社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

順序回路型 LUT カスケードによる多出力論理関数の実現

シン　キ[†]　笹尾　勤[†,††]　松浦　宗寛[†]　永山　忍[†]　中村　和之[††]
井口　幸洋[†††]

[†] 九州工業大学情報工学部電子情報工学科　〒 820–8502 福岡県飯塚市大字川津 680–4
[††] 九州工業大学マイクロ化総合科学技術センター　〒 820–8502 福岡県飯塚市大字川津 680–4
[†††] 明治大学理工学部情報科学科　〒 214–8571 神奈川県川崎市多摩区東三田 1–1–1
E-mail: [†] {qinhui,nagayama}@aries02.cse.kyutech.ac.jp, [†] {sasao,matsuura}@cse.kyutech.ac.jp,
[††] nakamura@cms.kyutech.ac.jp, [†††] iguchi@cs.meiji.ac.jp

あらまし　本稿では, 順序回路型 LUT(Look Up Table) カスケードによる多出力論理関数の実現について述べる. LUT カスケードは, 新しいプログラマブル・デバイス (PLD: Programmable Logic Device) であり, 組合せ回路型と順序回路型の 2 種類が存在する. 組合せ回路型と順序回路型は異なる利害得失を持ち, 応用分野に応じて使い分けることができる. 多出力論理関数を実現する場合, 順序回路型 LUT カスケードは, 組合せ回路型に比べ, より柔軟に設計可能である. 本稿では, 順序回路型 LUT カスケードのプロトタイプを 0.35μm CMOS テクノロジで設計し, その性能評価を行った. MPU(Micro Processing Unit) を用いたソフトウェア実現との比較実験では, 順序回路型 LUT カスケードは, SH-1(同一周波数換算) 上でのソフトウェア実現に比べ, 77 倍から 229 倍高速であり, PentiumIII(同一周波数換算) 上でのソフトウェア実現に比べ, 35 倍から 88 倍高速であった. また, 現在市販されている 0.18μm CMOS テクノロジの FPGA(Field Programmable Gate Array) で実現したものに近い性能が得られた.

キーワード　LUT カスケード, 多出力論理関数, 再構成可能, プログラマブル・デバイス

Realization of Multiple-Output Functions by Sequential Look-Up Table Cascade

Hui QIN[†], Tsutomu SASAO[†,††], Munehiro MATSUURA[†], Shinobu NAGAYAMA[†], Kazuyuki NAKAMURA[††], and Yukihiro IGUCHI[†††]

[†] Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
[††] Center for Microelectronics Systems, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
[†††] Department of Computer Science, Meiji University
1–1–1, Higasimita, Tamaku, Kawasaki, Kanagawa, 214–8571 Japan
E-mail: [†] {qinhui,nagayama}@aries02.cse.kyutech.ac.jp, [†] {sasao,matsuura}@cse.kyutech.ac.jp,
[††] nakamura@cms.kyutech.ac.jp, [†††] iguchi@cs.meiji.ac.jp

Abstract　A look-up table (LUT) cascade is a new type of a programmable logic device (PLD), which provides an alternative way to realize multiple-output functions. Two types of LUT cascades exist: A combinational LUT cascade, and a sequential LUT cascade. Comparison with a combinational LUT cascade, the sequential LUT cascade is more flexible. A prototype of a sequential LUT cascade has been custom-designed with 0.35μm CMOS technology. Simulation results show that the sequential LUT cascade is 77 to 229 times faster than software programs on SH-1 with the same clock frequency, and 35 to 88 times faster than software programs on PentiumIII with the same clock frequency, but is a little bit slower than the commercial FPGAs.

Key words　LUT cascade, Multiple-output function, Reconfigurable logic, Programmable logic device.

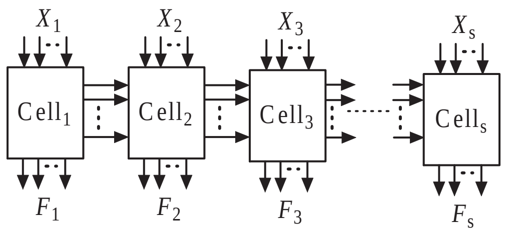


Fig.1 Combinational LUT cascade.

1. Introduction

Programmable logic devices (PLDs) are widely used for prototyping and final products to reduce turn-around time and financial risk. In this paper, we consider a realization of multiple-output logic functions by using PLDs. Various methods exist to realize multiple-output logic functions. Among them, RAMs and programmable logic arrays (PLAs) directly implement logic functions. However, when the number of input variables n is large, the necessary amount of the hardware becomes too large. Thus, field programmable gate arrays (FPGAs) are often used. However, FPGAs require both physical and logic design. Also, without the complete physical design, the prediction of the performance of FPGAs is hard because the area and delay for the interconnections are often much larger than that for logic cells.

A look-up table (LUT) [1] cascade is a new type of a PLD. Since it has a memory-like structure and the interconnections are simple, prediction of the circuit performance is easy. Two types of LUT cascades exist: A combinational LUT cascade, and a sequential LUT cascade.

A combinational LUT cascade shown in Fig. 1 consists of several memories (cells) connected in series to realize a function. Although the combinational LUT cascade is simple and fast, the logical capability is low. Once the number of inputs, outputs of cell, and the number of cells are fixed, the number of realizable functions is limited. Thus, an efficient use of memory is hard.

A sequential LUT cascade shown in Fig. 2 simulates a combinational LUT cascade sequentially using just one memory for logic. Since the number of inputs, the number of outputs of the cell and the number of cells can be changed by using programmable connection network, the sequential LUT cascade can implement wider range of functions. Also, memory-packing [1] technology can be used to reduce total amount of memory.

In this paper, we show a prototype of a sequential LUT cascade custom-designed with 0.35μm CMOS technology, and compare its performance with microprocessors and FPGAs. The rest of the paper is organized as follows: Section 2 explains the architecture and features of the sequential LUT cascade. Section 3 presents the circuit design of the prototype. The experimental results are shown in Section 4, and Section 5 concludes the paper.

2. Sequential LUT Cascade

In this section, we explain the architecture and features of a sequential LUT cascade.

The sequential LUT cascade shown in Fig. 2 consists of five parts: The **Input Reg.** stores the values of primary inputs; the **Output Reg.** stores the values of primary outputs; the **Memory for Logic** stores the LUT data for cells, where all of the LUT data are stored in one memory. The **Control Part** consists of the **Control** block that generates control signals and the **Memory for Interconnections** that stores the information of the interconnections among cells; the **Programmable Connection Network** implements the interconnections among cells.

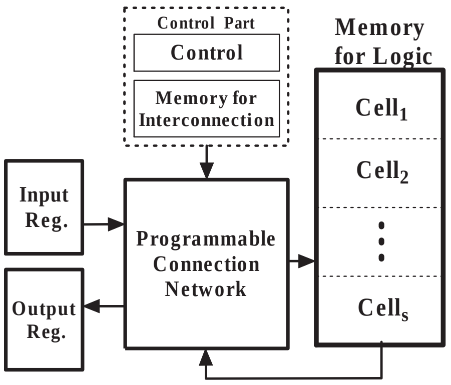


Fig.2 Architecture of sequential LUT cascade.

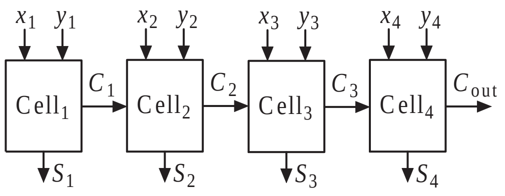


Fig.3 Structure of combinational LUT cascade for 4-bit adder.

A sequential LUT cascade simulates the combinational LUT cascade sequentially. Although it is slower than the combinational one, its logical ability is much higher. In the sequential LUT cascade, the number of inputs, outputs of the cell, and the number of cells can be changed by using programmable connection network.

Compared with the combinational LUT cascade, the sequential LUT cascade has the following features:

- Requires only one memory for LUT data.
- Can implement wide range of functions.
- Can use memory-packing [1] to reduce total amount of memory.
- 30 to 100 times faster than a microprocessor.

[Example 1] Consider a 4-bit adder.

$$\begin{array}{ccccc} & x_4 & x_3 & x_2 & x_1 \\ +) & y_4 & y_3 & y_2 & y_1 \\ \hline \text{Cout} & S_4 & S_3 & S_2 & S_1 \end{array}$$

The adder can be implemented by a combinational LUT cascade with four independent cells as shown in Fig. 3. Note that the number of cells is four.

However, if we use a sequential LUT cascade, we need just one memory for LUT data. Figs. 4 (a)-(d) show the operation of the 4-bit adder by the sequential LUT cascade. Suppose that the memory for logic with five address lines $(A_4, A_3, A_2, A_1, A_0)$, and eight outputs $(D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0)$.

The combinational cascade consists of four cells. In this case, the sequential cascade requires four steps to compute the outputs of the

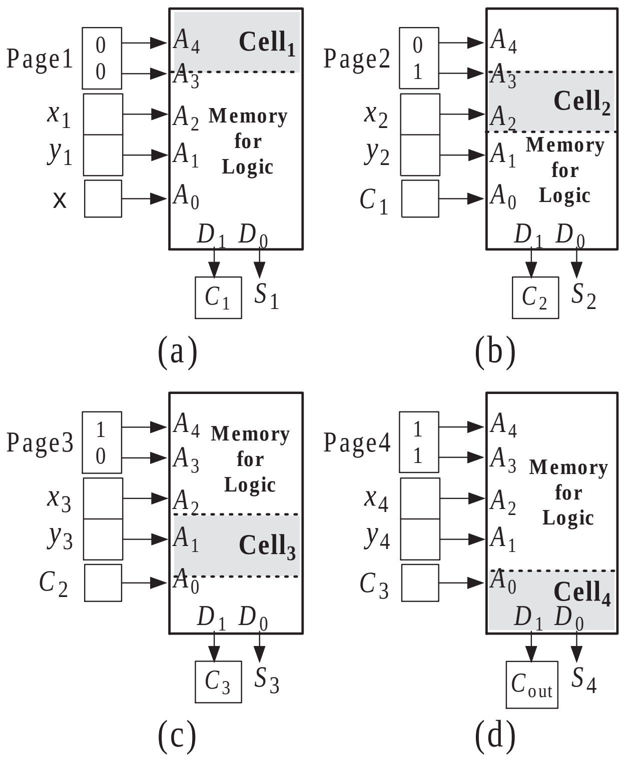


Fig.4 Operation of 4-bit adder.

adder. We partition the memory into four pages, and assume that the two most significant bits of the address denote the page. In the first step, the top two most significant bits are set to (0,0), as shown in Fig. 4(a). This corresponds to the page 1 in the sequential cascade, and the first cell in the combinational cascade. We have to read the memory to obtain C_1 and S_1. Since the first cell has only two inputs, the least significant bit of the address can be either 0 or 1. This operation is symbolically denoted by:

$(A_4,A_3,A_2,A_1,A_0) \leftarrow (0,0,x_1,y_1,\times)$,

where $\times$ denotes either 0 or 1.

After reading the values of (C_1,S_1), C_1 is transferred to the least significant bits of the address for the next lookup. At the same time, S_1 is set to the least significant bit of the output register. This wiring is done by the programmable connection network in Fig. 2. The information for the connection for this step is stored in the memory for interconnections. This operation is symbolically denoted by:

Read (D_1,D_0), and let $(C_1,S_1) \leftarrow (D_1,D_0)$.

Let OUT_REG [0] $\leftarrow S_1$.

In the second step, the 2nd page is used to obtain (C_2,S_2) as shown in Fig. 4(b). Symbolically, the operation is denoted by:

$(A_4,A_3,A_2,A_1,A_0) \leftarrow (0,1,x_2,y_2,C_1)$.

Read (D_1,D_0), and let $(C_2,S_2) \leftarrow (D_1,D_0)$.

Let OUT_REG [1] $\leftarrow (S_2)$.

In the third step, the 3rd page is used to obtain (C_3,S_3) as shown in Fig. 4(c). Symbolically, the operation is denoted by:

$(A_4,A_3,A_2,A_1,A_0) \leftarrow (1,0,x_3,y_3,C_2)$.

Read (D_1,D_0), and let $(C_3,S_3) \leftarrow (D_1,D_0)$.

Let OUT_REG [2] $\leftarrow (S_3)$.

In the last step, the 4th page is used to obtain (C_{out},S_4) as shown in Fig. 4(d). Symbolically, the operation is denoted by:

$(A_4,A_3,A_2,A_1,A_0) \leftarrow (1,1,x_4,y_4,C_3)$.

Read (D_1,D_0), and let $(C_{out},S_4) \leftarrow (D_1,D_0)$.

Let OUT_REG [4:3] $\leftarrow (C_{out},S_4)$.

Note that it simulates the combinational LUT cascade in Fig. 3. In this way, the 4-bit adder is evaluated by accessing the memory four times. (End of Example)

3. Circuit Design

This section presents a transistor-level design using 0.35μm 3.3V CMOS technology, and evaluates the delay of the prototype using circuit simulator **SPICE**.

3.1 Circuit Design of the Prototype

The specifications of the prototype sequential LUT cascade are as follows:

- The number of external inputs is at most 32.
- The number of outputs is at most 24.
- The number of cells of LUT cascades, s is at most 8.
- The maximum number of inputs for each cell, k is 13. Each cell may have different number of inputs.
- The maximum number of outputs for each cell is 8.
- It can use memory-packing to reduce total amount of memory.

Fig. 5 shows the architecture of the prototype. It consists of the following components:

Input Reg.: 32-bit, stores the values of primary inputs.

Output Reg.: 24-bit, stores the values of primary outputs.

MAR: 13-bit memory address register, stores the values of intermediate address.

64K-bit SRAM: Stores the LUT data for cells. 13 inputs and 8 outputs.

Control Part: Includes two blocks. The RAM stores the information of the interconnections among cells, and the control block consists of one counter and several logic gates that generate the control signals for the LUT cascade.

Shifter Network: Consists of several barrel shifters and a 32-bit register that stores the values of intermediate outputs of cells. It implements the programmable interconnections among cells.

The prototype has two important parts: The 64K-bit SRAM and the shifter network. Since the SRAM should operate as a data path in the cascade mode, we use an asynchronous SRAM. Because design methods of SRAMs are described in literature [2], we just show the circuit design of the barrel shifter. The delay of an n-bit shifter is proportional to $\log n$, so combined with the fast transmission gates, shift can be fast [2]. Furthermore, we reduced chip area by using a single n-channel pass transistor instead of a full transmission gate. Fig. 6 shows the detail design of a 4-by-4 barrel shifter.

3.2 Operation Modes in Cascades

The sequential LUT cascade has two modes: The *configuration mode* and the *cascade mode*. In the configuration mode, all of the address lines of two RAMs are directly connected to the inputs through two multiplexers. 8-bit data input lines which connected to **DIN** in Fig. 5 are used to store LUT data for cells into the 64K-bit SRAM, while another 28-bit data input lines are used to store the information of interconnection into the RAM in the control part. In the cascade mode all the address lines of 64K-bit SRAM are connected to the MAR, while the address lines of the RAM in the control part are connected to the control block, and both RAMs are in

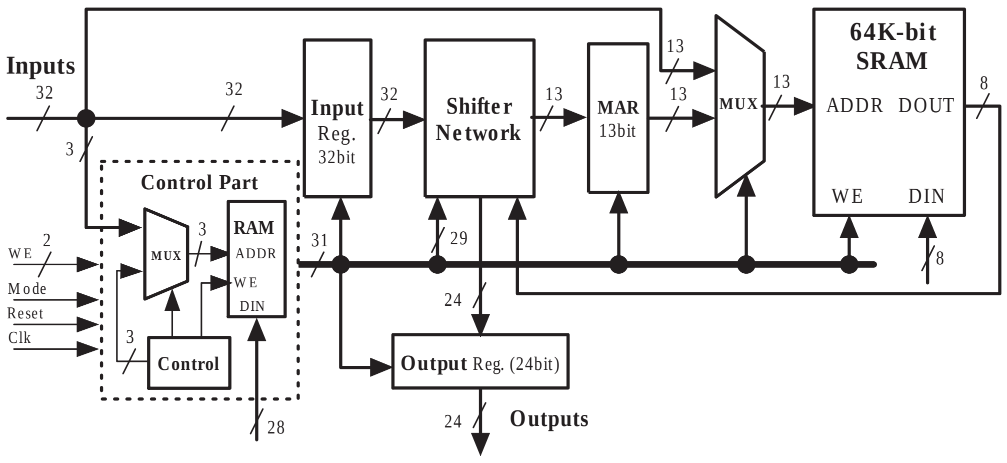


Fig.5 Architecture of the prototype.

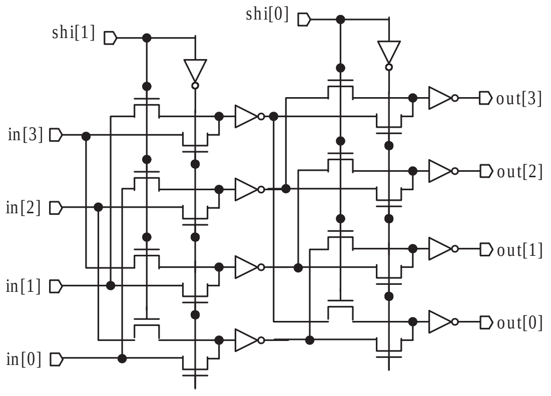
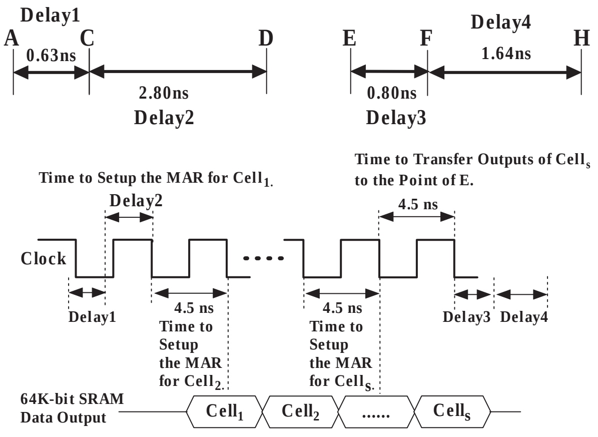


Fig.6 Details of a 4-by-4 barrel shifter.



Delay1
A
C
D
E
F
H
Delay4
0.63ns
2.80ns
Delay2
0.80ns
Delay3
1.64ns
Time to Setup the MAR for Cell1.
Time to Transfer Outputs of Cells to the Point of E.
Delay2
4.5 ns
Clock
Delay1
4.5 ns Time to Setup the MAR for Cell2.
4.5 ns Time to Setup the MAR for Cells.
Delay3
Delay4
64K-bit SRAM Data Output
Cell1
Cell2
......
Cells

Fig.8 Delay time in the signal path.

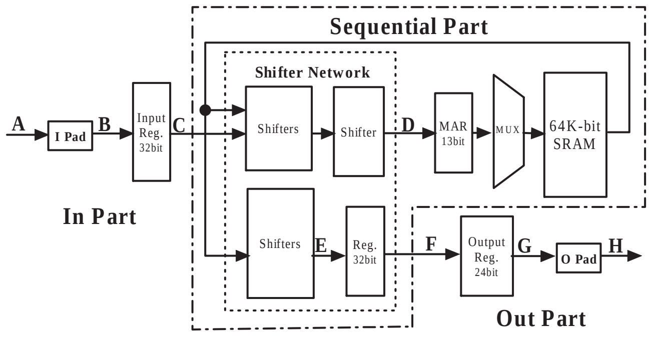


Fig.7 Signal path of the prototype.

the READ operation.

To use the LUT cascade, first, the chip is set to the configuration mode: We need to store LUT data into the 64K-bit SRAM and store the interconnection information into the memory for interconnection. Then, the chip is set to the cascade mode. Finally, we can get the desired result after the evaluation time.

3.3 Operation Speed of the Prototype

To obtain the operation speed of the sequential cascade, we partition the sequential LUT cascade into three parts as shown in Fig. 7. The **In Part** consists of input pad (I Pad) and input register, the **Out Part** consists of output pad (O Pad) and output register, and the others are represented as the **Sequential Part** dotted line. Fig. 8 shows the delay time in the signal path, where the values are obtained by SPICE simulation for a 0.35μm, 3.3V CMOS process.

In Fig. 8, the delay for **In Part** is denoted by Delay1, and the delay for **Out Part** is denoted by Delay4. The delay for the **sequential part** depends on the number of clocks. In the prototype, during one clock cycle we can access the 64K-bit SRAM once. Note that before the memory access, we need to setup the MAR with the values of cell address. The time to setup the MAR for the first cell is denoted by Delay2, which is shorter than the time for the rest of the cells because its path is C to D. The time to setup the MAR for the other cells is equal to one clock cycle. Thus, when the combinational cascade consists of *s* cells, the time to setup the MAR for cells is equal to **(Delay2 + (*s*-1) * CLK)**, where CLK is 4.5 ns. However, we also need one clock cycle to transfer the outputs of the last cell to the point E. And then after Delay3, the primary outputs are out of the **sequential part**. Therefore, the delay time for the **sequential part** is equal to **(Delay2 + *s* * CLK + Delay3)**. And the total delay of the sequential LUT cascade is obtained by the following:

$$\begin{aligned} \text{Delay} &= \text{Delay1} + \text{Delay2} + s * \text{CLK} + \text{Delay3} + \text{Delay4} \\ &= 0.63 + 2.8 + 4.5 * s + 0.8 + 1.64 \\ &= 4.5 * s + 5.9 \text{ (ns)} \qquad (1) \end{aligned}$$

［Example 2］ Consider the 4-bit adder in Example 2.1, where *s* is

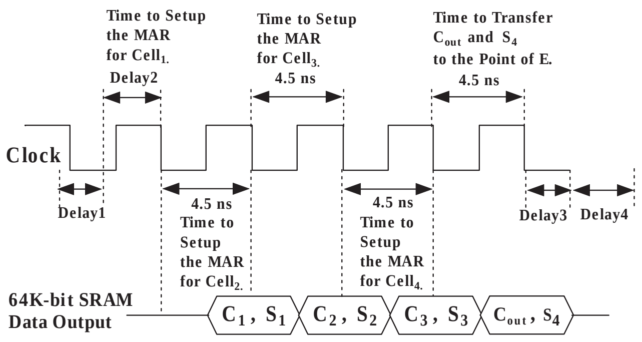

Time to Setup the MAR for Cell$_1$.
Delay2
Time to Setup the MAR for Cell$_3$.
4.5 ns
Time to Transfer C_{out} and S_4 to the Point of E.
4.5 ns
Clock
Delay1
4.5 ns Time to Setup the MAR for Cell$_2$.
4.5 ns Time to Setup the MAR for Cell$_4$.
Delay3
Delay4
64K-bit SRAM Data Output
C_1, S_1
C_2, S_2
C_3, S_3
C_{out}, S_4

Fig.9 Delay time for the 4-bit adder.

4. The evaluation time is $4.5 \times 4 + 5.9$ (ns).

Fig. 9 shows the distribution of the evaluation time for the 4-bit adder. First, after Delay1, the values of $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ are stored into the Input Reg. Second, before accessing the 64K-bit SRAM, we need to spend Delay2 to setup the MAR with the values of $(0,0,0,x_1,y_1,\times,\times,\times,\times,\times,\times,\times,\times)$ for $Cell_1$, where $\times$ denotes either 0 or 1. Then, to obtain C_2 and S_2, we need to spend one clock cycle to setup the MAR with the values of $(0,0,1,x_2,y_2,C_1,\times,\times,\times,\times,\times,\times,\times)$ for $Cell_2$, meanwhile S_1 is transferred to the point of E. During the next clock, the values of $(0,1,0,x_3,y_3,C_2,\times,\times,\times,\times,\times,\times,\times)$ have to be sent to the MAR for $Cell_3$ to obtain C_3 and S_3, meanwhile S_1 is stored in the 32-bit register, and S_2 is transferred to the point of E. Similarly, to obtain C_{out} and S_4, the values of $(0,1,1,x_4,y_4,C_3,\times,\times,\times,\times,\times,\times,\times)$ have to be sent to the MAR for $Cell_4$, meanwhile S_2 is stored in the 32-bit register, and S_3 is transferred to the point of E. During the next clock, S_3 is stored in the 32-bit register, then C_{out} and S_4 are transferred to the point of E. During Delay3, both C_{out} and S_4 are stored in the 32-bit register, and then the values of $(S_1,S_2,S_3,C_{out},S_4)$ are transferred to the output register. Finally, we can get the evaluated results after Delay4.

(End of Example)

4. Experimental Results

We evaluated the performance of the prototype, and compared it with microprocessors and FPGAs. To make the argument simple, we selected functions from MCNC combinational benchmark set [3].

Table 1 compares the performance of the prototype with two microprocessors and commercial FPGAs. In this table, "Name", "In", and "Out" denote the function name, the number of inputs, and the number of outputs, respectively. The column "s" denotes the number of cells in the LUT cascade. To obtain the number of cells, we used the newly developed logic synthesis tool [4], where the number of inputs for each cell k is set to 10. The column "Time" denotes the evaluation time for the prototype estimated by the equation 1 in Section 3. 3, in nano second.

4. 1 Comparison with Microprocessors

At least two approaches exist to implement software for a combinational benchmark functions. The first approach is to simulate the multi-level combinational circuit by some logic simulator. The second approach is to represent the function by a binary decision diagram (BDD), and then traverse the BDD by a special program [5], [6]. In this experiment, we used the second approach, since it was faster than the first approach for the benchmark functions. We represented benchmark functions using binary decision diagrams (BDDs) for characteristic functions (CFs). The numbers of nodes in BDDs for CFs were reduced by using sifting algorithm [7]. Then, we generated a table that represents the BDD, and we used a special program to traverse the BDD data.

To compare the performance of the prototype with the speed for the software programs, we used RISC microprocessor SH-1 (SH7020) 20MHz [8] and PentiumIII 1GHz with 256KB cache [9]. SH-1 is an embedded MPU used in DVDs, navigation systems, digital cameras, etc. On the other hand, PentiumIII is a high-performance CPU for desktop PCs. For SH-1, we compiled the software program using SH C/C++-compiler [10] with the optimization option for speed, and obtained the CPU time using SH simulator [11]. For PentiumIII, we compiled the software program using GNU C-compiler gcc with -O2 option, and obtained the CPU time by executing it on Linux with 4GB memory.

In Table 1, the column "SH-1" denotes the average CPU time per test vector for the software program on SH-1, in micro second. When the number of inputs for benchmark function is smaller than 17, we obtained the average CPU time for the benchmark function using the exhaustive test (i.e., 2^n test vectors, where n is the number of inputs). When the number of inputs for benchmark function is larger than or equal to 17, we obtained the average CPU time for the benchmark function using 1,000,000 random test vectors. Similarly, the column "PenIII" denotes the average CPU time per test vector for the software program on PentiumIII, in micro second. PentiumIII was too fast to obtain the average CPU time accurately using small number of test vectors. Thus, for all benchmark functions, we used 1,000,000 random test vectors. In the last three columns, the column "SH-1" denotes the relative speed of LUT cascades to the speed of SH-1, where the speed of SH-1 with 222MHz is set to 1. Similarly, the column "PenIII" denotes the relative speed of LUT cascades to the speed of PentiumIII, where the speed of PentiumIII with 222MHz is set to 1. That is, they are calculated by

$$\text{LUT/SH-1} = \frac{\text{SH-1 time} \times 20\text{MHz}/222\text{MHz}}{\text{LUT cascade time}},$$

$$\text{LUT/PenIII} = \frac{\text{PenIII time} \times 1\text{GHz}/222\text{MHz}}{\text{LUT cascade time}},$$

Note that 222MHz is the clock frequency for the LUT cascades.

Table 1 shows that the sequential LUT cascades are 77 to 229 times faster than SH-1, and 35 to 88 times faster than PentiumIII when the clock frequencies for LUT cascade and MPUs are equal.

4. 2 Comparison with FPGAs

The same MCNC benchmark functions were implemented by FPGAs. Firstly, each benchmark function was optimized by SIS tool [12] with *script.algebraic* and then it was converted to Verilog HDL source. Secondly, by Synplify Pro (version: 7.3.3) [13] it was optimized for Altera EP20K30EFC144-1 FPGA device (1.8-V, 0.18μm) [14]. Finally, Quartus (version 2000.09) [14] was used to map it into the FPGA.

In Table 1, the column "LEs" denotes the number of logic elements actually used in the FPGA. The column "Delay" denotes the delay time for each benchmark function obtained by Quartus, in

Table1 Comparison of sequential LUT cascades with MPUs and FPGA.

			LUT cascades		SH-1	PenIII	FPGA		Relative speed of LUT		
Name	In	Out	s	Time [ns]	[μs]	[μs]	LEs	Delay [ns]	SH-1	PenIII	FPGA
misex2	25	18	5	28.4	41.9	0.36	35	13.2	133	57	0.46
misex3	14	14	4	23.9	38.6	0.30	263	23.4	146	57	0.98
mlp6	12	12	7	37.4	42.0	0.34	1074	29.0	101	41	0.78
in4	32	20	8	41.9	45.6	0.39	109	17.3	98	42	0.41
chkn	29	7	5	28.4	24.3	0.22	178	24.8	77	35	0.87
b3	32	20	7	37.4	45.3	0.37	103	18.4	109	45	0.49
b2	16	17	6	32.9	43.0	0.33	216	25.9	118	45	0.79
amd	14	24	7	37.4	54.6	0.43	107	16.5	132	52	0.44
apex4	9	19	4	23.9	49.0	0.39	820	23.5	185	74	0.98
bc0	26	11	5	28.4	31.9	0.25	417	24.0	101	40	0.85
intb	15	7	3	19.4	32.3	0.25	393	36.2	150	58	1.87
prom2	9	21	3	19.4	49.4	0.38	764	25.1	229	88	1.29
tial	14	8	3	19.4	34.0	0.27	357	20.9	158	63	1.08
p1	8	18	6	32.9	46.2	0.38	91	14.0	127	52	0.43
m4	8	16	3	19.4	39.8	0.30	204	19.9	185	70	1.03
x9dn	27	7	5	28.4	28.2	0.26	31	15.9	89	41	0.56
gary	15	11	4	23.9	29.8	0.24	181	20.4	112	45	0.85
exam	10	10	2	14.9	34.3	0.26	148	18.8	207	79	1.26
t2	17	16	5	28.4	42.3	0.33	67	14.4	134	52	0.51

nano second. In the last three columns, the column "FPGA" denotes the relative speed of LUT cascades to the speed of FPGA, where the speed of FPGA is set to 1. That is, it is calculated by

$$\text{LUT/FPGA} = \frac{\text{Delay time of FPGA}}{\text{LUT cascade time}}.$$

Table 1 shows that the prototype is a little bit slower than the FPGAs for many functions. Note that FPGA uses 0.18 μm CMOS technology, while the LUT cascade uses 0.35 μm CMOS technology. In spite of the advantages of process technology in FPGAs, the sequential LUT cascade gives a competitive performance to the FPGAs.

5. Conclusions

In this paper, we have shown a realization of multiple-output functions by a sequential LUT cascade. A prototype of a sequential LUT cascade has been custom-designed by using 0.35μm CMOS technology. Our experiment results show that the sequential LUT cascades is 77 to 229 times faster than software programs on SH-1 with the same clock frequency as the LUT cascade, and 35 to 88 times faster than software programs on PentiumIII with the same clock frequency, but is a little bit slower than the commercial FPGAs.

Acknowledgments

This research is partly supported by JSPS, the Grant in Aid for Scientific Research, and MEXT, the Kitakyushu area innovative cluster project.

References

[1] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic and Synthesis(IWLS01)*, Lake Tahoe, CA, June 12-15, 2001, pp.225-230.

[2] Neil H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective (second edition),* , Addision-wesley publishing company 1994.

[3] MCNC-Benchmark set: http://www.cbl.ncsu.edu/www

[4] T. Sasao and M. Matsuura "A method to decompose multiple-output logic functions," (in Japanese). *Technical report of IEICE,* VLD2003-108, pp. 229-234, Nov. 28, 2003.

[5] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD'95,* pp. 402-407, Nov. 1995.

[6] S. Nagayama and T. Sasao, "Code generation for embedded systems using heterogeneous MDDs," *the 12th workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2003)*, pp. 258-264, Hirosima, Japan, April 3-4, 2003.

[7] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD'93*, pp. 42–47.

[8] Hitachi SuperH 32-bit RISC CPU SH-1 (SH7020), Renesas Technology Co., http://www.renesas.com/eng/products/mpumcu/32bit/sh/.

[9] Intel Pentium III Processor, Intel Co., http://www.intel.com/products/desktop/processors/pentiumiii/.

[10] Hitachi Embedded Workshop (HEW), SuperH RISC Engine C/C++ Compiler Package Ver. 6.0Ar2, Hitachi ULSI Systems Co., http://www.hitachi-ul.co.jp/MYICE/XSOFT/.

[11] Hitachi Debugging Interface (HDI) for SH Series Simulator Ver. 5.01, Hitachi ULSI Systems Co., http://www.hitachi-ul.co.jp/MYICE/XSOFT/.

[12] E. M. Sentovich et. al. " SIS: A System for Sequential Circuit Synthesis," Dept. of Electrical Engineering and Computer Science, University of California, Berkeley,CA 94720, 1992.

[13] http://www.synplicity.com

[14] http://www.altera.com