2008 14th IEEE International Conference on Parallel and Distributed Systems

Processor, Assembler, and Compiler Design Education using an FPGA

Koji Nakano and Yasuaki Ito
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527, JAPAN

Abstract

This paper reports the design of two courses, "Embedded Hardware" and "Embedded Software" offered in 2008 Spring semester at Hiroshima University. These courses use 16-bit processor TINYCPU, cross assembler TINYASM, and cross compiler TINYC. They are designed very simple and compact: The total number of lines of the source code is only 427. Thus, students can understand the entire design easily, and can learn the basics of computer and embedded system, including processor architecture, assembler and compiler design, assembler programming in a unified way by experiment.

1 Introduction

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions. It is usually embedded as part of a complete device including hardware and mechanical parts. Since embedded systems usually use processors with limited computational power and few hardware resources, it is important to write efficient programs controlling the systems.

In most computer engineering departments, students are leaning computer programming using high-level languages such as C, Pascal, FORTRAN, etc. For developing efficient programs, it is necessary to understand how computer programs are executed. For example, it is not easy to understand how formulas in programs are evaluated. Further, it is very difficult to understand a data structure "pointer" if one does not know how it is implemented in a machine language. Unfortunately, most students are learning high-level language programming from just syntax of programs.

The main contribution of this paper is to present a simple, compact and portable processor, TINYCPU, which can be implemented in various FPGAs. We also present a cross assembler TINYASM and a cross compiler TINYC for TINYCPU. TINYCPU, TINYASM and TINYC are designed using Verilog HDL, Perl, and Flex/Bison, respectively. Table 1 summarizes TINYCPU, TINYASM, and TINYC. In this table, code size includes blank lines. Quite surprisingly, TINYCPU, TINYASM, and TINYC are so simple and compact that their total code size is 427 lines. Hence, it is not difficult for students to understand the entire design of TINYCPU, TINYASM, and TINYC. Also, students can extend the design by themselves very easily.

We also report the design of two courses "Embedded Hardware" and "Embedded Software" (2 credits each) offered for graduate students in 2008 Spring semester at Department of Information Engineering, Hiroshima University. These two courses use TINYCPU, TINYASM, and TINYC as course materials. These courses use the FPGA boards in Spartan-3E and Spartan-3A starter kit (Figure 1) to implement and execute TINYCPU. The Spartan-3E and Spartan-3A starter kits FPGA boards are equipped with Spartan-3E family FPGA XC3S500E [11] and Spartan-3A family FPGA XC3S700A [10], respectively. Both FPGA boards have various switches (slide switches, button switches and a rotary switch), LEDs, and LCD. They also have VGA, PS/2, and Ethernet ports. In our TINYCPU implementation, we use switches to provide clock pluses and input values to TINYCPU, and LED to indicate the current state of the state machine, and LCD to display miscellaneous values including the program counter, the instruction register, the address bus, the data bus of TINYCPU. Thus, students can trace the behavior of TINYCPU in each clock cycle easily.

Although many courses [2, 5, 8] for embedded systems are offered, they are using existing processor such as ARM and so on. As far as we know, there is no course teaching the design of processor, assembler, and compiler in a unified way by experiment.

This paper is organized as follows: In Section 2, we show the architecture of TINYCPU, and its instruction set. Section 3 describes our cross assembler TINYASM and cross compiler TINYC. It also shows an example of a C-based language program and the resulting object code provided using TINYCPU and TINYASM. Sections 4 and 5 show how we have used TINYC, TINYASM, and TINYC as course materials for education. Section 6 offers concluding remarks.

1521-9097/08 $25.00 © 2008 IEEE
DOI 10.1109/ICPADS.2008.71

Table 1. Our tiny processing system and its code size

		language	module or function	code size (lines)
TINYCPU	16-bit Processor	Verilog HDL	definitions (`defs.v`)	36
		Verilog HDL	ALU (`alu.v`)	39
		Verilog HDL	counter (`counter.v`)	14
		Verilog HDL	state machine (`sate.v`)	23
		Verilog HDL	stack (`stack.v`)	28
		Verilog HDL	memory (`ram.v`)	27
		Verilog HDL	top module (`tinycpu.v`)	112
		total		279
TINYASM	Cross Assembler	Perl		38
TINYC	Cross Compiler	Flex	lexical analysis	29
		Bison	context analysis code generation	81
		total		148
total				427

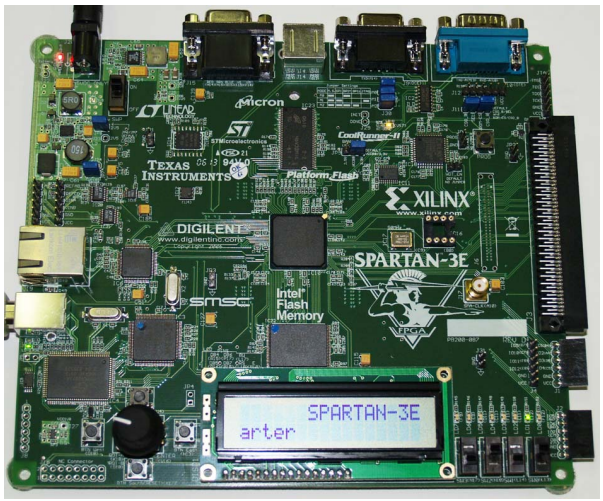

Figure 1. Spartan-3E Starter Kit

2 The Architecture of TINYCPU

TINYCPU is a pure stack architecture [4] and does not have an accumulator or a register set. Instead, it has an operation stack, which is used for all operations including store, load, arithmetic, and logic operations. The arithmetic and logic instructions do not have operands, and the operations performed for the stack.

TINYCPU is designed using Verilog HDL. The Verilog HDL code for TINYCPU is written as simple as possible, and the efficiency including hardware resources and clock frequency are of secondary importance. Figure 2 illustrates the block diagram of TINYCPU. It has seven components including state machine `state`, 12-bit program counter `pc`, 16-bit instruction register `ir`, 16-bit output buffer `obuf`, 16-bit ALU `alu`, 16-bit stack `stack`, 16-bit data and 12-bit address memory `ram`. It also uses 16-bit data bus `dbus` and 12-bit address bus `abus`.

Every instruction of TINYCPU is a 16-bit word. Table 2 shows the list of all instructions of TINYCPU. It has 9 control instructions and 19 instructions for arithmetic and logic operations. In the table, operands `I` and `A` are immediate and address values, respectively, and `f` is a 5-bit code to specify an operation. Also, `top` and `next` denote the top and the second elements of the stack. Hence, the binary operations are performed for `next` and `top` and the resulting value is stored in `top`. The unary operations are performed for `top`. The readers may think that TINYCPU has too few instructions. However, these instructions are sufficient to execute machine codes generated by C-based language programs that we will explain later.

3 Cross Assembler and Cross Compiler

We have designed a cross assembler and a cross compiler for TINYCPU. The Assembler, TINYASM, translates an assembly language program into a machine code, which is a list of pairs of a 12-bit address and a 16-bit instruction codes. The compiler, TINYC, translates a C-based language program into an assembly language program for TINYASM. TINYC programming language supports 16-bit signed integers, and if, if-else, while, do, and goto statements. Also, it has basic arithmetic and logic operations including addition ($+$), subtraction ($-$), multiplication ($*$),

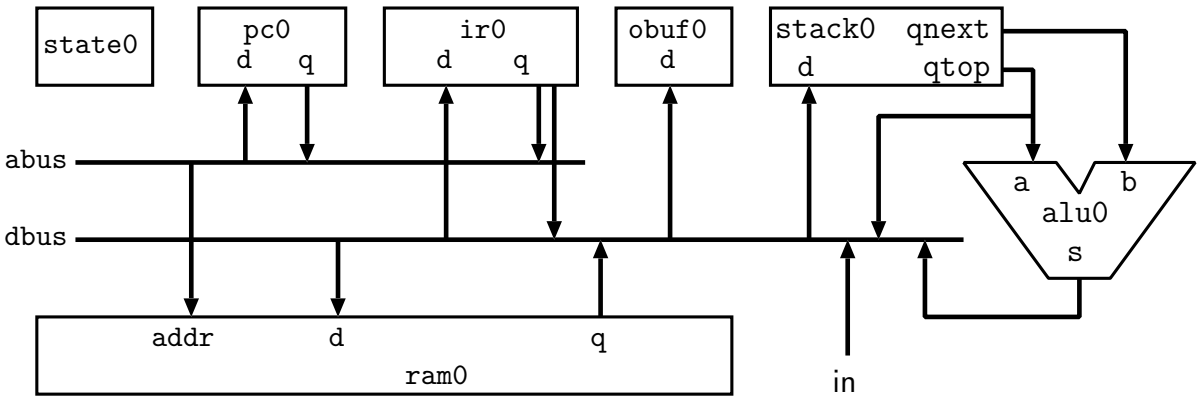

Figure 2. TINYCPU architecture

Table 2. Instruction set of TINYCPU: Mnemonic names and instruction codes

	Mnemonic	Machine Code (HEX)	Operation
1	HALT	0000	Stop
2	PUSHI I	1000+I	I→ top
3	PUSH A	2000+A	mem[A]→ top
4	POP A	3000+A	top → mem[A]
5	JMP A	4000+A	A→pc
6	JZ A	5000+A	A→pc if top=0
7	JNZ A	6000+A	A→pc if top!=0
8	IN	D000	in→top
9	OUT	E000	top→out
10	OP f	F000+f	Perform operation f
	ADD	F000	next + top → top
	SUB	F001	next - top → top
	MUL	F002	next * top → top
	SHL	F003	next >> top → top
	SHR	F004	next << top → top
	BAND	F005	next & top → top
	BOR	F006	next | top → top
	BXOR	F007	next ^ top → top
	AND	F008	next && top → top
	OR	F009	next || top → top
	EQ	F00A	next == top → top
	NE	F00B	next != top → top
	GE	F00C	next >= top → top
	LE	F00D	next <= top → top
	GT	F00E	next > top → top
	LT	F00F	next < top → top
	NEG	F010	- top → top
	BNOT	F011	~ top → top
	NOT	F012	! top → top

I: 12-bit signed integer A: 12-bit unsigned integer

negation ($-$), bit shifts ($<<$, $>>$), bitwise logic operations (&, |, ^, ~), logic operations (&&, ||, !), and comparisons ($==$, $!=$, $>$, $>=$, $<$, $<=$).

List 1 shows an example of TINYC language program `collatz.c`. Using TINYC compiler and TINYASM assembler, `collatz.c` is translated into an assembly language program and a TINYCPU machine code in List 2. The C-language program in List 1 computes the formula in Collatz conjecture [1, 9] as follows. Consider the following operation on an arbitrary positive integer n: (1) If the number is odd, triple it and add one, that is, $n \leftarrow 3n+1$, and (2) If the number is even, divide it by two, that is, $n \leftarrow n/2$. The Collatz conjecture asks if iterating this operation returns 1 for any initial value n. For example, if $n = 3$ then, we have the following sequence by iterating the operation. $3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$. It remains open if the Collatz conjecture is true.

List 2 shows the TINYASM assembly language program and TINYC machine code obtained using our TINYC compiler and TINYASM assembler. It contains a list of labels with their address values, and a machine code, which is a list of 16-bit instructions and initial values of variables.

List 1. C-based language program collatz.c for Collatz conjecture

```
n=in;
while(n>1){
  out(n);
  if(n&1){
    n= n*3+1;
  } else {
    n = n>>1;
  }
 }
out(n);
halt;
int n;
```

List 2. The translated assembly language program and machine program of collatz.c

```
*** LABEL LIST ***
_001F 018
_001T 002
_002F 013
_002T 017
n 01B

*** MACHINE PROGRAM ***
000:D000            IN
001:301B            POP n
          _001T:
002:201B            PUSH n
003:1001            PUSHI 1
004:F00E            GT
005:5018            JZ _001F
006:201B            PUSH n
007:E000            OUT
008:201B            PUSH n
009:1001            PUSHI 1
00A:F005            BAND
00B:5013            JZ _002F
00C:201B            PUSH n
00D:1003            PUSHI 3
00E:F002            MUL
00F:1001            PUSHI 1
010:F000            ADD
011:301B            POP n
012:4017            JMP _002T
          _002F:
013:201B            PUSH n
014:1001            PUSHI 1
015:F004            SHR
016:301B            POP n
          _002T:
017:4002            JMP _001T
          _001F:
018:201B            PUSH n
019:E000            OUT
01A:0000            HALT
01B:0000      n: 0
```

4 Course Design

We have used TINYCPU, TINYASM, and TINYC for two courses "Embedded Hardware" (Weeks from 1 to 4) and "Embedded Software" (Weeks from 5 to 8) for graduate students. These courses are organized in 8 weeks of 5 hours each as follows: *Week 1*: Full Adders, N-bit Adders and ALU, *Week 2*: Flip-Flops, Counters, State Machines, and Stacks, *Week 3*: Chattering removal, LCD controller, memory, and instruction fetch, *Week 4*: CPU design: instruction set, control logic, and machine program, *Week 5*: Perl language and cross assembler TINYASM design, *Week 6*: Compiler Compiler :Flex and Bison , *Week 7*: TINYC Compiler, and *Week 8*: TINYC programming. We have used Spartan-3E Starter Kit (Figure 1) and Spartan-3A Starter Kit and students implement TINYCPU in the FPGA board and confirm it works correctly by operating it. The details of the contents of the eight weeks are as follows.

4.1 Week 1: Full Adders, N-bit Adders and ALU

In Week 1, students first learn how to use ISE WebPACK, which is a free version of the FPGA development tool for Xilinx FPGAs. The main objective of Week 1 is to learn how to design a combinational logic. For this purpose, they design a full adder module and an 4-bit adder module by instantiating the full adder module. To learn how to see the correctness of a Verilog HDL module, they write a test bench for the 4-bit adder module, and perform the simulation using simulator included in ISE WebPACK. Finally, they write a Verilog HDL module for ALU (Arithmetic and

Logic Unit) and its test bench, and perform the simulation to verify if all functions of the ALU works properly.

4.2 Week 2: Flip-Flops, Counters, State Machines, and Stacks,

In Week 2, students learn how to design a sequential logic. They first write a (D-type) flip-flop with asynchronous reset, which has 1-bit input ports `clk`, `reset`, `d`, and 1-bit output port `q` and then learn a Verilog HDL description of an N-bit counter.

Students design the state machine with five states, `IDLE`, `FETCHA`, `FETCHB`, `EXECA`, and `EXECB`. Later, states `FETCHA` and `FETCHB` are used to fetch an instruction code from a memory, and the instruction is executed using two states `EXECA` and `EXECB`.

Finally, they design a stack, which has four 16-bit registers. The stack supports operations, push, pop, and load to the stack. Also, they design an operational stack using the stack and the ALU, which evaluates a formula in postfix notation. For example, for formula `3 4 5 * +`, the operational stack outputs `23`.

4.3 Week 3: Chattering removal, LCD controller, memory, and instruction fetch

In Week 3, students write the Verilog HDL source code of a chattering removal circuit for the FPGA board of the Spartan-3E/Spartan-3A starter kit. This circuit is used to remove chattering of miscellaneous switches on the FPGA board. They also implement an LDC controller, which is used to display six 4-digit hexadecimal numbers in the LCD on the FPGA board.

Students also implement a distributed RAM and a block RAM. A distributed RAM is a asynchronous read/synchronous write memory, which will be implemented in slices of the FPGA. On the other hand, a block RAM is a synchronous read/synchronous write memory, which will be implemented in building block RAMs of the FPGA. Students write test benches for them and perform the simulation to understand their difference, and implement them in the FPGA.

Finally, students develop an instruction fetch circuit using the counter module, the block RAM module, the chattering removal module, and the LCD controller module. By rotating the rotary switch on the FPGA board, clock pluses are given to the main module through the chattering removal module, and the LCD controller module is used to display the values of program counter and the instruction register, etc.

4.4 Week 4: CPU design: instruction set, control logic, and machine program

In Week 4, students first determine the logic of seventeen control wires of the TINYCPU. For example, `pcinc` is a control wire, which should be 1 iff the program counter is incremented. Thus, `pcinc` is connected to `inc` input port of the program counter and `pcinc` is 1 if the current state is `FETCHA`. Based on the logic of the seventeen control wires, they complete the Verilog HDL source code of the TINYCPU.

To verify the correctness of TINYCPU, they write a simple countdown program, which outputs n, $n-1$, ..., 0 for a given n from the input port. They write a test bench and see the countdown program works correctly by the simulation. Further, they implement TINYCPU with the countdown program on the FPGA board. The LCD on the FPGA board shows the current values of program counter, instruction register, stack top, address bus, data bus, and the output buffer. Also, 5 LEDs on the FPGA display the current state.

4.5 Week 5:Perl language and cross assembler TINYASM design

In Week 5, students first learn basics of Perl language programming including lists, associative arrays, regular expressions, pattern matching, and substitution, necessary to understand TINYASM. Students are given the source code of TINYASM which does not support error messages. Hence, they extend it to show appropriate error messages. The error messages include *undefined mnemonic*:undefined mnemonic is used, *undefined label*: undefined label is used, *multiply defined label*: the same label is defined twice or more, *immediate operand is out of range*: the immediate value of an operand is not in the range of 12-bit 2's complement, *initial value is out of range*: the initial value of a variable is not in the range of 16-bit 2's complement. This extension is not possible if they do not understand the source code of TINYASM.

4.6 Week 6: Compiler Compiler :Flex and Bison

The main objective of Week 6 is to understand how to write a compiler using the lexical scanner generating tool Flex [7] and the parser generating tool Bison [3]. For this purpose, students first write a postfix notation (or inverse polish) calculator using Flex. For example, the postfix notation calculator computes formula `3 4 5 * +` and outputs `23`. They also write the same calculator using Bison. After that, students also write a infix notation calculator using both Flex and Bison. For example, the infix notation calculator computes formula `3+4*5` and outputs `23`.

Finally they develop a very small C-like programming language cross compiler, that supports 16-bit signed integers, addition (+), subtraction (-), multiplication (*), and equality (==), goto, if-goto, and unless-goto statements.

4.7 Week 7: TINYC cross compiler

In Week 7, students develop a TINYC cross compiler using Flex and Bison by extending the small C-like language compiler written in Week 6. TINYC cross compiler supports, 16-bit signed integers, if, if-else, while, do, and goto statements. Also, it supports basic arithmetic and logic operations including addition (+), subtraction (−), multiplication (∗), negation (−), bit shifts (<<, >>), bitwise logic operations (&, |, ^, ~), logic operations (&&, ||, !), and comparisons (==, ! =, >, >=, <, <=). After that, students extend the function of TINYC, TINYASM, and TINYC. For example, to support ++ and -- operators in TINYC, they add `INC` and `DEC` unary operators in TINYCPU and also add `INC` and `DEC` mnemonics in TINYASM. Also, they modify TINYC cross compiler to support conditional operator (? :). Finally, they write a BCD counter program using TINYC programming language.

4.8 Week 8: TINYC programming

In Week 8, students develop a stop watch program using TINYC programming language. The stop watch should have three buttons, *start/stop*, *reset*, and *lap*. It also should have accuracy 0.01 seconds from 00.00 to 99.99 seconds. If "start/stop" is clicked while the counter stops, then the counter starts or restarts. if "start/stop" is clicked while the counter is incrementing, then the counter stops. If "reset" is clicked, then the value of counter becomes 0. If "lap" is clicked while the counter is working, then the counter increment continues but the display of the counter value stops. If "lap" is clicked while the counter display stops, then the displayed value is updated by the current counter value, and the counter display continue to stop. If "start/stop" is clicked while the counter display stops, then the display starts to show the current value of the counter. They also need to verify if the developed stop watch works correctly using the FPGA board.

5 The Results of Two Courses Embedded Hardware and Embedded Software

Two courses "Embedded Hardware"(Weeks 1 to 4) and "Embedded Software" (Weeks 5 to 8) offered in 2008 Spring semester at Department of Information Engineering, Hiroshima University. Eighteen students enrolled in these courses, and fifteen students of them have completed 8-week course. In Week 9, fifteen of them took the final exam to confirm that they understand the contents of the courses. From the result of final exam, we can say that students have learned and understood CPU design, Verilog HDL, assembler design, and compiler design very well.

6 Concluding Remarks

We have presented a small computer system including 16-bit processor TINYCPU, cross assembler TINYASM, and cross compiler TINYC. We have also reported the course design of "Embedded Hardware" and "Embedded Software" offered in 2008 Spring semester at Department of Information Engineering, Hiroshima University. Students taking these courses were able to learn digital circuit design using HDL, processor architectures, assembler design, assembly language programming, and compiler design in a unified way. Thus, TINYCPU, TINYASM, and TINYC can be used as good course materials to learn basics of computer and embedded system by experiment.

Based on TINYCPU, TINYASM, and TINYC, the authors have been publishing serialized articles "Verilog HDL & FPGA Design Learned from Basics" in Design Web Magazine from April 2007 [6].

References

[1] E. Akin. Why is the 3x+1 problem hard ? *Contemporary Mathematics*, 356:1–20, 2002.

[2] J. Chen, H.-M. Su, and J.-H. Liu. A curriculum design on embedded system education for first-year graduate students. In *Proc. of International Conference on Parallel and Distributed Systems (ICPADS)*, volume 2, pages 1–6, 2007.

[3] C. Donnelly and R. Stallman. *Bison: The YACC-compatible Parser Generator*. Free Software Foundation, 1995.

[4] P. Koopman. *Stack Computers: the new wave*. Ellis Horwood, 1989.

[5] I. McLoughlin, D. Maskell, S. Thambipillai, and W.-B. Goh. An embedded systems graduate education for singapore. In *Proc. of International Conference on Parallel and Distributed Systems (ICPADS)*, volume 2, pages 1–5, 2007.

[6] K. Nakano and Y. Ito. Verilog HDL & FPGA design learned from basics. appears bimonthly in Design Wave Magazine, 2007-2009.

[7] G. T. Nicol. *Flex: The Lexical Scanner Generator*. Free Software Foundation, 1993.

[8] N. Vun and W.-B. Goh. Issues and challenges of embedded processor education for working professionals. In *Proc. of International Conference on Parallel and Distributed Systems (ICPADS)*, volume 2, pages 1–8, 2007.

[9] E. W. Weisstein. Collatz problem. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/CollatzProblem.html.

[10] Xilix Inc. *Spartan-3A FPGA Family: Data Sheet*, 2008.

[11] Xilix Inc. *Spartan-3E FPGA Family: Complete Data Sheet*, 2008.