Universal Logic Gate for FPGA Design

Chih-chang Lin, Malgorzata Marek-Sadowska and Duane Gatlin
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106

Abstract

In this paper the problem of selecting an appropriate programmable cell structure for FPGA architecture design is addressed. The cells studied here can be configured to the desired functionality by applying input permutation, negation, bridging or constant assignment, or output negation. A general methodology to determine logic description of such cells, which are capable of being configured to a given set of functions is described.

Experimental results suggest that the new cell behaves as well as the Actel 2 cell in terms of logic power but requires substantially less area and wiring overhead.

1 Introduction

We address the problem of selecting a logic block structure which will use as little silicon as possible and yet retain good mapping properties.

We propose an FPGA cell design approach based on the classification of Boolean functions [1]. We show that the FPGA architecture implementing our cell has excellent performance in terms of area and speed.

2 Universal Logic Gate

Universal logic gate $G(m,n)$ [1, 2, 3] is an m-input function, $G(y_1,\ldots,y_m)$, which can realize all the n-input functions, i.e. $f(x_1,\ldots,x_n)$, by assigning y_j to $1, 0, x_i$ or $\bar{x}_i$, or negating the output of G.

In this section, instead of finding the $G(m,n)$ which can realize all the n-input functions, we find an m-input function, $G(A)$, which can realize a given set of functions, A. The maximum number of inputs of functions in A is assumed be to n. The function $G(A)$ is called a Semi-ULG(A,m,n).

2.1 Terminology and definitions

Let f_i be a function of n variables $x_1,\ldots,x_n$.

Definition 1 (Input negation) *Two functions f_1, f_2 are said to be N_I equivalent iff there exists an assignment, $\dot{x_i} = x_i$ or $\bar{x}_i$, such that $f_1(\dot{x_1},\ldots,\dot{x_n}) = f_2(x_1,\ldots,x_n)$. $N_I(f)$ represents the set of all functions which are N_I equivalent to f.*

Definition 2 (Input permutation) *Two functions f_1, f_2 are said to be P equivalent iff there exists a permutation π, such that $f_1(x_{\pi(1)},\ldots,x_{\pi(n)}) = f_2(x_1,\ldots,x_n)$. $P(f)$ represents the set of all functions which are P equivalent to f.*

Definition 3 (Output negation) *Two functions f_1, f_2 are said to be N_O equivalent iff $f_1 = f_2$ or $\bar{f}_2$. $N_O(f)$ represents the set of all functions which are N_O equivalent to f.*

It can be shown that $N_I(), P()$ and $N_O()$ commute and are idempotent. The composition of three operators is defined as follows.

Definition 4 (NPN) *Two functions f_1, f_2 are said to be NPN equivalent iff there exist functions g_1, g_2 such that*

- *f_1 is N_I equivalent to g_1,*
- *g_1 is P equivalent to g_2, and*
- *g_2 is N_O equivalent to f_2.*

$NPN(f)$ represents the set of all functions which are NPN equivalent to f.

Now we consider the constant assignment (C) and bridging (B) operators. Let f be a function of n variables $(x_1,\ldots,x_n)$ and g be a function of $n+1$ variables $(x_1,\ldots,x_n,x_{n+1})$.

Definition 5 (Constant assignment) *A function g is said to be C_+ over f iff the cofactor of g with respect to x_{n+1}, $g_{x_{n+1}} = f$. A function g is said C_- over f iff $g_{\bar{x}_{n+1}} = f$. A function g is said C over f iff g is C_+ over or C_- over f. $C(f)$ represents the set of all functions which are C over f.*

Definition 6 (Bridging) *A function g is said to be B_+ over f iff there exists x_i, such that,*

$$g = x_{n+1}x_i f_{x_i} + \bar{x}_{n+1}\bar{x}_i f_{\bar{x}_i} + (x_{n+1} \oplus x_i) * [don't\ care].$$

A function g is said to be B_- over f iff there exists x_i, such that,

$$g = \bar{x}_{n+1}x_i f_{x_i} + x_{n+1}\bar{x}_i f_{\bar{x}_i} + (\bar{x}_{n+1} \oplus x_i) * [don't\ care].$$

A function g is said to be B over f iff g is B_+ over or B_- over f. $B(f)$ represents the set of all functions which are B over f.

We extend the definition of $N_I(), P(), N_O, C()$ and $B()$ to operate on a set of functions as follows.

Definition 7 *Let A represent a set of functions. $N_I(A) = \bigcup_{f\in A} N_I(f)$, $P(A) = \bigcup_{f\in A} P(f)$, $N_O(A) = \bigcup_{f\in A} N_O(f)$, $C(A) = \bigcup_{f\in A} C(f)$, $B(A) = \bigcup_{f\in A} B(f)$, and $(B + C)(A) = C(A) \bigcup B(A)$.*

For an n-input function f, let F_f be the set of all m-input functions which can realize f. If a function g can realize f, then all the functions which are NPN equivalent to g can also realize f. We have the following lemma.

Lemma 1 *$F_f = NPN(B + C)^{m-n}(f)$.*

2.2 ROBDD based implementation of the operators

Let C be a subset of B^p, where $B = \{0, 1\}$, C can be represented by its characteristic function as follows:

$$\chi_C(x) = \begin{cases} 1 & \text{if } x \in C \\ 0 & \text{otherwise.} \end{cases}$$

We will represent a set of functions using ROBDD [4]. Given a set F of n-input functions, its characteristic function is a boolean function with 2^n variables, since there are maximally 2^{2^n} n-input functions in F and 2^{2^n} different minterms in χ_F. In the following, we will use $\mathcal{F}$ to denote χ_F. Let a set of n-input functions F be represented by its characteristic function $\mathcal{F}(a_0, \ldots, a_{2^n-1})$. The following expression gives a formal condition for a membership in F:

$$f \in F \text{ iff } \mathcal{F}(a_0, \ldots, a_{2^n-1}) = 1,$$

$$\text{where } a_i = f(x_1, \ldots, x_n), \text{ and } i = (x_n \cdots x_1)_2.$$

2.2.1 $N_I(\mathcal{F})$ operator

When functions originally expressed in terms of some variable x_i are to be expressed in terms of $\bar{x}_i$, then the new characteristic function can be obtained from $\mathcal{F}$ by reordering of variable as follows:

$$N_I^1() : a_0, ..., a_{2^n-1} \Rightarrow a_1, a_0, a_3, a_2, ..., a_{2^n-1}, a_{2^n-2}$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$N_I^n() : a_0, ..., a_{2^n-1} \Rightarrow a_{2^{n-1}}, ..., a_{2^n-1}, a_0, ..., a_{2^{n-1}-1}.$$

To compute the characteristic function of a set of functions which are N_I equivalent to another set of n-input functions F represented by the characteristic function $\mathcal{F}$, n reorderings of the characteristic function are required.

2.2.2 $P(\mathcal{F})$ operator

Permutation of two input variables, x_i and x_j, on functions in F, results in reordering of the corresponding variables in $\mathcal{F}$. That is, if $p = (x_n, \cdots, x_i, \cdots, x_j, \cdots, x_1)_2$, then the input variable of $\mathcal{F}$, a_p will be re-labeled as a_q, where $q = (x_n, \cdots, x_j, \cdots, x_i, \cdots, x_1)_2$. Relabeling needs to be done only for $p \neq q$, and it can be formulated as swapping every pair of such variables a_p, a_q where $p = (x_n, \cdots, x_i = 1, \cdots, x_j = 0, \cdots, x_1)_2$ and $q = (x_n, \cdots, x_i = 0, \cdots, x_j = 1, \cdots, x_1)_2$. Let's denote the permutation of x_i and x_j as $P^{ij}(\mathcal{F})$. We need to perform the reordering $\frac{n(n-1)}{2}$ times, to generate all the $n!$ permutations, since the results are accumulated in ROBDD.

2.2.3 $N_O(\mathcal{F})$ operator

To negate outputs of the functions in F, the corresponding operation on $\mathcal{F}$ is an exchange of e_{high} and e_{low} edges for each node in the $\mathcal{F}$'s ROBDD representation.

2.2.4 $C(\mathcal{F})$ operator

The constant assignment operator on the functions in F results in the following re-labeling of the input variables in $\mathcal{F}$.

$$C_-() : a_0, \ldots, a_{2^n-1} \Rightarrow a_0, \ldots, a_{2^n-1}, \text{ and}$$
$$C_+() : a_0, \ldots, a_{2^n-1} \Rightarrow a_{2^n}, \ldots, a_{2^{n+1}-1}.$$

2.2.5 $B(\mathcal{F})$ operator

The positive bridging operation on variable x_i of functions in F, i.e. $x_{n+1} \leftarrow x_i$, results in relabeling input variable of $\mathcal{F}$ such that a_p maps to

$a_{p'}$, where $p = (x_n x_{n-1} \cdots x_i \cdots x_1)_2$ and $p' = (x_i x_n x_{n-1} \cdots x_i \cdots x_1)_2$. The positive bridging operation is denoted as $B^i_+(\mathcal{F})$. For $B^i_-(\mathcal{F})$, i.e. when $x_{n+1} \leftarrow \bar{x}_i$, we have $p' = (\bar{x}_i x_n x_{n-1} \cdots x_i \cdots x_1)_2$.

2.2.6 Semi-ULG

For each function f_i in A, we compute the characteristic function, $NPN(B + C)^{m-n}(f_i)$. From Lemma 1, we know that it contains all the m-input functions which can realize f_i. Then the intersection of all $NPN(B + C)^{m-n}(f_i)$s will contain all Semi-ULG(A, m, n)s which can realize all the functions in A.

3 Experiment with SIS LUT technology mapper

We have mapped the MCNC 91 benchmark circuits to 3 input LUT blocks. In our experiment, we used SIS standard script to obtain the mapping. The circuits were optimized by the SIS script, **source.rugged** first, and then the following script was used to map the circuits into 3-input functions:

```
xl_part_coll -m -g 2 -n 3
xl_coll_ck -n 3
xl_partition -m -n 3
simplify
xl_imp -n 3
xl_partition -t -n 3
xl_cover -e 30 -u 200 -n 3
xl_coll_ck -n 3.
```

Then, for each mapped function, we have determined the NPN equivalence class to which it belonged. The results are summarized in Figure 1. Note that the last 4 classes constitute only 3% of the mapped functions. Since those 4 equivalence classes are rarely used, we can exclude them from the set of functions A, for which Semi-ULG(A, m, n)s are determined. Clearly, we sacrifice some functional programmability but the size of the logic block and the number of inputs are reduced. The algorithm described in Subsection 2.2.6 was used to find all Semi-ULG$(A, 4, 3)$s. The number of inputs to Semi-ULG$(A, 4, 3)$ is 4, which is 1 less than the minimum required to realize all the 14 classes of 3-input functions. Among the 26 solutions, f_{11} is the best in terms of area and speed:

$$f_{11} = \bar{x}_0 \bar{x}_1 + x_1 x_2 x_3 + \bar{x}_2 \bar{x}_3.$$

Moreover, without using negative bridging, f_{11} still can realize the 10 of 14 classes. This is important because a negative bridging realization requires both phases of an input. From now on, we will refer to the f_{11} as the Semi-NPN-ULG(4,3).

4 Global phase assignment

It is possible that both phases of the same signal are required while only one phase is available. We call such a situation a polarity disagreement.

The circuit is assumed optimized and mapped by the SIS LUT FPGA technology mapper. Each node, n_i, of the circuit has at most three inputs. A function f_i is associated with each node n_i. f_i is a logic function of its direct inputs. For each f_i, there exist several ways to realize it by the Semi-NPN-ULG(4,3).

For a node, if its output polarity is different from the polarity required by its fanout node or when both positive and negative phases are required by the fanouts, we have to insert an extra Semi-NPN-ULG(4,3) to implement an inverter. Thus, the goal is to minimize the number of inserted inverters. We formulated it as $0-1$ integer programming problem.

We use the method of simulated annealing to solve the phase assignment problem, and the results for 22 MCNC 91 benchmark examples are shown in Figure 2. The fifth column, SIS, lists the count of the 3-input functions mapped by the SIS script mentioned in Section 3. For Semi-NPN-ULG(4,3), the percentage of disagreement drops from 38.5% to 30.1% when global phase optimization is applied, while for I-Semi-NPN-ULG(4,3) (with programmable output inverter), the percentage drops from 11.2% to 6.7%. The column labeled K counts the number of nodes which belong to the excluded 4 classes. Their total percentage is less than 3%. For each such a function which belongs to an excluded class, we need to use 2 Semi-NPN-ULG(4,3) blocks to realize it.

The columns, Amap XAmap and Xmap, are from [5, 6]. Amap and Xmap are specialized technology mappers for Actel 2 and LUT, respectively. The first column, NEW SIS, is from [7] which is an improved version of Amap. On the average, the number of blocks used by I-Semi-NPN-ULG(4,3) is about 9% more than that of LUT(3). Note, that the number of blocks required by I-Semi-NPN-ULG(4,3) is determined by adding the entries labeled SIS and entries labeled G_2 and K. Comparing to Actel 2, the number of blocks[1] required by the dual-output Semi-NPN-

[1] For a fair comparison, we count the number of blocks for each circuit as the smaller one from SIS or XMAP LUT technology mapper.

ULG(4,3) is about 5% more. Note that, when calculating the number of blocks required by dual-output Semi-NPN-ULG(4,3), the value of G is zero.

5 CMOS realization of the Semi-NPN-ULG(4,3)

In the CMOS complex gate design, it is desirable to reduce the number of inversions in the formula. Thus, we re-work the formula which describes the Semi-NPN-ULG(4,3) from $\bar{x}_0\bar{x}_1 + x_1x_2x_3 + \bar{x}_2\bar{x}_3$ to

$$\overline{(x_0 + x_1)\ \overline{x_0x_1x_2}\ (x_2 + x_3)}.$$

It takes 6 transistors to implement $\overline{x_0x_1x_2}$ and 10 for the remaining circuit. Totally, the Semi-NPN-ULG(4,3) can be implemented with 16 transistors. For the dual-output Semi-NPN-ULG(4,3) cell, 2 extra transistors are needed for the inversion, while for the I-Semi-NPN-ULG(4,3) cell, the exor gate requires 12 transistors, and the one-bit memory cell requires additional 6 transistors.

A multiplexor function $\bar{s}d_0 + sd_1$, requires 10 transistors, where s is the selection bit, and d_0 and d_1 are the data bits. The total number of transistors for the Actel 2 cell is about 38. Comparing the dual-output Semi-NPN-ULG(4,3) to the Actel 2 cell, besides the fewer number of transistors used, it also reduces by 33% the number of programmable switches[2] (anti-fuse) in the routing channels. This is because there are 4 inputs and 2 outputs in the dual-output Semi-NPN-ULG(4,3) cell, while there are 8 inputs and one output in Actel 2 cell. The speed analysis by SPICE indicates that the dual-output Semi-NPN-ULG(4,3) is 30% faster than the Actel 2 cell.

For the LUT(3), we assume that each memory bit costs 6 transistors. Added up with the decoder circuitry, the total number of transistors for each LUT(3) cell is about 102. Compared to LUT(3), although the I-Semi-NPN-ULG(4,3) cell requires less transistors (34), it suffers from having one more input.

6 Conclusions

The theory of NPN equivalence has been described and applied to characterize the three input functions produced by SIS LUT technology mapper for the MCNC 91 benchmarks. The results of this characterization revealed that ten of the fourteen NPN equivalent classes were used in 97% of the mapped functions. A method was then proposed to construct logic modules that could cover the most frequently used NPN equivalent classes. The logic module selected by us is the Semi-NPN-ULG(4,3). We have compared the area and speed characteristics of the dual-output Semi-NPN-ULG(4,3) logic module with the Actel 2 logic module, and I-Semi-NPN-ULG(4,3) logic module with three inputs look up table realization.

[2]Each input or output of the logic cell needs to be be connected to the routing channel by a programmable anti-fuse switch.

Acknowledgements

This work was supported in part by the National Science Foundation under Grant MIP 9117328 and in part by AT&T Bell Laboratories and Digital Equipment Corporation through the California MICRO Program.

References

[1] C. R. Edwards, "A special class of universal logic gates (ULG) and their evaluation under the Walsh transform," *Int. J. Electronics*, vol. 44, pp. 49–59, 1978.

[2] X. Chen and S. L. Hurst, "The derivation of universal logic modules for n >= 3 by algebraic means," *Proc. IEE, part E, Vol. 128, No. 5*, pp. 205–211, 1981.

[3] H. S. Stone, "Universal logic modules," in *Recent Developments in Switching Theory* (A. Mukhopadhyay, ed.), ch. VI, pp. 229–254, 1971.

[4] O. Coudert, C. Berthet and J. C. Madre, "Verification of sequential machines based on symbolic execution," *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, 1989.

[5] K. Karplus, "Using if-then-else DAGs to do technology mapping for field-programmable gate arrays," Report UCSC-CRL-90-43, University of California, Santa Cruz, 1990.

[6] K. Karplus, "Amap: a technology mapper for selector-based field-programmable gate arrays," *ACM/IEEE Design Automation Conference*, pp. 244–247, 1991.

[7] R. Murgai et al., "An improved synthesis algorithm for multiplexor-based PGAs," *ACM/IEEE Design Automation Conference*, pp. 380–386, 1992.

ID	inputs	NPN class	%	Realizable
1	0	0	0	*yes*
2	1	$\bar{a_0}$	0	*yes*
3	2	$a_0 a_1$	19	*yes*
4	2	$a_0 \oplus a_1$	2	*yes*
5	3	$a_0(a_1 + a_2)$	42	*yes*
6	3	$a_0 a_1 a_2$	23	*yes*
7	3	$a_0 a_1 + \bar{a_0} a_2$	9	*yes*
8	3	$a_0(a_1 \oplus a_2)$	2	*yes*
9	3	$a_0 \bar{a_1} + a_0 \bar{a_2} + \bar{a_0} a_1 a_2$	2	*yes*
10	3	$a_0 a_1 + \bar{a_0} \bar{a_1} a_2$	0	*yes*
11	3	$a_0 \oplus a_1 \oplus a_2$	2	*no*
12	3	$a_0 a_1 + a_1 a_2 + a_0 a_2$	1	*no*
13	3	$\bar{a_0} \bar{a_1} \bar{a_2} + a_0 a_1 a_2$	0	*no*
14	3	$\bar{a_0} a_1 a_2 + a_0 \bar{a_1} a_2 +$ $a_0 a_1 \bar{a_2}$	0	*no*

Figure 1: The percentages of 3-input NPN classes used by the SIS LUT technology mapper on the MCNC 91 benchmark circuits.

MCNC example	(Actel) NEW SIS	(Actel) Amap	(Actel) XAmap	(LUT) Xmap	(LUT) SIS	I-Semi-ULG(4,3)			Semi-ULG(4,3)		
						G_1	G_2	K	G_1	G_2	K
5xp1	39	53	58	49	48	6	2	0	24	13	0
9symml		102	110	93	25	0	0	6	8	6	6
C1908	159	204	200	167	142	26	9	19	50	28	19
C499	166	141	139	105	100	0	0	32	8	1	32
C5315	560	604	685	577	555	53	25	42	205	122	42
alu2	175	196	210	177	165	28	17	0	86	69	0
alu4		336	365	311	300	59	39	5	156	123	5
apex6	275	366	370	320	327	21	13	1	103	89	1
apex7	92	112	109	96	95	11	7	0	17	14	0
bw	54	94	102	77	71	13	10	1	35	30	1
clip	43	53	57	46	57	8	6	2	27	21	2
count	39	62	55	47	55	1	1	0	8	8	0
des	1318	1605	1652	1418	1537	136	101	2	679	616	2
duke2	158	169	188	156	207	37	23	0	100	81	0
f51m	39	58	64	54	40	8	6	1	17	13	1
frg1		61	61	58	62	2	1	0	27	20	0
frg2		406	412	350	375	20	9	2	49	26	2
k2		482	473	418	467	103	29	0	221	124	0
pair		684	729	614	625	65	46	8	237	204	8
rd84	36	93	101	85	33	0	0	6	6	4	6
rot	259	309	309	272	277	32	23	0	82	58	0
vg2	31	37	39	37	39	1	0	0	11	11	0
Total		6227	6488	5527	5602	630	367	127	2156	1681	127
						11.2%	6.6%	2.3%	38.5%	30.0%	2.3%
Sub total	3443	4156	4338	3683	3748	381	243	106	1458	1178	106
						10.2%	6.5%	2.8%	38.9%	31.4%	2.8%

Figure 2: The results of global phase assignment. G_i counts the number of polarity disagreement. G_1 is the number of the initial polarity disagreements, while G_2 is the number after performing minimization by simulated annealing. The value K counts the number of functions which belong to the 4 un-realizable classes.