FPGA based Design of a Low Power Asynchronous MIPS Processor

Jukiya Furushima
University of Aizu
Aizu-Wakamatsu 965-8580, Japan
m5201118@u-aizu.ac.jp

Hiroshi Saito
University of Aizu
Aizu-Wakamatsu 965-8580, Japan
hiroshis@u-aizu.ac.jp

ABSTRACT

In this paper, we design a low power asynchronous MIPS processor on a Field Programmable Gate Array (FPGA). We synthesize the MIPS processor by assigning the maximum delay constraints for control paths and data paths. In the experiment, we evaluate the area, execution time, power consumption, and energy consumption of the designed MIPS processor comparing with the synchronous MIPS processor. We achieved 35% power reduction by the designed MIPS processor.

Categories and Subject Descriptors

B.6.0 [Hardware]: LOGIC DESIGN—General

General Terms

Design

Keywords

Asynchronous circuits, FPGA, processor design

1. INTRODUCTION

Most of currently used processors are synchronous circuits. Circuit components in synchronous circuits are controlled by global clock signals. In synchronous circuits, clock skew, power consumption, and electromagnetic radiation will be significant problems when semiconductor submicron technology is advanced more and more.

In asynchronous circuits, circuit components are controlled by local handshake signals. Asynchronous circuits are potentially low power consumption and low electromagnetic radiation due to the absence of global clock signals. However, the design of asynchronous circuits is more difficult than the design of synchronous circuits.

Several asynchronous processors have been proposed. Handshake Solutions designed a clockless ARM996HS using TiDE design flow [1]. Zhang and Theodoropoulos designed SAMIPS [2] using Balsa [3]. SAMIPS is an asynchronous MIPS processor similar to this paper. Amde et.al, designed an asynchronous DLX processor using Pipefitter [4]. Chang-Jiu et.al, designed an asynchronous 8051 microcontroller using Balsa [5]. Iwasaki designed an asynchronous AVR processor considering a cycle time constraint [6]. The targets of these researches were not FPGA but Application Specific Integrated Circuits (ASICs).

In this paper, we design a low power asynchronous MIPS processor on an FPGA. We synthesize the MIPS processor assigning the maximum delay constraints for data-paths and control paths. In addition, we evaluate the area, execute time, power consumption and energy consumption of the designed MIPS processor.

The organization of this paper is as follows. In section 2, we describe background used in this paper. In section 3, we describe the design of an asynchronous MIPS processor. In section 4, we describe the experimental result. In section 5, we conclude this work.

ICAIT '16, Oct. 6 – 8, 2016, Aizu-Wakamatsu, Japan.

2. BACKGROUND

2.1 Asynchronous Circuits with Bundled-data Implementation

Figure 1 represents an asynchronous processor model with bundled-data implementation. The left side is the control circuit and the right side is the data-path circuit. The data-path circuit consists of Program Counter (PC), Memories (IMEM and DMEM), Instruction Register (IR), Decoder, Register File (RF), ALU, and delay elements hd_k. PC stores the address of the instruction memory. IMEM is a memory to store instructions. DMEM is a memory to store data. IR is a register to store an instruction fetched from IMEM. RF is a collection of registers. Data from DMEM and ALU are written into RF. hd_k are delay elements for registers or memories to guarantee hold constraints of registers. The control circuit consists of control modules $ctrl_i$ ($0 \leqq i \leqq m-1$). A control module $ctrl_i$ consists of a Q-module q_i, delay elements sd_i, bd_i, cd_i, and glue logics. sd_i is used to guarantee setup constraints. bd_i is used to guarantee the timing of a control branch. cd_i is used to guarantee the timing of resetting of control signals. Glue logics are branch decision logics and C-elements [8]. C-elements are synchronization components. The output of C-elements is 0 when all inputs are 0. The output is 1 when all inputs are 1. Otherwise, the output does not change. In the case of pipeline execution, two control modules $ctrl_{i\_1}$ and $ctrl_{i\_2}$ are used to control one pipeline stage.

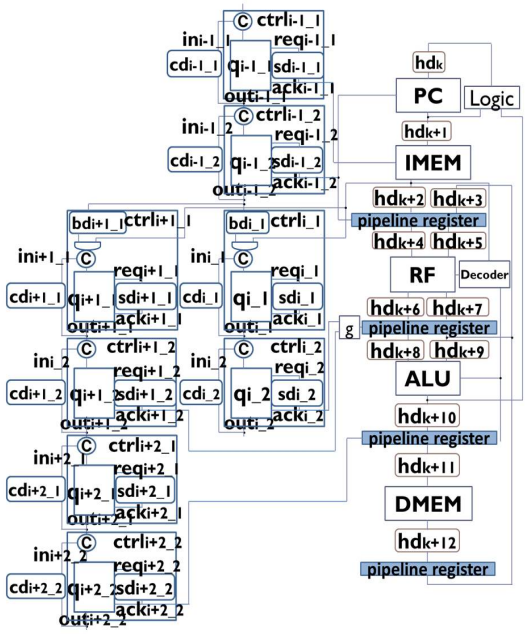

Figure 1: An asynchronous processor model with bundled-data implementation.

We describe behaviors of this processor model. Q-module q_i in $ctrl_i$ starts the control of operations in the data-path circuit when a rising edge in_i comes from the previous Q-module q_{i-1}. After the rising edge in_i signal arrives, q_i rises the request signal req_i, the signal passes sd_i, and rises the acknowledge signal ack_i, and go back to q_i. Then, q_i falls req_i signal and wait a falling edge of ack_i signal. Data are written into registers with the falling edge of ack_i. After ack_i returns q_i, q_i rises out_i and control signal moves to the next control module $ctrl_{i+1}$. In addition, out_i is returned to C-element to reset in_i and out_i.

The model needs to satisfy 5 types of timing constraints [6]. If some of them are violated, we need to adjust delay elements sd_i, hd_k, bd_i, cd_i to satisfy timing constraints.

2.2 The MIPS Processor

The MIPS processor used in this work consists of instruction fetch stage (IF), instruction decode stage (ID), execute stage (EX), memory access (MEM), and write back stage (WB) [9]. The block diagram of the MIPS processor is shown in Figure 2.

IF stage includes an instruction memory (IMEM), program counter (PC), an adder, and two multiplexers to decide the address of IMEM. ID stage includes a decoder, a register file (RF), a sign extension, a shifter, an adder, and a comparator. The decoder generates control signals from a fetched instruction. The register file saves data from a data memory (DMEM) or an arithmetic and logic unit (ALU). EX stage includes an ALU to execute arithmetic or logical operations. MEM stage includes a DMEM. In WB stage, data from ALU or DMEM are written into RF.

The MIPS processor supports 9 instructions. R type instructions (add, sub, or, and, slt) use register values. J instruction is a jump instruction. Beq instruction is a conditional branching instruction. Lw instruction reads data from DMEM and stores to a register. Sw instruction writes data in a register to DMEM.

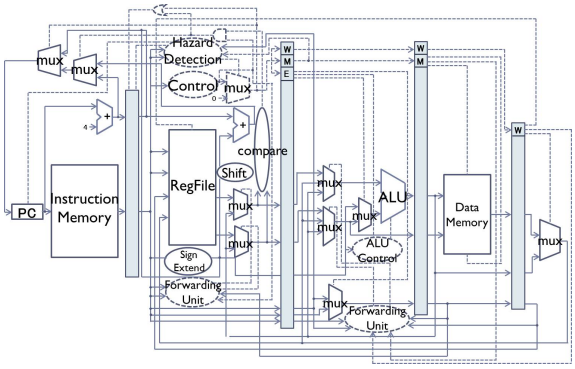

Figure 2: The block diagram of pipelined MIPS processor [9].

2.3 Field Programmable Gate Array

Field Programmable Gate Array (FPGA) is one of reconfigurable devices. FPGA has been used in many embedded systems because of the advantage such as lower design cost and flexibility to change circuit structure. Figure 3 shows the Altera Cyclone IV FPGA. The FPGA consists of Logic Arrays, Embedded Multipliers, Random Access Memories (RAMs), Input/output Elements (IOEs), and Phase Locked Loops (PLLs).

3. DESIGN OF AN ASYNCHRONOUS MIPS PROCESSOR

3.1 Design of an Asynchronous MIPS Processor Model

We design an asynchronous MIPS processor as follows. The data-path circuit is the same as the one used in [9]. The control circuit is modeled by a finite stage machine (FSM). Figure 4 shows an FSM of the MIPS processor. Nodes of the FSM represent pipeline stages. After IF stage, FSM is branched by a fetched instruction. Edges of the FSM represent dependencies between stages.

Figure 5 shows the generation of an asynchronous control circuit. We use the method in [6]. First, each node of FSM is divided into two. This is to hidden the initialization phase of control modules (i.e., reset of in_i and out_i). After division, a control module $ctrl_i$ is allocated to each node. If registers and memories are controlled by some control modules, write signals for them are generated by ack_i signals of the control modules. Finally, C-elements and feedback loops are inserted to control modules. Figure 6 shows the control circuit of the asynchronous MIPS processor.

Figure 7 shows the structure of $ctrl_i$ for Altera Cyclone IV FPGA. Primitives DLATCH and LCELL of Altera FPGAs are used. There are two DLATCHes in two C-elements. If there is no DLATCH in C-elements, timing analysis tool cannot analyze path delays due to the existence of a combinational loop. In addition, to perform timing analysis correctly, we set "synthesis_keep" attribute to the output of sd_i. Moreover, we insert two DUMMY modules to indicate a wire as a through point for path delay analysis. Delay

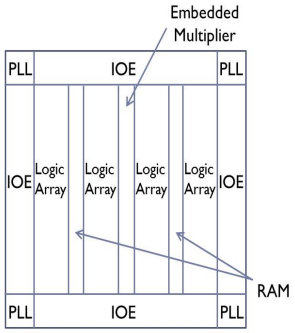

Figure 3: Structure of Altera Cyclone IV FPGA [10].

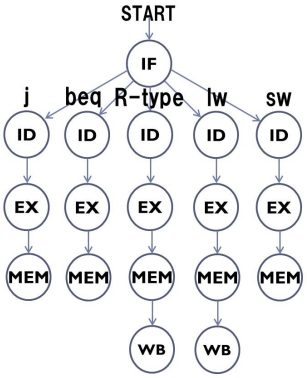

Figure 4: The FSM to represent pipeline stages of instructions.

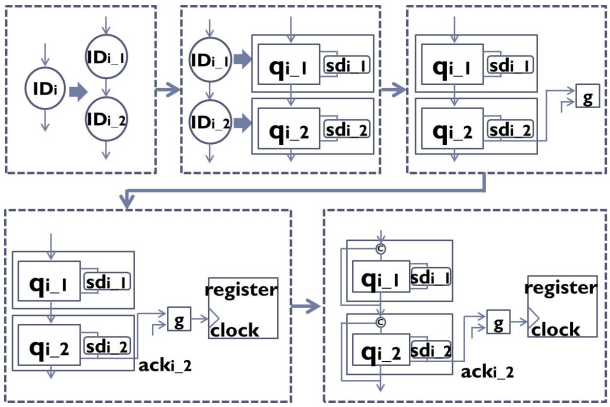

Figure 5: Generation of an asynchronous control circuit [6].

elements sd_i, hd_k, cd_i consists of LCELLs. They work as buffers. To avoid optimization for control modules, we set "Design Partition" for control modules.

Finally, we model the asynchronous MIPS processor using Verilog HDL.

3.2 Generation of the Maximum Delay Constraints and Setting of Margins

3.2.1 *Generation of the Maximum Delay Constraints*

To satisfy a given cycle time constraint CT, we assign the maximum delay constraints to paths related to setup constraints. Figure 8 represents a data-path $sdp_{i,p}$ and a control path $scp_{i,p}$ for a setup constraint. The data-path $sdp_{i,p}$ is divided into 2 sub-paths. The former is from sd_{i-1_2} to source register reg_{src} (PC in Fig. 8) and the latter is from reg_{src} to destination register reg_{dst} (IF/ID pipeline register in Fig. 8). The control path $scp_{i,p}$ is divided into 9 sub-paths: from sd_{i-1_2} to C-element in $ctrl_{i\_1}$, from C-element in $ctrl_{i\_1}$ to sd_{i_1}, from sd_{i_1} to C-element in q_{i_1}, from C-element in q_{i_1} to sd_{i_1}, from sd_{i_1} to C-element in $ctrl_{i\_2}$, from C-element in $ctrl_{i\_2}$ to sd_{i_2}, from sd_{i_2} to C-element in q_{i_2}, from C-element in q_{i_2} to sd_{i_2}, and from sd_{i_2} to reg_{dst}. The reason why we divide paths is to analyze path delays using general static timing analyzers correctly.

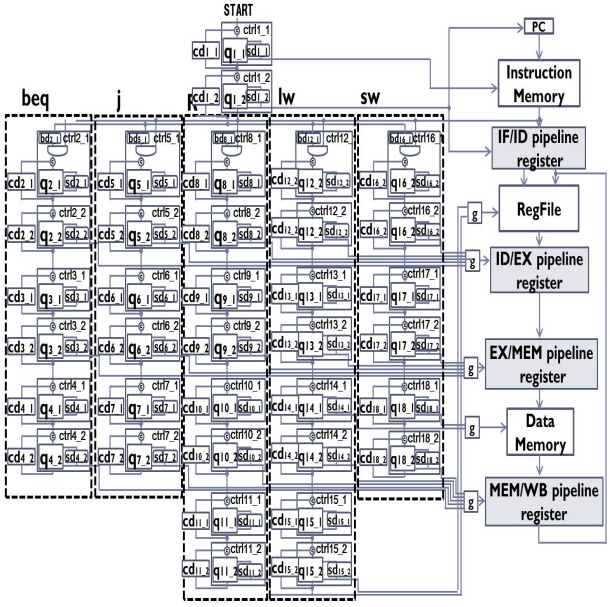

Figure 6: The control circuit of the asynchronous MIPS processor.

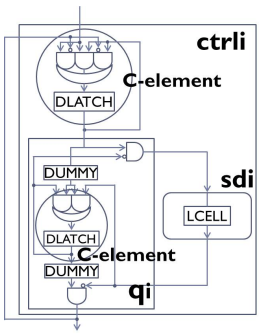

Figure 7: Structure of $ctrl_i$.

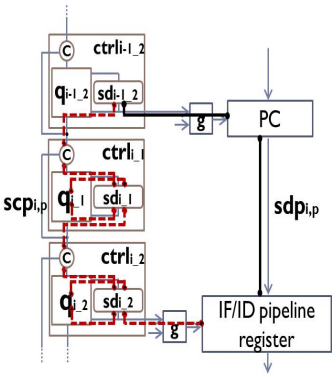

Figure 8: Paths to assign the maximum delay constraints.

We use the calculation method described in [11] to decide the values of the maximum delay constraints. In addition, we use *set_max_delay* command to represent the maximum delay constraint. *set_max_delay* command is provided in Synopsys Design Constraint (SDC) file format.

3.2.2 *Setting of Margins*

We setup margins for data-path delays and control path delays. For the former case, after FPGA implementation, there may be uncertain effects to data-path delays. We decide the margin based on a given cycle time constraint CT and actual path delays on an FPGA. We start from a small value. If there are timing violations during simulation, we increase the margin to solve timing violations.

In bundled-data implementation, from setup constraints, the minimum delay of $scp_{i,p}$ must be larger than the maximum delay of $sdp_{i,p}$ to write data into registers correctly. However, if the minimum delay of $scp_{i,p}$ is large enough for

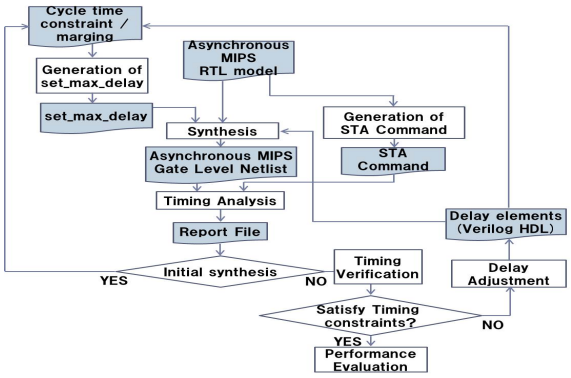

Figure 9: Design flow.

the maximum delay of $sdp_{i,p}$, it may results in performance degradation. Therefore, for the latter case, we decide the margin for control path delays from the delay of LCELLs and the number of delay adjustments to satisfy all timing constraints. We also start from a small value. If the number of delay adjustments for timing violations reported by static timing analysis overs a threshold value (e.g., 10 times), we increase the margin to satisfy timing constraints.

3.3 Design Flow

In this paper, we design an asynchronous MIPS processor based on a design flow shown in Figure 9. The inputs of the flow are a Verilog HDL model of the asynchronous MIPS processor explained in section 3.1, a cycle time constraint CT, and margins for data-path delays and control path delays.

The design flow is classified into the initial synthesis and the incremental synthesis. In the initial synthesis, we generate static timing analysis (STA) commands to analyze path delays related to timing constraints. Then, the initial synthesis and STA are carried out using Altera Quartus II and TimeQuest. From the STA results and cycle time constraint CT, we generate the maximum delay constraints using "set_max_delay" commands as described in section 3.2.

In the incremental synthesis, we repeatedly carry out synthesis and STA using the generated maximum delay constraints. If all of timing constraints are satisfied, we finish design. Otherwise, we carry out delay adjustment to satisfy timing constraints. If the number of delay adjustment overs a threshold value, we increase the margin for the control path delay to meet timing closure.

4. EXPERIMENT

In the experiment, we evaluate the designed asynchronous MIPS processor in terms of area, execution time, dynamic power consumption, and energy consumption comparing with a synchronous counterpart. The used synthesis tool and simulation tool are Altera Quartus II ver.14.1 and ModelSim-Altera ver.10.3.c. TimeQuest timing analyzer in Quartus II is also used to analyze path delays. The target device is Altera Cyclone IV (EP4CE115F29C7).

Initially, we synthesize the synchronous MIPS processor. We explore the best one in terms of clock frequency by

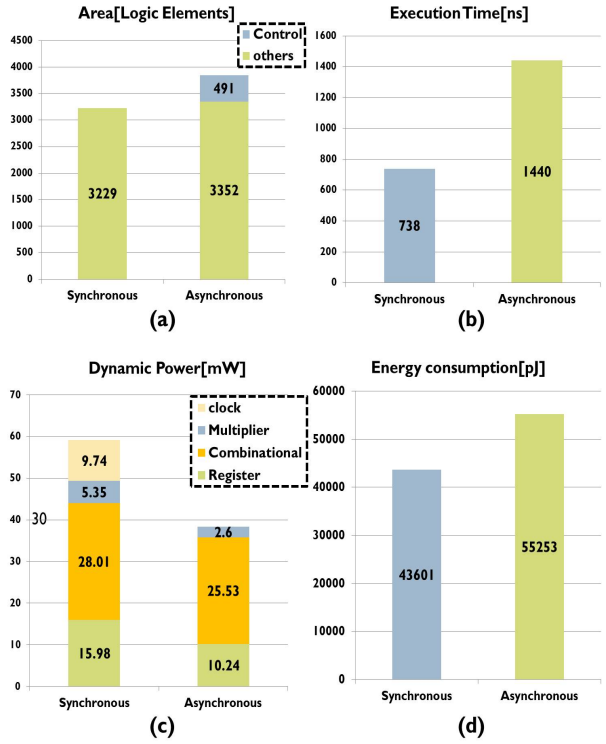

Figure 10: Experimental result: (a) area, (b) execution time, (c) dynamic power consumption, and (d) energy consumption.

changing clock constraint. The cycle time of the best one is 12 ns. We set 12 ns to CT and synthesize the asynchronous MIPS processor. The margin for data-path delays is set to 1.2 ns (10% of CT) and the margin for control path delays is set to 3.6 ns (30% of CT).

Figure 10.(a) shows the area in terms of the number of logic elements reported by Quartus II. The asynchronous MIPS processor requires 19% overhead. This mainly comes from the area of control modules and delay elements. Figure 10.(b) shows the execution time when a multiplication is given as a test input and simulated by using ModelSim-Altera. The asynchronous MIPS processor requires double of the synchronous counterpart. The increase of the execution time in the asynchronous MIPS processor comes from the margin for the control path delay. In fact, we set the margin to 4.8ns. The performance of the asynchronous MIPS processor will be the same as that of the synchronous MIPS processor if we synthesize the asynchronous MIPS processor with the margin of 2.4ns. This is our future work. Figure 10.(c) shows the dynamic power consumption obtained by PowerPlay Power Analyzer inside Quartus II. As we expect, 35% of power is reduced from the synchronous MIPS processor. This is mainly comes from the reduction of power consumption for the clock network and register. Finally, figure 10.(d) shows the energy consumption obtained by the product of execution time and dynamic power consumption. The energy is increased 27% due to the increase of the execution time.

5. CONCLUSIONS

In this paper, we designed a low power asynchronous MIPS processor on an FPGA. The asynchronous MIPS processor was synthesized with the maximum delay constraints for paths related to setup constraints. In the experiment, 35% dynamic power was reduced compared to the synchronous counterpart. However, the execution time and energy consumption were increased due to the insertion of many LCELLs to delay elements.

As future work, we improve the execution time of the designed MIPS processor by restricting the margin for the control path delay. In addition, we are going to reduce power consumption of the forwarding unit by removing unnecessary behaviors.

6. REFERENCES

[1] A. Bink and R. York, "ARM996HS: The First Licensable, Clockless 32-Bit Processor Core", Proc. Micro, pp. 58-68, Mar-Apr. 2007.

[2] Q. Zhang and G. Theodoropoulos, "MODELLING SAMIPS: A Synthesisable Asynchronous MIPS Processor", Proc. 37th Annual Simulation Symposium, pp. 205-212, 2004.

[3] D.Edwards and A. Bardsley, "Balsa : An asynchronous hardware synthesis language", The Computer Journal, 45(1): pp. 12.-18, 2002.

[4] M. Amde, I. Blunno and P. Sotiriou, "Automating the Design of an Asynchronous DLX Microprocessor", Proc. Design Automation Conference, pp. 502-507, 2003.

[5] Chang-Jiu Chen, Wei-Min Cheng, Ruei-Fu Tsai, Hung-Yue Tsai, and Tuan-Chieh Wang, "A Pipelined Asynchronous 8051 Soft-core Implemented with Balsa", Proc. APCCAS, pp. 976-979, 2008.

[6] S. Iwasaki, "Design and Evaluation of a Low Power Asynchronous AVR Processor considering a Cycle Time Constraint", Master Thesis, the University of Aizu, 2014.

[7] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T. P. Fang, "Q-Modules:Internally Clocked Delay Insensitive Modules", IEEE Transaction of Computer, vol. C-37, no.9, pp. 1005-1018, 1988.

[8] D. E. Muller, and W. S. Bartky, "A theory of asynchronous circuits", Proc. International Symposium on the Theory of Switching, pp. 204-243, Apr 1959.

[9] D. A. Patterson and J. L. Hennessy, "Computer Organization and Design, Fifth Edition: The Hardware/Software Interface", Morgan Kaufmann, 2006.

[10] Altera Cyclone IV FPGA, "https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/features.html"

[11] K. Takizawa, et al., "A Design Support Tool Set for Asynchronous Circuits with Bundled-data Implementation on FPGAs", Proc. IEEE 24th International Conference on Field Programmable Logic and Applications (FPL), pp.1-4, September 2014.