Implementations of Reconfigurable Logic Arrays on FPGAs

Tsutomu Sasao and Hiroki Nakahara
Department of Computer Science and Electronics,
Kyushu Institute of Technology,
Iizuka 820-8502, Japan

Abstract

This paper presents a method to implement a reconfigurable logic array on an FPGA. To design circuits with 2-valued k-input LUTs, 2^k-valued logic is introduced. Standard benchmark functions as well as symmetric functions are efficiently implemented by a logic array with 2^k-valued variables. Number of products and number of bits to represent functions by the expressions with 2^k-valued variables for $k = 1, 2, 3, 4$, and 5 are compared. Both sum-of-products expressions and EXOR sum-of-products expressions of 2^k-valued logic significantly reduces needed FPGA resources, when $2 \leq k \leq 5$. Experimental results for benchmark functions and symmetric functions are shown. Implementations of arrays with 16-valued variables on Xilinx and Altera FPGAs are also shown.

1. Introduction

In this paper, we consider a method to implement a reconfigurable logic array on an FPGA. It is similar to a programmable logic array, but logically more powerful. We use 2^k-valued logic to represent a binary logic circuit. We present two different realizations on FPGAs: combinational one and sequential one. The combinational one uses configurable logic blocks (CLBs), while the sequential one uses M4Ks and logic elements (LEs). Such methods are promising for dynamically reconfigurable circuits [2, 4, 6, 15].

We also consider the optimal number k of the inputs for LUTs to implement various logic functions. This paper is organized as follows: Section 2 presents two methods to implement expressions with multi-valued variables by FPGAs. Section 3 introduces expressions with multi-valued variables. Section 4 considers the complexities of expressions. Section 5 shows the experimental results. And, finally, Section 6 concludes the paper.

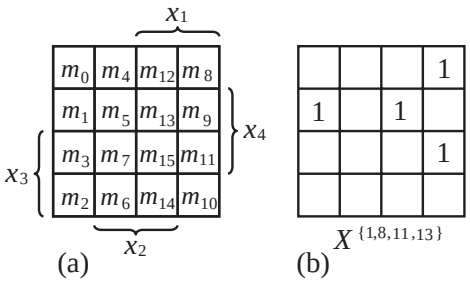

Figure 2.1. Map for 4-variable function.

2. Realization of Expressions with 2^k-Valued Variables on an FPGA

In this part, we introduce logic arrays with 2^k-valued variables, which can be more efficient than conventional logic arrays. An FPGA contains many look-up tables (*LUTs*). Here, we introduce 16-valued logic to design 4-input LUT circuits. A 4-input LUT realizes an arbitrary function of four variables. Fig. 2.1 (a) shows a map of a 4-variable function. An arbitrary 4-variable logic function can be viewed as a subset of 16 minterms $\{m_0, m_1, \ldots, m_{15}\}$. For example, the function in Fig. 2.1 (b) can be represented by a set of four minterms $\{m_1, m_8, m_{11}, m_{13}\}$. Instead of using a set of minterms, we can use a 16-valued literal. Let four variables be treated together as $X = (x_1, x_2, x_3, x_4)$. Then, X is considered as a 16-valued variable, and takes one of 16 values $\{0, 1, \ldots, 14, 15\}$. In this case, the function in Fig. 2.1 (b) can be represented by the literal $X^{\{1,8,11,13\}}$: It shows that the function is 1 if and only if the input combination $X = (x_1, x_2, x_3, x_4)$ represents either 1, 8, 11, or 13.

2.1 Using CLBs Only

Consider the case of a Xilinx FPGA [17]. In this FPGA, a multiplexer (MUX) is attached to the output of each LUT. By using multiplexers, the logical AND of 4-variable functions can be implemented. For example, as shown

1-4244-1472-5/07/$25.00 © 2007 IEEE

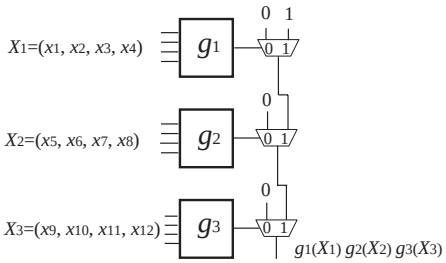

Figure 2.2. Realization of AND with MUXs.

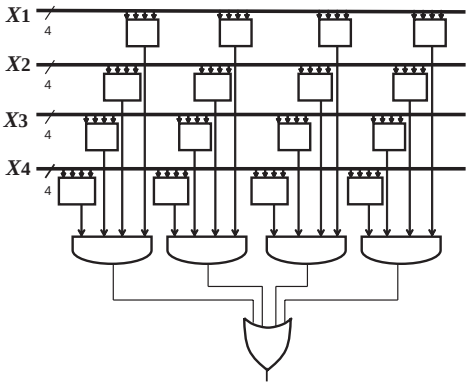

Figure 2.3. Realization of an SOP with 16-valued variables.

in Fig. 2.2, three LUTs are connected by the MUX chain to realize the logical product $g_1(X_1)g_2(X_2)g_3(X_3)$, where $X_1 = (x_1, x_2, x_3, x_4), X_2 = (x_5, x_6, x_7, x_8)$, and $X_3 = (x_9, x_{10}, x_{11}, x_{12})$. Thus, Fig. 2.2 implements a product of 16-valued literals of the form $X_1^{S_1}X_2^{S_2}X_3^{S_3}$, where $S_i \subseteq P$ and $P = \{0, 1, 2, \ldots, 15\}$. Fig. 2.3 shows a circuit for a sum-of-products expression (SOP), where an OR gate is implemented by LUTs. When the OR gate is replaced by an EXOR gate, we have a circuit for an EXOR sum-of-products expression (ESOP). In Fig. 2.3, thick horizontal lines denote bundles of four binary lines. Each horizontal bundle can be viewed as carrying a 16-valued variable, and each column realizes a product of up to four 16-valued literals.

To realize a multiple-output function, a programmable OR (EXOR) gate can be used for each output.

2.2 Using both Embedded RAMs and Logic Elements

Modern FPGAs contain many embedded RAMs as well as logic elements (LEs). For example, Altera's FPGAs have embedded RAMs, called M4K [1]. It can be used as a 7-input 32-output memory, which is equivalent to 32 copies of 7-input LUTs. However, as will be shown later, 7-input LUTs are inefficient to implement a logic array. So, we use M4K as $32 \times 8 = 256$ copies of 4-input LUTs by using a sequential method. Fig. 2.4 shows a sequential implementation of 32 products having forms $X_0^{S_0}X_1^{S_1}\cdots X_7^{S_7}$ by using an M4K. For each clock, each literal $X_i^{S_i}$ is evaluated, and after 8 clocks, products of 8 literals are evaluated. The *MPX* signal consisting of 3 bits specifies i, and the multiplexer selects X_i.

Fig. 2.5 shows a sequential realization of an SOP with 32 inputs, 8 outputs, and $8 \times 32 = 256$ products. Products are implemented by 8 copies of M4Ks and many LEs. Eight copies 256-input OR (EXOR) gates are implemented by LEs. (The details are shown in Subsection 5.2.) In this paper, the architectures shown in Figs. 2.3 and 2.5 are called **reconfigurable logic arrays with 16-valued variables**. In similar ways, reconfigurable logic arrays with 2^k-valued variables can be implemented using k-input LUTs.

3. Expressions with Multi-Valued Inputs [12]

To show a design method, we need some definitions. They are extensions of 2-valued cases to multi-valued cases [12, 8].

Definition 3.1 *A mapping $f : P^n \rightarrow B$ is a p-valued input 2-valued output function, where $P = \{0, 1, \ldots, p-1\}$ and $B = \{0, 1\}$. Let X be a variable that takes its value from $P = \{0, 1, \ldots, p-1\}$. Let S be a subset ($S \subseteq P$) of P. Then, X^S is a* **literal** *of X. When $X \in S$, $X^S = 1$, and when $X \notin S$, $X^S = 0$. Let $S_i \subseteq P$ ($i = 1, 2, \ldots, n$), then $X_1^{S_1}X_2^{S_2}\cdots X_n^{S_n}$ is a* **logical product**. *$\bigvee_{(S_1,S_2,\ldots,S_n)} X_1^{S_1}X_2^{S_2}\cdots X_n^{S_n}$ is a* **sum-of-products expression** *(**SOP**). When $S_i = P$, $X_i^{S_i} = 1$ and the logical product is independent of X_i. In this case, literal X_i^P is redundant and can be deleted. A logical product is also called a* **term**, *or a* **product term**. *When $|S_i| = 1$ for ($i = 1, 2, \ldots, n$), a logical product corresponds to an element of the domain. This product is a* **minterm**. *When $S_i = P$ for ($i = 1, 2, \ldots, n$), the logical product corresponds to the constant 1.*

When $p = 2$, a function is a 2-valued logic function. When we consider 2-valued logic functions only, we often represent the literal $X^{\{0\}}$ by $\overline{X}$, and $X^{\{1\}}$ by X. In an SOP, replacing the OR operators with the EXOR operators will produce an **EXOR sum-of products expression** *(**ESOP**). X_i is often called as a* **part**.

An arbitrary multi-valued input 2-valued output function is represented by an SOP (ESOP). Many SOPs (ESOPs) exist that represent the same function. Among them, the one with the minimum number of products is the **minimum SOP** (**minimum ESOP**). MINI [3] and ESPRESSO-MV [8] are SOP minimizers, while EXMIN2 [12] and EXORCISM-MV [14] are ESOP minimizers.

Interesting problems are

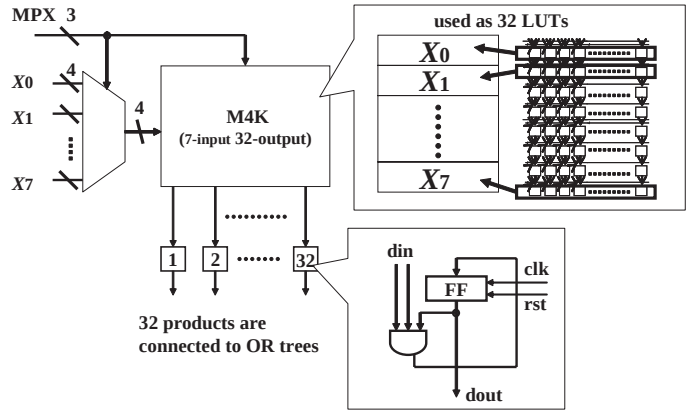

Figure 2.4. Sequential realization of products with 16-valued variable.

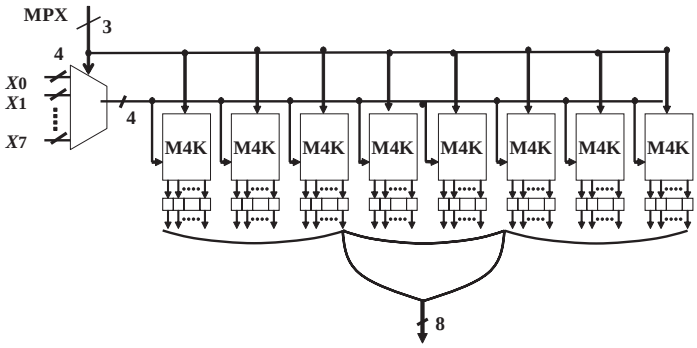

Figure 2.5. Sequential realization of SOP with 16-valued variables.

1. Find the optimum value for k, the number of binary variables in a group.

2. Find the partition of the input variables [13].

The first problem is similar to the problem of finding the optimum size of LUTs in FPGAs [7, 5]. In this paper, we focus on the first problem.

4. Expressions with 2^k-Valued Variables

4.1 Number of Products

The number of products to represent a function depends on n, the number of inputs, and k, the number of variables in parts. In general, the larger the value of k, the smaller the number of products to represent the function.

Theorem 4.1 *An arbitrary function of $n = kr$ variables can be represented by an SOP (ESOP) with 2^k-valued variables using at most 2^{n-k} products.*

Theorem 4.2 *For any expression with 2-valued variables, there exist an expression with 4-valued variables that represents the same function as 2-valued expression, and that requires not more products than 2-valued one.*

Expressions with multi-valued variables efficiently represent symmetric functions.

Theorem 4.3 *Consider a function $f(X_1, X_2, \ldots, X_r)$, where X_i consists of k binary variables. Let $f(X_1, X_2, \ldots, X_r)$ be partially symmetric with respect to X_i for $i = 2, \ldots, r$. That is, f is invariant under the permutation of variables in X_i. Then, f can be represented by an SOP (ESOP) with 2^k-valued variables using at most $(k+1)^{r-1}$ products.*

4.2 Number of Bits

SOPs or ESOPs can be represented by the positional cube notation [12]. To represent a *part* with a k-valued variable, 2^k bits are used. The amount of memory to represent an expression is estimated by the number of the bits for the positional cubes, since the LUTs in Fig. 2.3 store these bit patterns.

In the expression, each part takes 2^k values, and there are r parts, so the number of bits to represent a product $X_1^{S_1} X_2^{S_2} \cdots X_r^{S_r}$ is $r2^k$.

Definition 4.1 *Let $\mu(k)$ be the number of bits to represent a function $f(X_1, X_2, \ldots, X_r)$ by an expression, where X_i*

consists of k binary variables, and let $p(k)$ be the number of products in the expression. For m-output function, $\mu(k) = p(k)(\frac{n}{k}2^k + m)$, where $n = kr$. In the case of a single-output function, we can omit the programmable OR part by generating null products[1]*. Thus, $\mu(k) = p(k)r2^k = p(k)\frac{n}{k}2^k$, for single output function.*

A variable with a large k requires more bits to represent a part than a variable with a small k. On the other hand, variables with large k often require fewer products than variables with small k. Thus, for each function, there is an optimum k that minimizes the total number of bits.

Example 4.1 *Consider the function shown in Fig. 2.1.*
When $k = 1$ (2-valued variables)
The function can be represented as $f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3x_4$. Thus, the number of products is four. The positional cubes are

$$\begin{array}{c} x_1 \quad x_2 \quad x_3 \quad x_4 \\ 01 \quad 01 \quad 01 \quad 01 \\ \left[\begin{array}{cccccccc} 10 & - & 10 & - & 10 & - & 01 \\ 01 & - & 10 & - & 10 & - & 10 \\ 01 & - & 10 & - & 01 & - & 01 \\ 01 & - & 01 & - & 10 & - & 01 \end{array}\right] \end{array}$$

Thus, the number of bits is $8 \times 4 = 32$.
When $k = 2$ (4-valued variables)
Let $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4)$. The function can be represented as $f(X_1, X_2) = X_1^0X_2^1 \vee X_1^2X_2^0 \vee X_1^2X_2^3 \vee X_1^3X_2^1 = X_1^{\{0,3\}}X_2^1 \vee X_1^2X_2^{\{0,3\}}$. Thus, the number of products is two. The positional cubes are

$$\begin{array}{c} X_1 \quad X_2 \\ 0123 \quad 0123 \\ \left[\begin{array}{ccccc} 1001 & - & 0100 \\ 0010 & - & 1001 \end{array}\right] \end{array}$$

Thus, the number of bits is $8 \times 2 = 16$.
When $k = 4$ (16-valued variable)
Let $X_1 = (x_1, x_2, x_3, x_4)$. The function can be represented as $f(X_1) = X_1^1 \vee X_1^8 \vee X_1^{11} \vee X_1^{13} = X_1^{\{1,8,11,13\}}$. Thus, the number of products is just one. The positional cube is

$$\begin{array}{c} X_1 \\ 0000000000111111 \\ 0123456789012345 \\ \left[0100000010010100\right] \end{array}$$

Thus, the number of bits is $16 \times 1 = 16$. Note that this is the single-memory realization of the logic function. (End of Example)

Theorem 4.4 *For any expression with 2-valued variables, there exist an expression with 4-valued variables that represents the same function as 2-valued expression, and that requires not more bits than 2-valued one.*

[1] A null product can be generated by X^ϕ

The above theorem shows that to minimize the total number of bits, we have only to consider the case of $k \geq 2$.

In the next section, we confirm this observation.

5. Experimental Results

5.1 Logic Synthesis

We minimized standard PLA benchmarks [16] as well as adders and symmetric functions.

Standard Benchmark Functions

In Table 5.1, *In* denotes the number of inputs; *Out* denotes the number of outputs; *SOP* denotes the number of products in a sum-of-products expression; *ESOP* denotes the number of products in an EXOR sum-of-products expression. *2^k-valued* denotes the number of products in an expression with 2^k-valued variables. To derive 2^k-valued variables, k binary variables are grouped. To obtain 4-valued and 16-valued expressions, Algorithm 6.1 in [13] was used. To obtain 8-valued and 32-valued expressions, a greedy method was used. For minimization of SOPs, MINI2 [12] was used. For minimization of ESOPs, EXMIN3, an improved version of EXMIN2 [12], was used. Table 5.1 shows that expressions with larger k require fewer products than expressions with smaller k. For the adder (*adr12*), and some arithmetic circuits (e.g., *alu4, alupla, cordia, tial*), ESOPs require fewer products than SOPs, in many cases. However, for some functions (e.g., *apex2*), ESOPs require more products than SOPs.

Table 5.2 shows the numbers of bits to represent function: $\mu(k) = p(k)(\frac{n}{k}2^k + m)$. For *alu4, apex2, intb, rdm16* and *tial*, $k = 2$ gives the smallest realizations; for *adr12, alupla, cordia* and *t481*, $k = 4$ gives the smallest realizations; and for *misex3*, $k = 5$ gives the smallest realization. The minimum values are highlighted by bold face letters.

Symmetric Functions

Table 5.3 shows the number of products to represent symmetric functions *SYM(n)* and *WGT(n)* [12]. Let $(x_1, x_2, \ldots, x_n)$ be the inputs and $n = 3m$. Then, $SYM(n) = 1$ iff $m \leq \sum_{i=1}^{n} x_i \leq 2m$. $WGT(n)$ is a binary representation of the number of 1's in the inputs. When $n = 2$, it corresponds to a half adder, while when $n = 3$, it corresponds to a full adder. When the number of variables of an original function is not a multiple of k, only one part has fewer than k binary variables. For example, to implement *sym12* by 32-valued logic, 12 variables are partitioned into $X_1 = (x_1, x_2, x_3, x_4, x_5)$, $X_2 = (x_6, x_7, x_8, x_9, x_{10})$, and $X_3 = (x_{11}, x_{12})$. Note that in the case of symmetric functions, finding the optimum grouping is much simpler than non-symmetric functions.

Table 5.4 shows that the number of bits to represent the symmetric functions. For the functions in this table, expressions with 16-valued variables are more efficient than ones

Table 5.1. Number of products to represent benchmark functions.

	In	Out	2-valued $k=1$		4-valued $k=2$		8-valued $k=3$		16-valued $k=4$		32-valued $k=5$	
	n	m	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP
adr12	24	13	24523	8191	145	79			68	43		
alu4	14	8	577	288	253	124	193	110	156	107	121	100
alupla	25	5	2144	1429	1008	806	625	614	429	372	480	397
apex2	39	3	39	60	14	27	11	13	14	27	14	24
cordia	23	2	914	776	67	104	28	17	15	10	11	7
intb	15	7	629	261	294	172	230	140	200	129	135	110
misex3	14	14	690	507	462	401	345	334	192	177	126	121
rdm16	16	16	404	176	281	140	213	121	140	93	105	72
t481	16	1	481	13	32	8	32	8	5	3	5	3
tial	14	6	575	428	230	172	230	152	195	137	158	118

Table 5.2. Number of bits to represent benchmark functions.

	In	Out	2-valued $k=1$		4-valued $k=2$		8-valued $k=3$		16-valued $k=4$		32-valued $k=5$	
	n	m	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP
adr12	24	13	1495903	499651	8845	4819			7412	**4687**		
alu4	14	8	20772	11196	9108	**4500**	8749	4987	9984	7296	11810	10150
alupla	25	5	117920	78595	55440	45375	44792	44003	45045	**39795**	77220	65505
apex2	39	3	3159	4860	**1134**	2187	1177	1391	2226	3816	3536	6062
cordia	23	2	43872	37248	3216	4992	1773	1077	1410	**940**	1641	1044
intb	15	7	23273	9731	10915	**6512**	10810	6580	13400	8643	13905	11536
misex3	14	14	28980	21294	19404	18102	17710	17145	13440	12670	13054	**12536**
rdm16	16	16	19392	8448	13488	**6720**	12379	7099	11200	7440	12432	8525
t481	16	1	15873	429	1056	264	1397	349	325	**195**	517	310
tial	14	6	19686	14552	9588	**5848**	10097	6587	12090	8556	15105	11281

with 4-valued variables. For *sym12* and *wgt12*, $k = 4$ gives the realizations with the smallest number of bits; for *sym15* and *wgt15*, $k = 5$ gives the smallest realizations; and for *sym18*, $k = 6$ gives the smallest realization. Please note that by Theorem 4.3 and Definition 4.1, we have the relation:

$$\mu(k) \leq (k+1)^{\frac{n}{k}}(\frac{n}{k})2^k.$$

Thus, for a large value of n, literals with large value of k tend to reduce the total number of bits.

5.2 Implementation on FPGAs

To assess the feasibility of the implementations on FPGAs, we designed two types of reconfigurable logic arrays.

Using CLBs Only

We implemented the array with 16-valued variables shown in Fig. 2.3 on Xilinx Spartan-3 XC3S4000 FPGA. LUTs of Xilinx FPGAs have two operation modes: the shift register (SRL16) mode, and the 4-LUT mode. In the shift register mode, reconfiguration of logic is done.

To implement a logic array for 32 inputs, 8 outputs, and 256 products, we had the following:

The number of LUTs: 5165.
The number of Slices: 4955.
Operating Frequency: 115 MHz.
Latency: 3 clocks.

In this case, one clock is used for the evaluation of the products, and two clocks are used for the OR gates. Note that 256-input OR gates were implemented by two-stage pipeline of 16-input OR gates.

For the product terms, $8 \times 256 = 2048$ LUTs are used. For the pipeline registers, $(256 + 16) \times 8 = 2176$ LUTs are used. For the OR gates, $(64 + 16 + 4 + 1) \times 8 = 680$ LUTs are used. Additional LUTs are used for writing the data into LUTs.

Using Both Embedded Memories and Logic Elements

We implemented the array with 16-valued variables shown in Fig. 2.5 on Altera Cyclone II EP2C35 FPGA. The design was done by Quartus II version 6.0.

In Fig. 2.4, the products are evaluated as follows: Initially, all the FFs are set to 1. After the first clock, the FFs represent $X_0^{S_0}$. After the second clock, the FFs represent $X_0^{S_0}X_1^{S_1}$. And, after 8 clocks, the FFs represent $X_0^{S_0}X_1^{S_1}\cdots X_7^{S_7}$. Additional 2 clocks are used for the pipeline registers in the 256-input OR gates. Since M4Ks are easily modified during operation, products are dynami-

Table 5.3. Number of products to represent symmetric functions.

	In	Out	2-valued $k=1$		4-valued $k=2$		8-valued $k=3$		16-valued $k=4$		32-valued $k=5$		64-valued $k=6$	
	n	m	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP
sym12	12	1	495	245	90	66	31	26	15	12	12	12	6	6
sym15	15	1	3003	1310	438	268	101	223	49	45	21	13	22	13
sym18	18	1	18673	7902	1598	919	346	263	172	139	75	74	29	35
wgt12	12	4	4095	422	425	146	135	54	51	28	41	26	17	14
wgt15	15	4	32767	2978	2510	886	530	223	224	75	77	36	69	33

Table 5.4. Number of bits to represent symmetric functions.

	In	Out	2-valued $k=1$		4-valued $k=2$		8-valued $k=3$		16-valued $k=4$		32-valued $k=5$		64-valued $k=6$	
	n	m	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP	SOP	ESOP
sym12	12	1	11880	5880	2160	1584	992	832	720	**576**	922	922	768	768
sym15	15	1	90090	39300	13140	8040	4040	8920	2940	2700	**2016**	2208	3520	2080
sym18	18	1	668304	284472	57528	33084	16608	12624	12384	10008	8640	8525	**5952**	6720
wgt12	12	4	98280	10128	10200	3504	4320	1728	2448	**1440**	3149	1997	2176	1792
wgt15	15	4	983010	89340	75300	26580	21200	8920	13440	4500	7392	**3840**	11040	5280

cally reconfigurable. However, logic elements (LEs) of Altera FPGAs do not have the shift register mode. Thus, we appended mechanism to reprogram the OR (EXOR) connection as shown in Fig. 5.1. For each input of the OR gate, a 2-input AND gate and a flip-flop (FF) are attached. When the value of the FF is 1, the corresponding input is selected. Otherwise, that input is ignored. To program the OR connections, the FFs are connected to form shift registers. The number of LEs to implement the programmable OR (EXOR) gates is about $\frac{4}{3}pm$, where p is the number of the products, and m is the number of the outputs. To implement a logic array for 32 inputs, 8 outputs, and 256 products, we had the following:

The number of LEs: 2663.
The number of M4Ks: 8 (32640 bits).
Operating Frequency: 235 MHz.
Latency: 10 clocks.

In this case, 8 clocks are used for the evaluation of the products, and two clocks are used for the OR gates. Again, 256-input OR gates were implemented by two-stage pipeline of 16-input OR gates. Note that the delay time of the circuit increases with the number of *parts* ($r = \frac{n}{k}$). Thus, to reduce the delay, k must be increased. However, the increase of k can increase the number of bits to represent the function.

Our experimental results show that the circuit using CLBs only is faster than ones using M4Ks and LEs. However, the method using M4Ks and LEs can implement more products than the method using CLBs only in comparable FPGA devices.

6. Conclusion and Comments

In this paper, we showed that an FPGA with k-input LUTs directly implements 2^k-valued expressions. We also presented two methods to implement reconfigurable logic arrays on an FPGA. Experimental results show that expressions with 2^k-valued variables ($k \geq 2$) require fewer products than corresponding expressions with 2-valued variables. For some functions, ESOPs require fewer products than SOPs, and vice versa. Since both expressions can be implemented in the same architecture, we can select the smaller ones. The number of bits to represent the expression with 2^k-valued variables takes its minimum when $k = 4$ for many functions. For some symmetric functions, $k = 5$ or $k = 6$ give the smallest realizations.

A reconfigurable logic array can be considered as a generalization of a content addressable memory (CAM) [2, 4, 6]. A function can be modified by only changing the contents of LUTs or BRAMs. The input/output pin assignment can be modified by only permutating the data for the columns or rows, in many cases. The reconfigurable logic array can implement different functions without changing the interconnections, so it can be reconfigured dynamically. Thus, it is suitable for pattern matching and networking.

This paper, we assume that all the LUTs have the same number of inputs. However, LUTs with different number of inputs can be used. The logic design method is similar to that of PLAs with k-bit input decoders [9, 12]. The PLAs with k-bit input decoders use 2^k literal lines for each group, while the reconfigurable logic array with 2^k-valued variables uses only k horizontal lines. Also, in the PLAs,

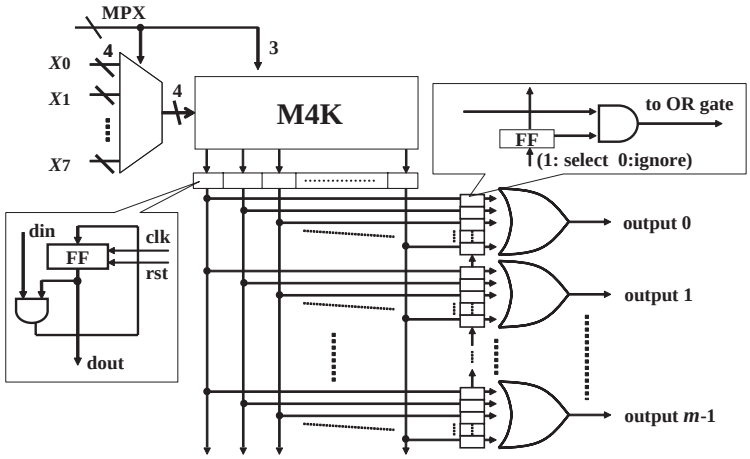

Figure 5.1. Implementation of programmable multi-output circuit.

each input decoder implements all the 2^k literals, while in the reconfigurable logic array, each LUT implements only one literal.

We are now improving methods to partition of the input variables [13], and outputs.

7. Acknowledgments

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, and the Grant of Knowledge Cluster Project of MEXT.

References

[1] Altera, http://www.altera.com/

[2] S. A. Guccione, D. Levi, and D. Downs, "A reconfigurable content addressable memory," In Jose Rolim et al. editors, *Parallel and Distributed Processing*, pp. 882-889, Springer-Verlag, Berlin, May 2000. *Proceedings of the 15th International Parallel and Distributed Processing Workshops*, IPDPS 2000. *Lecture Notes in Computer Science* 1800.

[3] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. & Develop.*, pp. 443-458, Sept. 1974.

[4] P. B. James-Roxby and D.J. Downs, "An efficient content-addressable memory implementation using dynamic routing," *FCCM'01 2001*, pp. 81-90, 2001.

[5] J. Kouloheris and A. El Gamal, "FPGA Performance vs. Cell Granularity," *Proc. 1991 CICC*, pp.6.2.1-6.2.4, May 1991.

[6] G. Nilsen, J. Torresen, and O. Sorasen, "A variable word-width content addressable memory for fast string matching," *Norchip Conference*, 2004.

[7] J. Rose, R.J.Francis, D. Lewis, and P. Chow, "Architecture of field programmable gate arrays: The effect of logic block functionality on area efficiency," *IEEE J. Solid State Circ. 25,5*, pp. 1217-1225, Oct. 1990.

[8] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization", *IEEE Trans. CAD*, Vol. 6(5), pp. 727-750, Sep. 1987.

[9] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. Comput.*, Vol. C-33, No. 10, pp. 879-894, Oct. 1984.

[10] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 5, pp. 621-632, May 1993.

[11] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[12] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[13] T. Sasao, "An application of 16-valued logic to design of reconfigurable logic arrays," *ISMVL-2007*, Oslo, Norway, May 13-16, 2007.

[14] N. Song and M. A. Perkowski, "Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions," *IEEE Trans.Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-15, No. 4, pp. 385-395, April 1996.

[15] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM'04), pp. 258-267, April 20-23, 2004.

[16] S. Yang, "Logic synthesis and optimization benchmark user guide, version 3.0," MCNC, Jan. 1991.

[17] Xilinx, http://www.xilinx.com/