A Tiny Processing System for Education and Small Embedded Systems on the FPGAs

Koji Nakano, Kensuke Kawakami, Koji Shigemoto, Yuki Kamada, Yasuaki Ito
Department of Information Engineering, Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, JAPAN

Abstract

The main contribution of this paper is to present a simple, scalable, and portable tiny processing system which can be implemented in various FPGAs. Our processing system includes a 16-bit processor, a cross assembler, and a cross compiler. The 16-bit processor runs in 89MHz on the Xilinx Spartan-3A family FPGA XC3S700A using 336 out of 5888 slices (5.7%) and in 76MHz on the Altera Cyclon III family EP3C25F324 using 569 out of 24624 logic elements (2.3%). Every instruction can be executed in only one clock cycle, that is, CPI=1. Using a cross assembler and a cross compiler that we have developed, a C-based language program can be translated into a machine language object code, which can be executed on the 16-bit processor. The source codes of our processing system are very simple and compact. The 16-bit processor is designed by Verilog HDL using 268 lines, and the cross assembler is written in 38 lines using Perl language. The cross compiler has 23 lines of Flex grammar file for lexical analysis, and 90 lines of Bison grammar file for context analysis and code generation. Hence, our tiny processing system is portable and easy to understand and the function expansion is not difficult. Actually, the tiny processing system has been used for the embedded system course of graduate students as a course material. As real-life applications, we have developed a PONG-like mini game and an RSA encryption/decryption system based on the tiny processing system. Therefore, our tiny processing system benefits computer system education and small embedded system development.

1 Introduction

An FPGA (Field Programmable Gate Array) is a programmable VLSI in which a hardware designed by users can be embedded instantly. Typical FPGAs consist of an array of programmable logic blocks (slices or logic elements), memory blocks, and programmable interconnect between them. The logic block usually contains four-input logic functions and/or several registers. The memory block is usually a dual-port RAM which can be read/written a word of data for distinct addresses in the same time. Using design tools provided by FPGA vendors or third party companies, a hardware logic designed by users using hardware description languages can be embedded in FPGAs. Thus, the FPGAs are widely used for platform of embedded system as well as computer system education. Also, it has been shown that a lot of computation can be accelerated using a circuit implemented in FPGAs [4, 5, 11, 12, 13, 14].

The main contribution of this paper is to present a simple, scalable, and portable tiny processing system which can be implemented in various FPGAs. The processing system includes an N-bit processor TINYCPU, a cross assembler TINYASM, and a cross compiler TINYC. Using TINYC and TINYASM, C-based language programs can be translated into a machine language object code, which can be executed in TINYCPU. Table 1 summarizes our tiny processing system. In this table, the code sizes in lines include blank lines.

Our tiny processing system has a lot of advantages as follows:

Simple and compact The source codes of our processing system are very simple and compact. TINYCPU has 268 lines with Verilog HDL, TINYASM is described in 38 lines using Perl language. TINYC has 23 lines of Flex grammar file for lexical analysis, and 90 lines of Bison grammar file for context analysis. Thus, our tiny processing system is easy to understand and the function expansion is very easy.

Low CPI Every instruction of TINYCPU can be executed in only one clock cycle, that is, CPI (Clock Per Instruction)=1. Thus, the performance our 16-bit version of TINYCPU is 89MIPS(Million

Table 1. Our tiny processing system and its code size

		language	module or function	code size (lines)
TINYCPU	N-bit Processor	Verilog HDL	definitions	36
		Verilog HDL	ALU	39
		Verilog HDL	state machine	20
		Verilog HDL	stack	34
		Verilog HDL	memory	29
		Verilog HDL	top module	110
		total		268
TINYASM	Cross Assembler	Perl		38
TINYC	Cross Compiler	Flex	lexical analysis	23
		Bison	context analysis code generation	90
		total		113
total				419

Instructions Per Second) on XC3S700A, and 76MIPS on EP3C25F324.

Minimum usage of block RAMs Altera and Xilinx FPGAs have dual-port block RAMs as building blocks. For example, Altera Cyclon III family FPGA EP3C25F324 and Xilinx Spartan III Family FPGA XC3S700A have 66 9k-bit block RAMs and 20 18k-bit block RAMs, respectively. Read-/Write operations for two distinct addresses can be performed in the same time for a dual-port block RAM. We use this function of dual-port to fetch an instruction code and execute a read/write operation in the same time. Thus, instruction codes and data can be in the same block RAM, and TINYCPU uses only one block RAM as a memory to store both instruction codes and data for the minimum configuration. Also, the size of memory can be extended easily.

Scalable The word size of TINYCPU can be changed. In other words, for arbitrary integer $N \geq 8$, TINYCPU can take an N-bit architecture. Any N-bit architecture can be generated by just setting parameter N in the top Verilog HDL module of our source code.

Portable Our Verilog HDL code for TINYCPU follows Verilog-95 standard, and thus it can be synthesized by tools supporting Verilog-95 as well as Verilog 2001. Therefore, most of logic design tools can synthesize our Verilog HDL code for the processor and generate its net list. Actually, Xilinx ISE Foundation, Altera Quartus II, and third party logic synthesis tool Synplify Pro can synthesize it, and Icarus Verilog [9] can perform the simulation correctly. Also, Perl, Flex, and Bison tools that we have used are standard UNIX tools. They can also be executed on Microsoft Windows-based PCs using Cygwin [7], which provides a Linux-like environment on them.

Benchmark TINYCPU has a lot of fundamental logic components including, registers, counters, memories, combinational circuits for arithmetic and logic operations, state machines, multiplexers, tri-state buses, etc. The arithmetic and logic operations include addition, subtraction, multiplication, negation, shift, comparison, and so on. Further, since the Verilog HDL code for TINYCPU is portable, it can be a standard benchmark to measure the performance of FPGAs and logic design tools. Therefore, TINYCPU can be a standard benchmark circuit design to measure the goodness of logic design and synthesis tools as well as the performance of FPGAs.

Education Since our processing system including a processor, a cross assembler, and a cross compiler is simple, they are easy to understand for students. Thus it can be used as course materials for learning basics of computer and embedded systems from various aspects.

Since our main target is to design a processor, a cross assembler, and a cross compiler as simple as possible, The efficiency including used hardware resources and the clock frequency is of secondary importance. Nevertheless, the performance of our TINYCPU is 89MIPS for XC3S700A and 76MIPS for EP3C25F324.

Several processors designed for embedding in FPGAs have been presented. Xilinx Inc. offers 8-bit CPU, Picoblaze [23] and 32-bit CPU Microblaze [25]. Altera Corp. provides 32-bit Nios II processor [3]. They are optimized for their FPGAs and porting to the other FPGAs is not possible. Embedded processors for executing Java Virtual Machine codes on the FPGA have been presented [16, 18], but they are too complicated.

We have also developed a PONG-like mini game [17] and an RSA encryption/decryption system using our tiny processing system. The RSA encryption/decryption system was submitted to the Design Wave Magazine Design Contest 2008 and won vice-champion [6].

2 The Architecture of TINYCPU

TINYCPU is a scalable processor that we have developed. This processor is a pure stack architecture [10] 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 operations and arithmetic and logic operations. In the standard configuration, the word size of TINYCPU is 16 bits. In other words, the width of the data bus and the stack is 16 bits. As we will explain later, the word size can be changed to any integer $N \geq 8$.

TINYCPU is designed using Verilog HDL. The Verilog HDL code is written as simple as possible. Figure 1 illustrate the block diagram of TINYCPU. It has six components including state machine `state`, 12-bit program counter `pc`, 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 11 control instructions and 19 instructions for arithmetic and logic operations. In the table, operands `I` and `A` show 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.

Quite surprisingly, every instruction in Table 2 can be performed in one clock cycle. The function of a dual-port block RAM is used to fetch an instruction and read/write data in the same time. Thus, during the execution of an instruction, TINYCPU fetches next instruction.

We will show how TINYCPU is designed by the Verilog HDL using an example. List 1 shows the Verilog HDL source code `alu.v` for ALU (Arithmetic and Logic Unit). In the first line, Verilog HDL source code `defs.v`, which has miscellaneous mappings of binary constant numbers to instruction codes and states, is included. The word size N can be changed by specifying the parameter value. This module has N-bit integers `a` and `b` as inputs and N-bit integer `s` as output. It also has 5-bit input `f` which is used to select a function out of 19 functions. We assume that `a`, `b` and `s` are signed and 2's complements if they are treated as integers.

List 1. The Verilog HDL source code alu.v for ALU (Arithmetic and Logic Unit)

```
1  `include "defs.v"
2
3  module alu(a, b, f, s);
4  parameter N = 16;
5
6    input [N-1:0] a, b;
7    input [4:0]   f;
8    output [N-1:0] s;
9    reg [N-1:0] s;
10   wire [N-1:0] x,y;
11
12   assign x = {~a[N-1],a[N-2:0]};
13   assign y = {~b[N-1],b[N-2:0]};
14
15   always @(a or b or x or y or f)
16     case(f)
17       `ADD : s = b + a;
18       `SUB : s = b - a;
19       `MUL : s = b * a;
20       `SHL : s = b << a;
21       `SHR : s = b >> a;
22       `BAND: s = b & a;
23       `BOR : s = b | a;
24       `BXOR: s = b ^ a;
25       `AND : s = b && a;
26       `OR : s = b || a;
27       `EQ : s = b == a;
28       `NE : s = b != a;
29       `GE : s = y >= x;
30       `LE : s = y <= x;
31       `GT : s = y > x;
32       `LT : s = y < x;
33       `NEG : s = -a;
34       `BNOT : s = ~a;
35       `NOT : s = !a;
36       default : s = {N{1'bx}};
37     endcase
38
39 endmodule
```

We use two nets `x` and `y` for the technical reasons to follow Verilog-95 standard. In Verilog-95, a bit vector are treated as an unsigned integer. Thus, we use two nets `x` and `y` to obtain correct results of comparisons

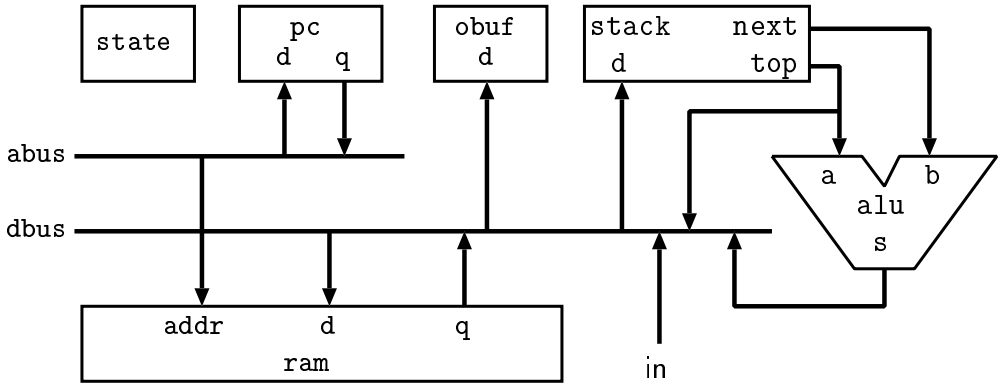

Figure 1. TINYCPU architecture

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	LD	7000	mem[top]→top
9	ST	8000	top→ mem[next]
10	IN	D000	in→top
11	OUT	E000	top→out
12	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

>=, <=, >, and < for signed integers a and b. It should be clear that x = a +2^{15} and y = b +2^{15} hold for 16-bit signed integers a and b and 16-bit unsigned integers x and y. Also, the results of comparisons for unsigned integers x and y is equal to those for signed integers a and b. If we used Verilog 2001 [20] or later, which supports signed integers, we did not have to use two nets x and y. However, since we want to design TINYCPU by Verilog-95 [19] for portability, we use such technical conversion technique. Actually, some logic design, synthesis, and simulation tools give wrong results for comparison of signed integers. For example, Icarus Verilog [9] does not return correct results for comparisons of "signed" nets.

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. The compiler, TINYC, translates a C-based language program into an assembly language program for TINYASM. TINYC language supports 16-bit signed integers and its 1-dimensional array, and if, if-else, while, do, and goto statements. Also, it has basic arithmetic and logic operations including addition (+), subtraction (−), multiplication (∗), negation (−), bit shifts (<<, >>), bitwise logic operations (&, |, ^, ~), logic operations (&&, ||, !), and comparisons (==, ! =, >, >=, <, <=).

List 2 shows an example of TINYC language program collatz.c. Using TINYC compiler and TINYASM assembler, collatz.c is translated into a TINYASM assembly language program and a TINYCPU machine code in List 3. The C-language program in List 2 computes the formula in Collatz conjecture [1, 22] as follows. Consider the following operation on an arbitrary positive integer n:

- If the number is odd, triple it and add one, that is, $n \leftarrow 3n + 1$.
- 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. The readers should have no difficulty to confirm that TINYC program collatz.c correctly repeats the operation of Collatz conjecture until n becomes 1.

List 3 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 2. 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;
```

Surprisingly, the source programs for TINYASM and TINYC are very simple and compact. TINYASM that we have developed has 38 lines using Perl language [21]. We have developed compiler TINYC using the lexical scanner generating tool Flex [15] and the parser generating tool Bison [8]. The source code of TINYC consists of two files: a Flex grammar file that defines how an input C-based language program is converted into a sequence of tokens, and a Bison grammar file that defines how the token sequence are parsed and machine code are generated. Our Flex grammar file has 23 lines, and Bison grammar file has 90 lines (Figure 1).

List 3. The translated assembly language program and machine program of collatz.c for Collatz conjecture

```
*** 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 Scalability of TINYCPU and the Performance Evaluation

The word size of TINYCPU is 16 bits width in its standard configuration, and the data bus and the stack has 16-bit width. The word size of our TINYCPU can be N bits for any integer $N \geq 8$. Users can change the word size by specifying the value of parameter N in the top module of Verilog HDL for TINYCPU.

Table 3 shows the performance of TINYCPU for each word size N. It includes clock frequency, used slices, and 18-bit multipliers in the Xilinx Sparta-3A family FPGA XC3S700A-5, which has 5888 slices and 20 18-bit multipliers. It also shows clock frequency, used logic elements, and 9-bit multipliers in the Altera Cyclon III family FPGA EP3C25F324C6, which has 24624 logic cells and 132 9-bit multipliers. We have used ISE Foundation 9.2i for XC3S700A, and Quartus II 7.2 for EP3C25F324 for logic synthesis and timing analysis. XC3S700A is used in Spartan-3A starter kit [28], which is sold for 189USD, and EP3C25F324 is used in Cyclon III FPGA starter kit [2], which is sold for 199USD. So these FPGAs are in the same price range. We can see that the clock frequencies of these FPGA devises are almost the same. Further, we can compare the hardware resources for these two FPGAs. *A slice* of Sparta-3A family FPGA has two four-input LUTs (Look Up Table), two storage elements, and two multiplexers. *A logic element* of Cyclon III family FPGA contains a four-input LUT, a programmable register, a carry chain connection, and a register chain connection. Thus, a slice of Spartan-3A family FPGA has larger capacity than a logic element of Cyclon III family FPGA. From Table 3, we can confirm that the number of used slices is 60-90% of the number of used logic elements. TINYCPU is portable, and has a lot of fundamental logic components including, registers, counters, memories, combinational circuits for arithmetic and logic operations, state machines, multiplexers, tri-state buses, etc, it can be used as a benchmark circuit design to measure the goodness of logic design and synthesis tools as well as the performance of FPGAs.

5 Implementation in the FPGA board

We have implemented our TINYCPU system in the FPGA boards, Spartan-3E starter kit [24] (Figure 2) and Spartan-3A starter kit [26]. The Spartan-3E and Spartan-3A starter kits FPGA boards are equipped with Spartan-3E family FPGA XC3S500E [27, 29] and Spartan-3A family FPGA XC3S700A [27, 28], respectively. Both FPGA boards have various switches (slide switches, button switches and rotary switches), LEDs, and LCD. They also has ports for VGA, PS/2, and Ethernet.

Figure 2. Spartan-3E Starter Kit

We have implemented our TINYCPU system in these FPGA board to confirm that it works properly. In our implementation, various values including program counter, data bus, address bus, output buffer, etc, are displayed in the LCD display.

We have also developed VGA display, keyboard, and mouse controllers for the FPGA boards of Spartan-3E and Spartan-3A Starter kits. These controllers are connected to TINYCPU and can be operated by TINY-

Table 3. Performance of scalable TINYCPU for various word sizes.

	Xilinx XC3S700A-5			Altera EP3C25F324C6		
bits	clock (MHz)	slices (out of 5888)	18-bit multiplier (out of 20)	clock (MHz)	logic elements (out of 24624)	9-bit multiplier (out of 132)
8	111	181 (3.0%)	1 (5%)	85	191 (0.8%)	1(0.8%)
16	89	336 (5.7%)	1 (5%)	76	569 (2.3%)	2(1.5%)
24	63	480 (8.2%)	3 (15%)	62	849 (3.4%)	6 (4.5%)
32	60	650 (11%)	3 (15%)	60	1096 (4.5%)	6 (4.5%)
40	52	809 (14%)	6 (30%)	48	1404 (5.7%)	12 (9.1%)
48	50	1002 (17%)	6 (30%)	51	1716 (7.0%)	12 (9.1%)
56	44	1202 (20%)	10 (50%)	47	1995 (8.1%)	20 (15%)
64	43	1394 (24%)	10 (50%)	45	2313(9.4%)	20 (15%)

CPU using input/output instructions. We have developed two applications using our tiny processing system including TINYCPU and controllers. The first application is a PONG-like [17] mini game (Figure 3), which is a two-player computer game based on the sport ping pong. This game uses two mice connected to a PS/2 port through a PS/2 splitter. Each player control a racket in the display using a mouse. The second application is an RSA encryption/decryption system (Figure 3), which is a given task for Design Wave Magazine Design Contest [6]. In our RSA encryption/decryption system, plain text is given using a keyboard connected to the FPGA board. The results of the encryption/decryption are exhibited as plain text and hexadecimal in the display. Our system won vice-champion of the contest [6]. These two actual implementation results prove that our tiny processing system can be used for developing a small embedded system.

6 Course Material for Education

We have used degenerated version of our tiny processing system as course materials for embedded system design class of graduate students. This class is organized in 8 weeks of 5 hours each. We have used Spartan-3E Starter Kit (Figure 2) and Spartan-3AN Starter Kit and students implement the tiny processing system in the FPGA board and confirm it works correctly by operating it. The contents of each of the eight weeks as follows: (1) Full Adders, N-bit Adders and ALU, (2) Flip-Flops, Counters, State Machines, and Stacks, (3) Buses and Memories, (4) TINYCPU design, (5) Assembler TINYASM design using Perl language, (6) Flex and Bison, (7) Compiler TINYC design, and (8) TINYC programming. Also, students are required to extend the function of TINYC, TINYASM, and TINYC.

Using this course material, students can learn digital circuit design using HDL, processor architectures, assembler design, assembly language programming, and compiler design. Thus, our processing system can be used as a good course material to learn basics of computer and embedded system by experiment.

7 Concluding Remarks

We have presented a tiny processing system which can be implemented in various FPGAs. This tiny processing system has a lot of advantages and merits including simplicity, scalability, portability, low CPI, and minimum usage of block RAMs. We have implemented processor of the system in Altera and Xilinx FPGAs and evaluate the performance. Also, two applications, PONG-like mini game and RSA encryption/decryption system implemented in the Xilinx FPGA board. We have used the tiny processing system as a course material of embedded system class for graduate students.

We have a lot of future works that we plan to do. First, we extend instructions of TINYCPU to support signal and image processing. We also plan to embed two or more TINYCPU in a FPGA for parallel computation. Since the 16-bit version of TINYCPU uses only 2.3% logic elements of EP3C24F324, it may be possible to embed 40 TINYCPUs into a single FPGA. Further, developing a tiny operating system for TINYCPU is also an interesting research theme.

References

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

[2] Altera Corp. *Cyclon III FPGA Starter kit User Guide*, 2007.

[3] Altera Corp. *Nios II Processor Reference Handbook*, 2008.

Figure 3. PONG-like mini game and RSA encryption/decryption system

[4] J. L. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):803–810, May 2003.

[5] J. L. Bordim, Y. Ito, and K. Nakano. Instance-specific solutions to accelerate the CKY parsing for large context-free grammars. *International Journal on Foundations of Computer Science*, pages 403–416, 2004.

[6] The results of design wave design contest 2008. *Desgin Wave*, 5:124–125, May 2008.

[7] Cygwin. http://www.cygwin.com/.

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

[9] Icarus verilog. http://icarus.com/eda/verilog/.

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

[11] R. Lin, K. Nakano, S. Olariu, M. C. Pinotti, J. L. Schwing, and A. Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Trans. on Parallel and Distributed Systems*, 11(8):838–850, August 2000.

[12] K. Nakano and E. Takamichi. An image retrieval system using FPGAs. *IEICE Transactions on Information and Systems*, E86-D(5):811–818, May 2003.

[13] K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science*, 197:57–77, 1998.

[14] K. Nakano and Y. Yamagishi. Hardware n choose k counters with applications to the partial exhaustive search. *IEICE Trans. on Information & Systems*, 2005.

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

[16] C. Pitter and M. Schoeberl. Towards a Java multiprocessor. In *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems (JTRES 2007)*, pages 144–151, Vienna, Austria, September 2007. ACM Press.

[17] Pong story. http://www.pong-story.com/.

[18] W. Puffitsch and M. Schoeberl. picoJava-II in an fpga. In *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems (JTRES 2007)*, pages 213–221, Vienna, Austria, September 2007. ACM Press.

[19] D. E. Thomas and P. R. Moorby. *The Verilog Hardware Description Language, Fourth Edition*. Kluwer Academic, 1998.

[20] D. E. Thomas and P. R. Moorby. *The Verilog Hardware Description Language, Fifth Edition*. Kluwer Academic, 2002.

[21] L. Wall, T. Christiansen, and J. Orwant. *Programming Perl*. O'Reilly, 2000.

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

[23] Xilix Inc. *Picoblaze: Embedded Micro Controller User Guide*, 2005.

[24] Xilix Inc. *Spartan-3E Starter Kit Board Users Guide*, 2006.

[25] Xilix Inc. *Microblaze Processor Reference Guide*, 2007.

[26] Xilix Inc. *Spartan-3A/3AN Starter Kit Board Users Guide*, 2007.

[27] Xilix Inc. *Spartan-3 Generation FPGA User Guide*, 2008.

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

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