Rapid Prototyping Asynchronous Processor

Puah W.B., Suparjo B.S., Wagiran R., and Sidek R.

Department of Electrical and Electronic Engineering, Faculty of Engineering,
Universiti Putra Malaysia, 43400 UPM Serdang, Selangor.

Abstract

An asynchronous processor has been an attractive research fields since it offers many advantages over synchronous processor. Field-programmable gate arrays (FPGA), one of today's dominate media for prototyping and implementing digital circuits is used to construct an 8-bit asynchronous RISC processor. The asynchronous processor employs conceptual framework of a Sutherland micropipeline, a modular approach to design asynchronous circuits.

1. Introduction

Almost all high performance processors today are based on synchronous design framework. As systems grow increasingly large and complex, the synchronous design method faces some serious problems like clock-skew, worst case performance, high power dissipation and high noise emission and electro-magnetic interference (EMI)[1][2]. Asynchronous logic, which works without global clock, can offer an alternative design platform to overcome these problems.

Many academic research groups have been established to exploit the benefits of the asynchronous circuits by designing asynchronous processors. The most successful projects are Amulet and TITAC processors [3][4]. These processors are developed using full custom design, which need a lot of time and effort.

With the rapid development of Field Programmable Gate Array (FPGA) technology, designing asynchronous circuits like processor on FPGA become possible. The 8-bit asynchronous processor is developed using mixed Very High Speed Hardware Description Language (VHDL) and schematic editor in Altera Maxplus-II software environment. The processor employs two-phase transition signaling which functions within the Micropipeline methodology. Because all of the current FPGA technology is basically based on synchronous framework, designing asynchronous circuits are rather difficult than synchronous counterparts since the design methodology is different. Designer must ensure that the synthesized circuits work within asynchronous methodologies environment and avoid hazards occur in the resulting circuits.

2. Micropipelines

All asynchronous design methodologies are based on the delay model. They are bounded-delay, micropipelines, delay-insensitive, quasi-delay insensitive and speed-independent methodologies [2].

Ivan Sutherland has introduced a novel design framework to design asynchronous (self-timed) circuit called Micropipelines in his Turing Award lecture [5]. Micropipelines are an event-driven elastic pipeline, which employ a two-phase handshaking protocol with bundled data for sending data between processing units as illustrated in Figure 2.1.

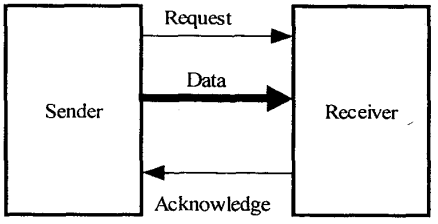

Figure 2.1: Two-phase bundled data protocol

In this handshaking protocol, rising and falling transitions are trigger events for request and acknowledge wires. This may offer speed potential over conventional clocking scheme and saves the time and energy costs compared to return to zero signaling.

When the data become valid on a bus, the sender will issue a *Request* event to inform the receiver about the availability of the data.

0-7803-6430-9/00/$10.00 ©2000 IEEE

The receiver accepts the data and causes a transition on *Acknowledge* wire, completing the data transfer. The sender is then free to remove the current data value and set up for next data transmission. For correct operation, the data value must arrive at the receiver before the *Request* event and data must be held stable until the *Acknowledge* event is received (bundling constraint).

To ease the design of control circuits for transition signaling in a modular approach, a set of event logic modules has been proposed as shown in Figure 2.2 [5]. Note that, an arbiter module is excluded since it is not reliable to be implemented in current FPGA technology [6].

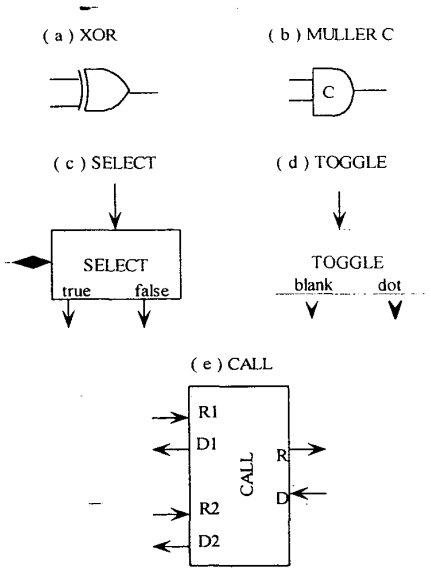

Figure 2.2: FPGA Event Logic Modules

These modules are described below:

a) **XOR circuit:**
The XOR circuit act as an OR function for events.

b) **Muller C-element:**
The MULLER C-element performs AND function for events. When both inputs of a MULLER C-element are in the same logical state, the input's state is copied into its output. When both inputs are different, it retained its previous state and hold its' output unchanged.

c) **TOGGLE circuit:**
The TOGGLE circuit steers an incoming events to its outputs alternately starting with the dot output.

d) **SELECT module:**
The SELECT module steers events to one of two outputs (true or false) based on an input Boolean value.

e) **CALL module:**
The CALL module act as a procedure call where it remember which client, R1 or R2, called the procedure, R, and after the procedure is done, D, returns a matching done event on D1 or D2.

Micropipelines offer the opportunity to construct complex control circuits and systems by the hierarchical composition of simple modules. The two-phase signalling protocol allows modules of widely differing performance to be easily integrated into a complete, correctly functioning system. The data-driven execution rates of individual asynchronous modules allow the benefits of average performance of the systems.

3. Processor organization and design

The developed 8-bit (Reduce Instruction Set Computer) RISC asynchronous processor is based on Harvard architecture with three stages pipeline: fetch, decode and execution unit. Harvard memory architecture (separate instruction and data memory) allows greater parallelism for instruction execution [7]. Data passing through each of the units of the processor are based on bundle data approach. The request and acknowledge signals are generated by request and acknowledge controller in each processor unit. Figure 3.1 shows the processor organization.

The main function of the fetch unit is to generate sequential instruction address. It consists of program counter (PC), incrementer (INC) circuit and First In First Out (FIFO) buffer, which are shown in Figure 3.2. After reset signal is asserted, the program counter starts to fetch the first instruction from the instruction memory and pass to the incrementer where the next instruction address is generated (PC+1). The FIFO buffer is used to prevent deadlock [8]. The multiplexer control whether branch, jump or incremented value will be the next instruction address. Event registers are served as a pipeline latch to decode unit.

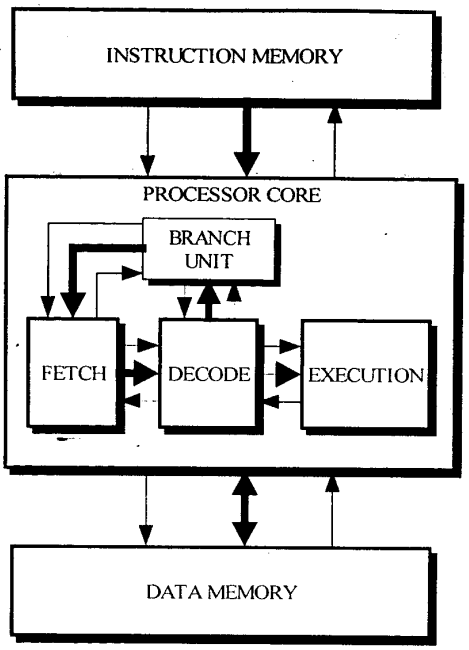

Figure 3.1: 8-bit FPGA asynchronous processor organization

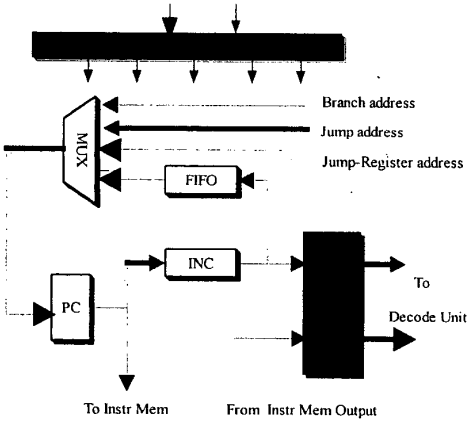

Figure 3.2 : Fetch Unit

In decode unit, the instruction is decode accordingly to the instruction format. They are register, immediate and jump format. The operation code (Opcode) of the instruction will be further decoded in the controller to generate the control signal to the operands. The FIFO serves as the decoupling buffers before the control signals are passed to the execution unit. The 8 8-bit register files have two read ports and one write port. The Extender unit is only accessible by the immediate format instruction. The decode unit is shown in Figure 3.3.

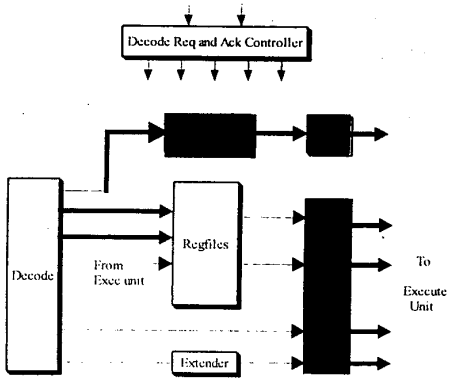

Figure 3.3: Decode Unit

The branch unit, which is shows in Figure 3.4, is responsible to generate the branch address to the fetch unit. This unit prevents the next instruction to stall due to the branches, hence eliminate pipeline suspensions and avoid performance degradation.

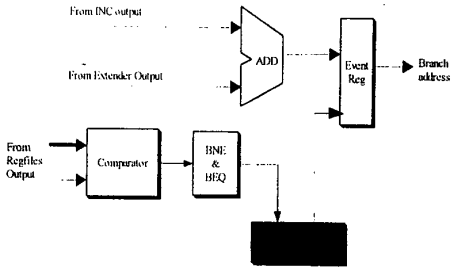

Figure 3.4 : Branch Unit

Major data processing of the processor is done at execution unit. Figure 3.5 shows the execution unit. The 8-bit ALU support arithmetic logic (ADD,SUB), logical (AND,OR,XOR), shift (SLL,SRL) and set operation (SLT). The ALU output can act as the address to the data memory or pass back to the register files. Only load (LB) and store (SB) instruction can access the data memory.

Currently, the processor supports 24 instructions, which include register, immediate, load-store, branch and jump operation but does not handle interrupt operation.

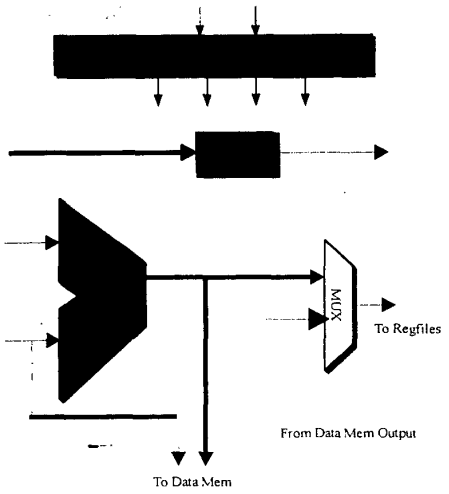

Figure 3.5 : Execute Unit

4. Design Flow

The development of the asynchronous processor generally follows top-down design approach. Each of the processor units is divided into smaller modules to ease the design. After each module has been checked and simulated correctly, they are connected as a structural VHDL to form bigger design units. This process is repeated until the whole processor is developed. Figure 4.1 shows the design flow of the asynchronous processor.

A behavioural model of the processor was first developed. This model cannot be synthesized into circuits by applying current FPGA technology but is very useful to check the functionality of the circuit and detect any error in the early design stage. By applying input test vectors (testbench), the functionality of the processor is checked through the waveform viewer.

The behavioural model is then refined to the gate-level VHDL model. This model is used as a input to the Maxplus-II software environment. Note that the datapath units like adder, multiplexer, decoder, register and arithmetic logic unit (ALU) are still using the same behavioral VHDL since the synthesize VHDL subset is supported by the software. The instruction and data memory are implemented using Altera Library of Parameterized Modules (LPM) function, which uses parameters to achieve scalability, adaptability, and efficient silicon implementation.

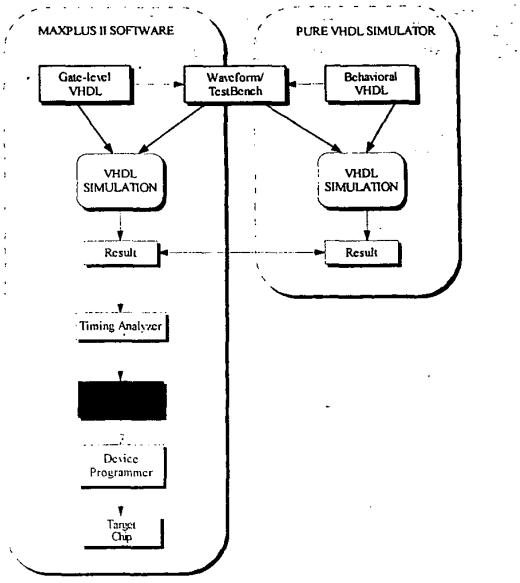

Figure 4.1: FPGA asynchronous processor design flow

The critical path of the datapath units is determined by using Timing Analyzer. This is importance to ensure the bundle data methodology is satisfied within the design environment. The delay elements are implemented using a series of inverter and LCELL buffer. ALU, instruction and data memory are implemented in Embedded Array Block (EAB) to reduce and optimize the delay time, hence boost up the performance of the processor.

The synthesized structural model of asynchronous processor are then simulated and tested by applying test vectors to the waveform editor before being program into the FPGA chip (FLEX 10K) using SRAM object nelist through device programmer.

5. Result and Discussion

The design of asynchronous circuits is in general more complex and difficult than synchronous. Asynchronous circuits need to avoid hazards and metastability, which will cause the circuit malfunction. Remove hazards from an FPGA architecture can be done by careful design. Adding delay element to the circuits can achieve hazard-free implementation. However, this may not be

true for other asynchronous circuits. The designer may need to manipulate the circuits' specification to avoid hazard.

In addition, placement and routing by the technology mapping will affect the delay of the modules. Designer needs careful routing and placement of the modules to ensure that the delay elements are within the bundled data design methodology.

Figure 5.1 shows the simulation environment and the waveform of the TOGGLE element.

The lack of arbitration mechanisms support for current FPGA architecture causes the arbiter element cannot reliably be implemented in a purely digital circuit. An arbiter element is useful to perform interrupt function since it can resolve the metastability and race condition efficiently.

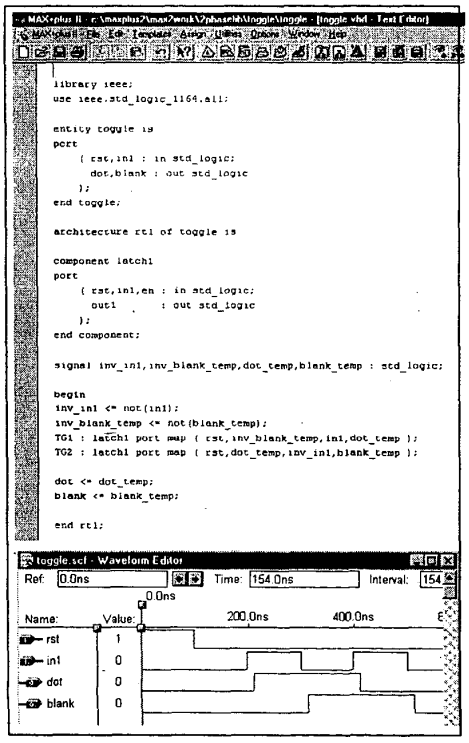

Figure 5.1: Maxplus-II VHDL Simulation environment and Waveform Viewer.

6. Conclusion

The design and implementation of asynchronous processor has shown that the programmable logic device can be used to construct relatively complex and powerful asynchronous circuits. This processor is not intended to be a fully custom designed processor such as the AMULET and TITAC processor, but rather to investigate the possibility to integrate with current FPGA technology.

References

[1] Berkel K.V., Norwick S.M., Josephs M.B., "Scanning the Technology", Proceedings of the IEEE, Special Issue : Asynchronous Circuits and Systems,pp.223-233,February 1999.

[2] Hauck S., " Asynchronous Design Methodologies : An Overview", Proceedings of the IEEE, 83(1):69-93,January 1995.

[3] Furber S.B., Day P., Garside J.D., Paver N.C., Woods J.V., "Amulet1: A Micropipelined ARM", Proceedings of IEEE Computer Conference, San Francisco, USA, March, 1994.

[4] Nanya T. et al., " TITAC : Design of a Quasi-Delay-Insensitive Microprocessor", IEEE Design and Test of Computers,1994,pp.50-53

[5] Sutherland I.E., "Micropipelines", Communicatin of the ACM, 32(6):720-738, June1989.

[6] Hauck S., Borriello G., Burns S., and Ebeling C., " An FPGA For Implementing Asynchronous Circuits", IEEE Design and Test of Computers, 11(3):60-69,1994

[7] Henessey J.L., Patterson D.A., " Computer Architecture: A Quantitative Approach", Morgan Kaufman Publishers, Palo Alto, CA, 1990.

[8] Paver N.C., "The design and implementation of an asynchronous microprocessor", Ph.D. Thesis, Department of Computer Science, University of Manchester, England, 1994.