramatically in different FPGA chips and so does the cost. Hence, the size of in-chip memory can not be very large. An average Xilinx Spartan3 400 chip can hold 9000 MIP instructions or 32-bit data words in its memory at most. In some applications, an external storage (a ROM for instance) must be used to accommodate large programs. In such cases, explicit MIPS I/O commands should be called to swap data in and out of the memory. Yet, it's up to the software program (a simple operating system or swap algorithm [6, 8]) to do the trick. Whether a system has external storage or not, the first program that ECOMIPS executes is always in RAM memory. Currently, FPGA RAM content can only be initialized by setting its initialization parameters in its HDL files. Those lines of HDL codes are even more difficult to understand than binary MIPS code. So currently, ECOMIPS uses a code converter program as shown in Figure 2.



```
...
...
defparam
UMEM.INIT_00=256'h00
00000000000000000a00
00000000000000000000
000600a0001800089200
00;
defparam
UMEM.INIT_01=256'h00
00000000000000000000
00000000006005000000
00000060050000000000
00;
...
...
```

Figure 2. Writing MIPS program

It helps ECOMIPS developers to convert binary MIPS codes or MIPS instructions into valid HDL RAM initialization strings.

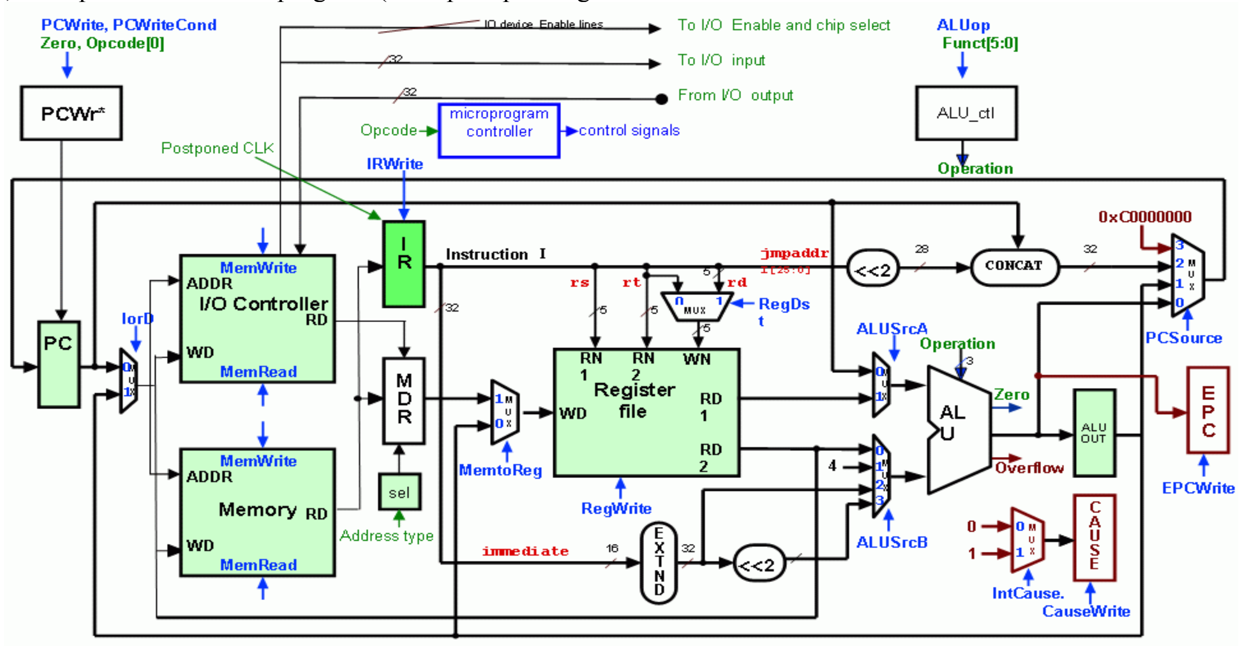


Figure 3. ECOMIPS datapath

2.4. Memory mapped I/O

An efficient I/O management scheme is the memory mapped I/O. In ECOMIPS, I/O polling modules share the data and address bus with the memory module. The high unused bits in the address input are used to decide whether the data is in the memory or in an I/O device. If it's the latter, the corresponding I/O module is enabled to access the data and returns the result through the standard memory data bus. There is no central arbitrator for bus usage; each I/O device (including the memory) is responsible for handling mutual exclusive use for the memory bus lines. This distributed scheme is more flexible. The address space of ECOMIPS is 4GB which is too large for embedded CPU. Therefore, we are free to use memory-mapping techniques for interfacing the CPU with the other application specific circuits in the FPGA; it's even possible to do a full system context-switch [9] by simply executing programs in different address space of the CPU. This simple approach of ECOMIPS has been used in a High-performance Servo Controller for PMSM [3] for its position feedback and user interface control.

2.5. Datapath and clocking

ECOMIPS uses a modified version of a standard MIPS multi-cycle datapath as shown in Figure 3.

Blocked RAM in Xilinx Spartan 3 chip saves output in registers; the output from these registers should be used directly to drive combinatorial circuits instead of connecting to explicit cache registers in the datapath. By taking advantage of this feature, three 32 bits registers can be removed from the datapath.

A general CPU core has many synchronous units. It is both difficult and inefficient to associate all units with a single global clock signal. What's more, the speed of these units differs greatly. For example, the speed of a blocked RAM and a simple flip-flop differ in several nanoseconds. So it's usually more desirable to group synchronous units according to their speed and allocate them with a distinct clock signal.

Xilinx Spartan-3 devices [1, 2] provide precise and complete control over clock phase shift by employing a Delay-Locked Loop (DLL), a fully digital control system that uses feedback to maintain clock signal characteristics with a high degree of precision.

ECOMIPS redesigns the CPU's clocking scheme over the entire circuit and phases the system clock into 4 clock signals. They have the same frequency as the system clock; yet, each is precisely delayed in nanoseconds than the previous one according to their usage. By doing so, the CPU clock rate can still depend solely on the slowest synchronous unit in the datapath, while increasing the simplicity and clarity of the whole CPU datapath.

3. Implementation result

ECOMIPS has been synthesized on a Xilinx Spartan3 400 chip with Xilinx XST. The maximum CPU clock rate is 64MHz. The synthesis report is shown in Table 3.

The customizable parts are memory module and ALU module. The total resource will be significantly cut down, if the size of memory is reduced or ALU data width is decreased.

Table 3. Resource usage

Module \ number	**Slices**	**Slice Flip Flop**	**4 input LUTs**	**BRAMs**
Microprogram controller	10	8	17	1
ALU control	3	0	5	0
Register file	3	0	6	1
I/O controller	10	0	18	0
32 bits memory	35	2	69	4
32 bits ALU	76	0	141	0

4. Conclusion

Modern FPGA chips have provided developers with rich resources for embedding a CPU module in the original application specific circuit. ECOMIPS is designed to take full advantages of chip resources. It explores how a general reconfigurable MIPS design can be implemented on modern chip families (Xilinx Spartan3) with minimum resource consumption. We have included many design details and principles in this literature, hoping to provide a useful guide for implementing integrated FPGA design in real applications.

References

[1] Xilinx Corp, Spartan 3 data sheet, 2003
[2] Xilinx Corp, Xilinx ISE 6.1i Lib Guide, 2003
[3] Zhao-yong Zhou and Li Tiecai, FPGA Realization of a High-performance Servo Controller for PMSM, APEC04, 2004
[4] John Reid Hauser, Augmenting a Microprocessor with Reconfigurable Hardware, PHD thesis. UC Berkeley, 2000.
[5] John R. Hauser and JohnWawrzynek, Garp: A MIPS Processor with a Reconfigurable Coprocessor, FCCM '97.
[6] M. Gschwind and D.Maurer, An extendable MIPS-I processor kernel in VHDL for hardware/software co-design, Proceedings of the conference on European design automation, 1996.
[7] R. Hartenstein, A decade of reconfigurable computing : a visionary retrospective, Proceedings of the conference on Design, automation and test in Europe (March 2001).
[8] Jeffrey A. Jacob and Paul Chow, Memory Interfacing and Instruction Specification for Reconfigurable Processors, international symposium on Field programmable gate arrays
[9] Kiran Puttegowda, Context Switching Strategies in a Run-Time Reconfigurable system, PHD thesis: Virginia Polytechnic Institute and State University, 2002.