Instruction Decoder Implemented with Balsa for an Asynchronous Pipelined 8051 compatible Microcontroller

Chang-Jiu Chen, Wei-Min Cheng, Tuan-Chieh Wang, Yuan-Teng Chang, Hung-Yue Tsai
Department of Computer Science,
National Chiao-Tung University,
HsinChu, Taiwan
{cjchen,wmcheng,wangtc,yuanteng,tsaihy}@csie.nctu.edu.tw

Abstract*-The 8051 is one of the most widely used microcontrollers today especially on many simple embedded systems. Traditionally, digital systems are implemented with synchronous circuits. Because of synchronous circuit nature, a global distributed clock signal is needed. However, the global distributed clock may cause some problems, such as harder and harder clock distribution, worse-case performance, sensitive to variations in voltage and temperature, more power consumption, and higher EMI. These problems can be easily overcome by asynchronous circuits. It is widely known that the 8051 processor is the most popular 8-bit microcontroller; however, because of its CISC nature, the instruction decoder is not very easy to implement, especially for asynchronous circuits. In this paper, we propose a new instruction fetcher and decoder model for an asynchronous pipelined 8051 microcontroller. The proposed design is modeled with a CSP-based asynchronous HDL called Balsa HDL and synthesized into Xilinx netlist with the Balsa synthesis tool. In addition, the performance will be estimated with several different design configurations.*

Keywords: Asynchronous circuit, 8051, Balsa, Instruction decoder, FPGA

1. Introduction

Microcontrollers today are widely used in lots of different applications, such as embedded systems. Most of these applications require microcontrollers that can operate in low power and different operation environments. Asynchronous circuits may be one of the best answers to these problems [1]. However, without the global clock signal, the asynchronous circuits become very hard to design. In addition, because lack of tools can be used for designing and testing, it causes the asynchronous circuits even harder to design. That's why almost all circuits are still implemented in synchronous circuits today [1].

Because of the applications, most embedded systems and simple handheld devices do not really need very powerful processor cores. Most of them needs processor core with some other special requirements such as adaptation to operating variations, low power consumption, and low EMI. These goals can be easily achieved by asynchronous circuits.

8051, AVR, and MicroChip PIC family microcontrollers are all popular 8-bit microcontrollers for embedded systems. Among these 8-bit microcontrollers, the 8051 microcontroller is the most popular of all. In fact, the 8051 series microcontrollers are still widely used in lots of different applications. There are several asynchronous 8051 compatible microprocessor implemented [2,3,4]. Because of its CISC nature, it's not very easy to implement it with pipeline architecture directly. Furthermore, the instruction decoder is the hardest part to implement. Moreover, different from most of those 8051 implementations our research focuses on developing an open synthesizable asynchronous 8051 implementation [5]. In this paper, we propose an instruction decoder implemented with Balsa HDL for such asynchronous pipelined 8051 compatible microcontroller.

2. Related Works

Asynchronous circuits have been studied for over 50 years. However, because of some historical issues and implementation difficulties, most systems today are still implemented with synchronous circuits.

Instead of the global clock signal, different handshaking protocols are used to make sure the operation correctness of asynchronous circuits. These protocols can be divided into control

signaling and data signaling. There are two types of control signaling. One is the two-phase handshaking protocol, and the other is four-phase handshaking protocol. There are also two major types of data signaling. One is the bundled-data or called single-rail data encoding, and the other is dual-rail data encoding. A complete handshaking protocol is a combination of the control and data signaling [6].

Based on these handshaking protocols, some different asynchronous pipeline models are proposed. David Muller proposed his famous Muller C-element and Muller pipeline in 1959 [7,8]. The Ivan E. Sutherland even proposed his famous Turing Award "Micropipeline" Lecture in 1959 [9]. Except the pipeline model as the control circuitry, some asynchronous processors are proposed. Amulet series processors may be the most important of all [10,11,12]. In fact, they are the earliest asynchronous ARM compatible processors. In fact, the original Amulet design is just based on the "micropipeline" model. Takashi Nanya et al. proposed their quasi-delay-insensitive (QDI) 8-bit microprocessor model called "TITAC" which uses Martin's Q-element as the control circuitry [13]. In addition, the TITAC2 was proposed to show a new delay model called scalable-delay-insensitive (SDI) [14]. Martin et al. in Caltech have already shown three generations of different asynchronous processors modeled with CHP [15]. In fact, there are also several asynchronous 8051 compatible implementations. The most famous of all is the asynchronous 8051 microcontroller developed by the Philips Research Laboratories [2]. This asynchronous 8051 was modeled with Tangram [16] and has already become a commercial product. Lee et al. proposed a novel asynchronous pipelined 8051 [3] and Martin et al. proposed a QDI asynchronous 8051 called the Lutonium [4]. In fact, because of new applications of microcontrollers, more and more research focuses on implementing microcontrollers with asynchronous circuits. However, designing the instruction decoder for an asynchronous pipelined processor with CISC ISA becomes the critical issue. Stevens et al. proposed an asynchronous instruction length decoder for CISC processors [17]. The design can be used in 400MHz high performance commercial CISC processors. Because 8051 is only a simple 8-bit microcontroller, we propose a simple but useful instruction decoder design model for asynchronous pipelined 8051 microcontroller.

3. Balsa Framework

Because lack of EDA tools can be used directly, it makes asynchronous circuit design much harder. To design synthesizable asynchronous circuits becomes even harder. However, Balsa is a framework for providing an asynchronous HDL and synthesizing of asynchronous circuits and systems [18,19,20]. It's an open source solution provided by the University of Manchester. The Balsa back-end can generate gate-level netlist that can be used by several target CAD systems. The Balsa back-end now can support three back-end protocols for target technology supported: bundled-data scheme using a 4-phase-broad/reduced-broad signaling protocol, a delay-insensitive dual-rail encoding and a delay-insensitive 1-of-4 encoding. With the Balsa framework, designing asynchronous circuits becomes easier.

4. The Proposed Instruction Decoder

The proposed instruction decoder was designed for an asynchronous pipelined 8051 with five pipeline stages as shown in figure 1 [5]. The five stages are IF (Instruction Fetch), ID (Instruction Decode), OF (Operand Fetch), EXE (Execute), and WB (Write Back) stages. In this paper, we focus on how to efficiently fetch variable length 8051 CISC instructions from instruction ROM and decode the instructions for an asynchronous pipelined 8051 microcontroller. In this section, the detailed design of IF and ID stages will be described.

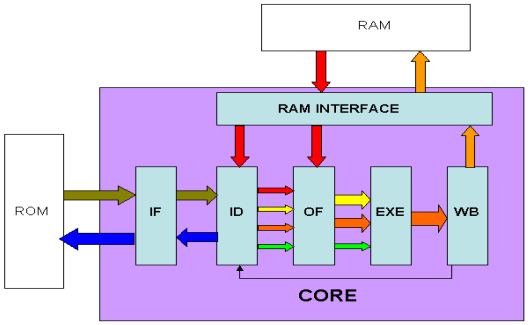

Figure 1. Architecture of Asynchronous Pipelined 8051 Microcontroller

4.1. The design of IF stage

Figure 2 shows the design of IF stage. It consists of mem_interface, two 32-byte buffers, and fetcher_ctrl. The mem_interface arbitrates request from the two buffers. The buffers are controlled by the fetcher_ctl in order to transfer a needed byte to fetcher_ctl or prefetch data from the instruction ROM. The fetcher_ctrl receives the program counter value and checks if it is hit in one

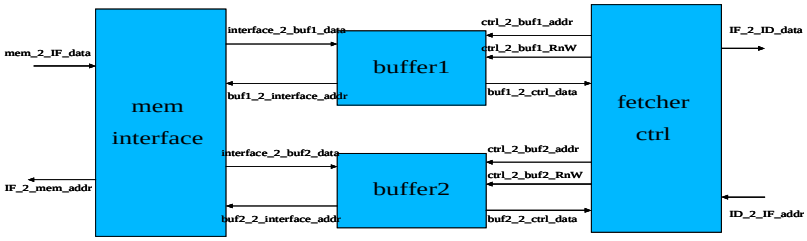

Figure 2. Block diagram of IF stage

of the two buffers. If it is a hit, fetcher_ctrl sends a request of read to the hit buffer; however if it is a miss, the fetcher_ctrl sends a request of prefetch to all buffers. Thus the mem_interface can begin to fetch data from the instruction ROM. It should be noticed that if the last byte of the target buffer is read, fetcher_ctrl will also send a prefetch request.

4.1.1. The mem_interface. The mem_interface arbitrates the instruction ROM access requests from the two buffers and returns the target byte depending on the target address.

4.1.2. The two buffers. Each buffer can receive Read/Write control signal and target address from the fetcher_ctrl. If the control signal is READ, the buffer returns target byte to the fetcher_ctrl; otherwise, if the control signal is WRITE, it fetches 32 bytes starting from the target address.

4.1.3. The fetcher control. The fetcher control (fetcher_ctrl) is responsible for controlling the operations of the two buffers. Once the fetcher_ctrl gets the target address, it checks if the target byte exists in one of the two buffers. If the target byte can be found in one of the two buffers, the target address and READ request can be sent to the hit buffer. Thus the target byte can be passed to the ID stage. Then it checks if this byte is the last byte of that buffer. If it were true, fetcher_ctrl generates the WRITE control signal to that buffer. On the contrary, if the target byte cannot be found in both of the two buffers, the fetcher_ctrl generates flush control signal to flush both of the two buffers. Therefore the two buffers can be refilled.

4.2. The design of ID stage

Figure 3 shows the design of ID stage. Because the 8051 instruction set is very complex and hard to decode directly, we divided the ID stages into two sub-stages, the ID1 and ID2. In ID1 stage, it fetches the first byte of an instruction, decodes the instruction, determines the length of the remainder bytes, abstracts the opcode, and finally generates control signals. Thus the remainder bytes can be fetched in the ID2 stage. In addition, if the current instruction is a branch instruction, the ID2 stage is responsible for calculating the target address and handling this branch. Finally, ID2 stage provides complete control signals to OF stage.

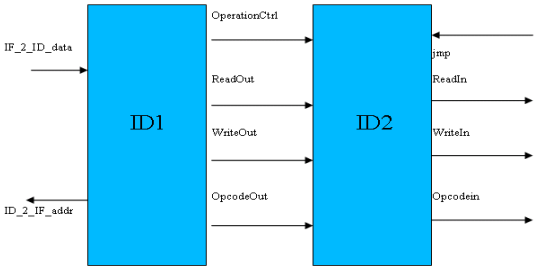

Figure 3. Block diagram of ID stage

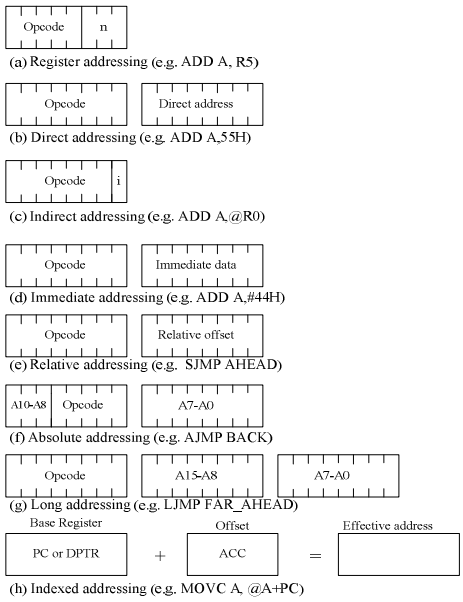

Figure 4. Addressing modes of 8051 instructions

4.2.1. The 8051 instruction set. There are total 255 instructions with variable length from one to three bytes of 8051 [21]. Depending on the addressing modes, they can be divided into eight types. The eight addressing modes are depicted in figure 4. In fact, an 8051 instruction can be determined from its first byte, and all the extra bytes are operands.

4.2.2. The ID1 stage. In order to decrease the size of the multiplexer in ID1 stage, the incoming byte is used to determine if it is a regular or irregular instruction. Then the control signals of opcode, operation control, read control, and write control that are needed in ID2 stage can be generated. Figure 5 depicts the operation of ID1 stage. In addition, the share procedure of Balsa is used; thus, only one component will be constructed whenever this procedure is called.

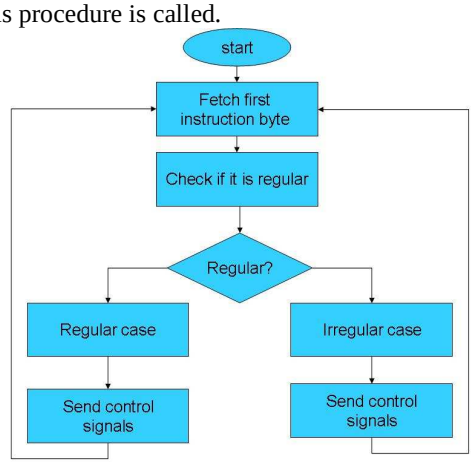

Figure 5. Flow control of ID1 stage

4.2.3. The ID2 stage. In ID2 stage, the remainder bytes of the current instruction will be fetched according to the control signal from ID1 stage. To avoid the race condition between ID1 and ID2 stages, the "handshake enclosure" description of Balsa is used. Therefore the remainder bytes can be correctly fetched before the next instruction can be begun to fetch in ID1 stage. Once all the remainder bytes are fetched, they will be transformed into corresponding operands and combined with the control signals from ID1 stage to OF stage. If the current instruction is a branch instruction, the target address will be calculated. In addition, if it's a taken branch, the PC value will also be changed in ID2 stage. Figure 6 shows the flow control of ID2 stage.

5. Implementation

The design is modeled with Balsa language and then compiled into a collection of "handshake components" with the balsa-c compiler. All these handshake components can be mapped into gate-level implementations. Hence, the balsa-netlist tool can be used to transfer them into Verilog netlist for Xilinx or other target synthesis tools automatically. We implemented the design with Xilinx Spartan-IIE 300 ft256 FPGA. To reduce the area cost, the four-phase bundled-data protocol was adopted. Figure 7 shows the Balsa FPGA design flow. Finally, the cross-verification with our design was done with an 8051 simulator.

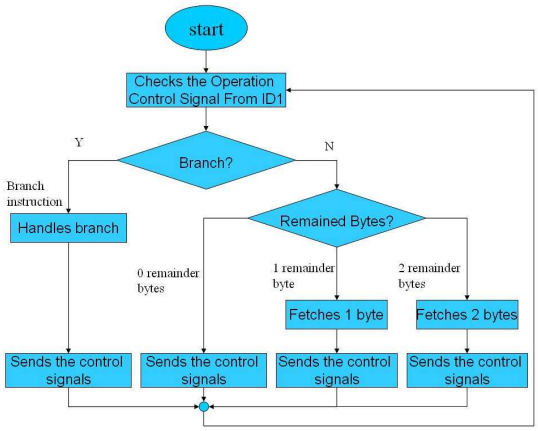

Figure 6. Flow control of ID2 stage

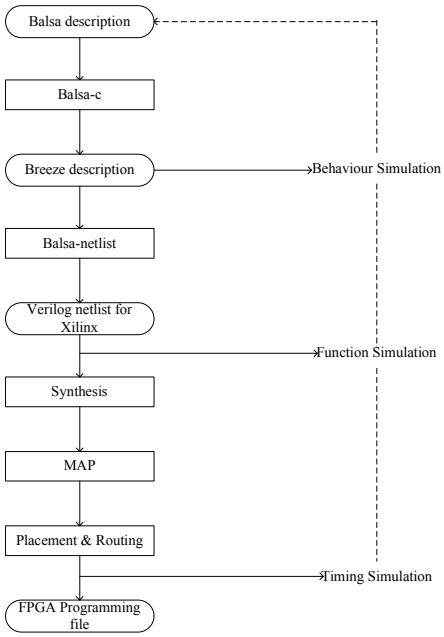

Figure 7. The Balsa FPGA design flow

6. Simulation result

There are two major different design configurations that may impact on the performance. In order to select the best configurations, simulations were used to compare the performance. Because the instruction memory size of 8051 is only 64KB or less, most applications of 8051 are very small and simple. Don't forget that 8051 is just a simple 8-bit microcontroller. In our simulation, we selected the GCD and Fibnacci code as our benchmark program. Though the simulation results may vary from application to

application, it may not have great impact on these design configuration selections.

The first design issue is the number of 32-byte buffers in IF stage. That may impact the instruction issue rate very seriously. Table 1 shows consuming time in Balsa time unit with number of buffers from 0 to 3. It can be found that the best performance can be delivered if the number of 32-byte buffers is 2. In comparison with the design of 0 or 1 buffers, the performance of design of 2 buffers would be better. That's very easy to understand. If the number of buffers is 2, it can not only save the instruction memory access time but hide some accesses when the instructions are executed. However, if the number of buffers is continuously increased, the performance will begin to decrease. That's because the arbitration in mem_interface will become more complex.

Table 1. The execution time needed with different number of buffers

Numbers of buffers	Execution time (in Balsa units)	Normalized result
0	155,062,000	20.55
1	11,896,100	1.58
2	7,544,200	1.00
3	10,012,600	1.33

The second issue is the buffer size. Though the 32-byte size is selected, we still hope to find the most suitable one. However, the most suitable size may really depend upon the applications. That's because with larger buffer size more instructions in the same basic block can be hold in the buffer. Table 2 shows the execution time of buffer with different buffer sizes. Maybe because of our application behavior, the performances of different buffer sizes have no great differences. For a reasonable selection, 32-byte size is still adopted in our design.

Table 2. The execution time needed with different size of buffers

Buffer Size (byte)	Execution time (in Balsa units)	Normalized result
8	7,801,000	1.03
16	7,629,800	1.01
32	7,544,200	1.00
64	7,501,400	0.99

Now, it's time to estimate the overall performance improvements of asynchronous pipelined 8051 design. In order to compare the performance, we compare the performance of two asynchronous 8051 models. The two asynchronous 8051 microcontroller core developed by us are all modeled with Balsa HDL. Table 3 shows the area costs and performance of the two designs in Balsa area and time units respectively. The table shows that the pipelined design can deliver about 3.3 times performance with 2.35 times area costs. The design was finally implemented with Xilinx FPGA. Table 4 shows the area cost and minimum path delay on Xilinx FPGA.

Table 3. Execution and Area Cost of Asynchronous Non-pipelined and Pipelined 8051

	Execution Time	Normalized time
Non-pipelined	24,891,300	3.30
Pipelined	7,544,200	1
	Cost	Normalized Cost
Non-pipelined	210,513.50	1
Pipelined	494,752.25	2.35

Note: The time and area are all represented in Balsa units.

Table 4. Cost and minimum path delay on FPGA

	Slice	Gate count	Minimum path delay(ns)
IF	1007	13987	757
ID	5353	61973	721
MEM_INTERFACE	1098	13217	125

7. Conclusion and discussion

In this paper, we propose an open and synthesizable asynchronous instruction fetcher and decoder implemented with Balsa HDL for an asynchronous pipelined 8051 microcontroller. In fact, the model we proposed has been already implemented inside an asynchronous pipelined 8051 microcontroller core and realized with Xilinx FPGA.

Because of the CISC nature, instruction fetch and decode is very hard to implement especially for an asynchronous pipelined processor. Though it's very hard, we propose a very simple model for an asynchronous 8051 core. Moreover, it doesn't really need to implement very complex instruction fetcher and decoder for such simple 8-bit microcontroller. Though to simplify the design we divided IF and ID into several sub-stages, all the

sub-stages can work correctly with its own speed. Once any sub-stage has finished its operation, the adjacent sub-stage can begin its operation directly. That's because it's implemented with asynchronous circuits. The synchronization is done via handshaking protocol. Thus the advantage of average-case performance can be achieved. Finally, via the simulations, we show why we implemented two 32-byte buffers in the instruction fetcher. We also show that it's worth to implement asynchronous 8051 with pipelined model. It should be pointed out that because this design was modeled with Balsa HDL, it can be re-synthesized with other handshaking protocols. Thus this design has very high flexibility.

References

[1] A. Davis and S.M. Nowick, "An Introduction to Asynchronous Circuit Design," *Technical Report*, UUCS-97-013, Computer Science Department, University of Utah, Sep. 1997.

[2] H.V. Gageldonk, K.V. Berkel, A. Peeters, "An Asynchronous low-power 80C51 microcontroller," in *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 96-107, 30 March-2 April 1998.

[3] J. H. Lee, W. C. Lee, and K. R. Cho, "A novel asynchronous pipeline architecture for CISC type embedded controller - A8051, " in *Proceedings of the 2002 45th Midwest Symposium on Circuits and Systems*, Vol. 2, pp. 675-678, Aug. 2002.

[4] A. J. Martin, M. Nyström, K. Papadantonakis, P. I. Pénzes, P. Prakash, C. G. Wong, J. Chang, K. S. Ko, B. Lee, E. Ou, J. Pugh, E. V. Talvala, J. T. Tong, and A. Tura, "The Lutonium: a sub-nanojoule asynchronous 8051 microcontroller," in *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems*, pp. 14-23, May 2003.

[5] C.J. Chen, W.M. Cheng, R.F. Tsai, H.Y. Tsai, and T.C. Wang, "A Pipelined Asynchronous 8051 Soft-core Implemented with Balsa," in *IEEE Asia Pacific Conference on Circuits and Systems*, 2008 (to be appeared).

[6] J. Sparsø and S. Furber, *Principles of asynchronous circuit design – a systems prospective*, Kluwer Academic Publishers, London, 2001.

[7] D. Muller and W. Bartky, "A theory of asynchronous circuits," in *Proceedings of an International Symposium on the Theory of Switching*, pp. 204-243, April 1959.

[8] J. Gunawardena, "A generalized event structure for the Muller unfolding of a safe net," in *Proceedings of the 4th International Conference on Concurrency Theory*, pp. 278-292, June, 1993.

[9] I.E. Sutherland, "Micropipelines," Turing Award Lecture, *Communications of the ACM*, Vol.32, Number 6, pp 720-738, June 1989.

[10] J. V. Woods, P. Day, S. B. Furber,J. D. Garside,N. C. Paver, S. Temple, "AMULET1: an asynchronous ARM microprocessor," *IEEE Trans. Computers*, Vol. 46, pp. 385-398, April 1997.

[11] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, N. C. Paver, "AMULET2e: an asynchronous embedded controller," in *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 290 – 299, April 1997.

[12] S. B. Furber, D. A. Edwards and J. D. Garside, "AMULET3: a 100 MIPS Asynchronous Embedded Processor", in *Proceedings of the International Conference on Computer Design*, pp. 329-334, 2000.

[13] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takmura, "TITAC: Design of a Quasi-Delay-Insensitive Microprocessor", *IEEE Design & Test of Computer*, pp. 50-63, Summer 1994.

[14] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya, "TITAC-2: A 32-bit Asynchronous Microprocessor based on Scalable-Delay-Insensitive Model," in *Proceedings of the International Conference on Computer Design*, pp. 288-294, Oct. 1997.

[15] Alain J. Martin, Mika Nyström, and Catherine G. Wong, "Three Generations of Asynchronous Microprocessors," *IEEE Design & Test of Computers*, pp. 9-17, Nov.-Dec. 2003.

[16] K. V. Berkel, J. Kessels, M. Roncken, R. Saeijs, P. Schalij, "The VLSI-programming language Tangram and its translation into handshake circuits," in *Proceedings of the European Conference on Design Automation. EDAC*, pp. 384 – 389, 25-28 Feb. 1991.

[17] Kenneth S. Stevens, Shai Rotem, Ran Ginosar, Peter Beerel, Chris J. Myers, Kenneth Y. Yun, Rakefet Kol, Charles Dike, Marly Roncken, "An Asynchronous Instruction Length Decoder," *IEEE Journal of Solid State Circuits*, Vol 36, No. 2, pp. 217-228, Feb. 2001.

[18] A. Bardsley, *Implementing Balsa Handshake Circuits*, PhD thesis, Dep. of Computer Science, Univ. of Manchester, 2000.

[19] A. Bardsley, D. A. Edwards, *The Balsa Asynchronous Circuit Synthesis System*, Dep. of Computer Science, Univ. of Manchester, 2000.

[20] A. B. Doug Edwards, *Balsa: A Turorial Guide version 3.4*, Dep. of Computer Science, Univ. of Manchester, 2004.

[21] Intel, *MCS51 Microprocessor Family User's Manual*, Intel, 1994.