SET OF DIGITAL CELLS ACCORDING TO LOGIC EQUIVALENCES

[1]Marcos F.L. Ledur, [1]Felipe Marranghello, [1]Leomar Rosa Jr.,
[2]André Reis, [1]Renato Ribas

[1]Instituto de Informática – UFRGS, Porto Alegre, Brazil
[2]Nangate Inc., Menlo Park, CA, Herlev, Denmark
{mflledur, fsmarranghello, leomarjr, rpribas@inf.ufrgs.br}, are@nangate.com

ABSTRACT

This paper discusses two linked subjects. The first one is the concept of equivalence classes of logic functions. An algorithm that lists all possible classes for logic functions with a certain number of inputs is conceived. Secondly, a brief analysis of digital circuits mapped by library-free and library-based technology mapping tools is presented, where the main goal is to identify equivalence classes and occurrence of cells in the circuits generated by them. To do that, a tool is also proposed. This tool may reduce the set of cells, grouping functionally equivalent cells and choosing one to represent the group. The preliminary results show a strong equivalence presence in library-free mapping processes.

1. INTRODUCTION

Boolean functions are the pedestal for the design of modern digital circuits. Basically, any logic function may be translated in a transistor network that represents it in an electrical arrangement. These transistor networks compose the logic gates to form the digital circuit. For a given number of input variables there is a well-defined number of functions. This number is given by 2^(2^*n*), where *n* is the number of input variables [1]. According to this statement, the number of 2-input functions is 16, 3-input functions is 256, 4-input functions is 65,536, 5-input functions is 4,294,967,296, and so on. This exponential relationship turns the search space almost intractable if many operations need to be repeated in a set of functions with more than 4-inputs. In order to reduce the search space, the n-input functions can be classified into different classes (set of functions). These sets are known as equivalence classes, and they may be obtained through input permutation/negation as well as output negation. *P-class, N-class, NP-class, PN-class*, and *NPN-class* are the possible reduced sets. These sets are composed by equivalent functions that represent different logic functions, but that have the same internal transistor networks arrangements (they are topologically identical). This approach can be used to implement and to evaluate logic cells and transistor networks without loss of generality.

Another important point is related to the technology mapping strategy. Essentially, technology mapping tools can be classified in two groups: library-based and library-free. Library-based tools are the ones that use a predefined and reduced set of cells to build the digital circuit. In other words, the circuit will be composed by logic cells that belong to a fixed library container. Alternatively, library-free tools are the ones that need an automatic cell generator to feed the technology mapping process. In these tools the set of cells that can be used are much more extensive.

This work proposes two main discussions regarding classes of functions and technology mapping. First, it presents an algorithm to identify functions that belong to the same classes. Second, it presents a mechanism that is able to identify equivalence and occurrence of logic functions inside circuits mapped by technology mapping tools.

The main contribution of this paper is to provide for the designer a fast and powerful way to analyze the characteristics and the quality of used cells in the technology mapping process.

The remaining of this paper is organized as follow. Section 2 discusses the identification of logic function classes. Section 3 presents the proposed mechanism to evaluate the set of cells used to compose the mapped circuits. Finally, Section 4 presents the conclusions.

2. LOGIC FUNCTION CLASSES

2.1 Logic function

In a truth table, for a certain number of inputs (*n*), there are *2^n* lines (minterms). Each output consists of choosing a bit for each minterm. Therefore, there are *2^(2^n)* possibilities of outputs. Each one is a boolean function, a binary number with *2^n* bits. For $n = 2$, e.g. there are 4 minterms and 16 logic functions, as shown in Table 1.

Table 1: Some functions for $n = 2$ inputs.

AB	f_0	f_1	f_2	f_3	f_4	...	f_{12}	f_{13}	f_{14}	f_{15}
00	0	1	0	1	0	...	0	1	0	1
01	0	0	1	1	0	...	0	0	1	1
10	0	0	0	0	1	...	1	1	1	1
11	0	0	0	0	0	...	1	1	1	1

2.2 Equivalence classes

If n increases, the total space of functions will enlarge considerably. As mentioned before, in a cell with 3 inputs, the number of possible functions equals to 256, and for 4 inputs, 65,536. To reduce these possibilities, the concept of classes of equivalent functions is introduced [3]. A class is a subset of logically equivalent functions as a result of a specific operation, or their combination.

A possible operation to obtain equivalent functions is the permutation of inputs. Table 2 presents an example of that operation. It is noticed that the minterms 01 and 10 changed the position in both tables. Two functions are equivalent if for each minterm the output is the same. Thus, f_2 and f_4 are equivalent by permutation, and can be gathered in a *P-class*.

Another operation to build equivalent functions is the negation of inputs. In a similar way, Table 3 shows an example of obtaining an *N-class* of 4 equivalent functions (f_1, f_2, f_4 and f_8) from this operation.

The last operation here used is the negation of the output. Table 4 displays and example of this operation.

The three operations can be combined. For instance, *NP-classes* are obtained after combining permutation and negation of inputs. An *NP-class* may have more functions than several *P-* or *N-classes*. Therefore, the number of *NP-classes* is smaller.

There are still *PN-classes*, where permutation of inputs and negation of outputs are executed. Finally, the most reduced group of classes studied is the set of *NPN-classes*, where all operations are performed.

Table 2: Two *P-equivalent* functions.

AB	f_2	**BA**	f_4
00	0	**00**	0
01	1	**10**	0
10	0	**01**	1
11	0	**11**	0

Table 3: Four *N-equivalent* functions.

AB	f_1	$\overline{\mathbf{AB}}$	f_2	$\overline{\mathbf{AB}}$	f_4	$\overline{\mathbf{AB}}$	f_8
00	1	**01**	0	**10**	0	**11**	0
01	0	**00**	1	**11**	0	**10**	0
10	0	**11**	0	**00**	1	**01**	0
11	0	**10**	0	**01**	0	**00**	1

Table 4: Two equivalent functions after output negation.

AB	f_9	f_6
00	1	0
01	0	1
10	0	1
11	1	0

2.2 Finding equivalence classes

The interest of the first part of this work was to list all classes (*P, N, NP* or *NPN*) for a known number of inputs. To do that, a function of each class was chosen to represent it. The chosen function was, for convenience, the lowest function of each class; 'lowest function', here, means the function f_k, where k is the smallest value observed in the class.

Fig. 1 shows a general procedure, in a C-like code, of the idea described above. Going through all *2^2^n* functions using the variable `f`, in case the lowest function of a class is found, it is printed.

The procedure `isLowestFun`, illustrated in Fig. 1, should be in agreement with the type of desired operation. In case it is wanted to know if `f` is the lowest function after all possible input permutations, this procedure resembles that in Fig. 2.

Basically, the lowest function is searched in the main loop. There are *n!* different combinations for n inputs. For each one, the procedure `kth_permutation` returns a logic function always obtained from a different permutation. In each cycle, if the found function is the lowest at the moment, it is stored in `lower`. At the end, if `lower` is not lower than `f`, it means that `f` is the lowest function of the class.

An example of this reasoning is that of Table 2. There are only two possible permutations for $n = 2$. If `f` = f_2, the result of `kth_permutation` is f_4, and f_2 will be printed. Later, when `f` equals to f_4, a lower function will be found, obviously f_2, and f_4 will not be printed. Hence, only one function per class is printed, which was the main goal.

If the goal is to obtain the lowest function of an *N-class*, then `isLowestFun` will be built in a similar way, but the number of search cycles will be *2^n*, and the search procedure, `kth_negation`. These changes are shown in Fig. 3. For $n = 2$, there are *2^2* = 4 possible negations of the inputs. According to Table 3 and the algorithm of Fig. 3, if `f` equals to f_1, f_2, f_4 or f_8, only when `f` = f_1 the procedure `print` will be called.

```
void list(int n) {
  for (int f = 0 ; f < 2^(2^(n)) ; f++)
    if (isLowestFun(f))
      print(f);
}
```

Figure 1: Listing one function per class.

```
int isLowestFun(long f) {
  long f' , lower;
  lower = f;
  for (int i = 0 ; i < n! ; i++) {
    f' = kth_permutation(f , i);
    if (f' < lower)
      lower = f';
  }
  if (lower < f)
    return 0;
  else
    return 1;
}
```

Figure 2: Obtaining *P-classes*.

```
int isLowestFun(long f) {
  ...
  for (int i = 0 ; i < 2^n ; i++) {
    f' = kth_negation(f , i);
    if (f' < lower)
      lower = f';
    } ...
}
```

Figure 3: Changes in isLowestFun procedure for obtaining *N-classes*.

Fig. 4 shows the necessary changes in `isLowestFun` for the accomplishment of permutation and negation operations simultaneously. Now the number of cycles is *n! × 2^n*.

At last, when the purpose is to find *NPN-classes*, each obtained function has to be negated and, in a similar way, verified if the new function is the lowest one of the whole class. To do that, some code is added to the procedure of Fig. 4, resulting in the procedure of Fig. 5. Procedure `neg()` returns a logic function replacing 0's by 1's, like it was done in Table 4.

```
int isLowestFun(long f) {
  ...
  for (int i = 0 ; i < n! ; i++)
    for (int j = 0 ; j < 2^n ; j++) {
      f' = kth_permutation(f  , i);
      f' =    kth_negation(f' , j);
      if (f' < lower)
        lower = f';
    }  ...
}
```

Figure 4: Obtaining *NP-classes*.

```
int isLowestFun(long f) {
  ...
  for (int i = 0 ; i < n! ; i++)
    for (int j = 0 ; j < 2^n ; j++) {
      f' = kth_permutation(f  , i);
      f' =    kth_negation(f' , j);
      if (f' < lower)
        lower = f';
      if (neg(f') < lower)
        lower = neg(f');
    }  ...
}
```

Figure 5: Changes in isLowestFun procedure for obtaining *NPN-classes*.

Some remarks should be made. For simplicity, the algorithms are more illustrative and less realistic than the used ones. Improvements are visible and necessary. Besides, the possible number of logic functions and results turn then unfeasible for high values of *n*. The results were obtained for an *n* up to 5 inputs, according to Table 5. They are in agreement with [2].

With the developed algorithms, it is possible to compare two functions and to verify if they are logically equivalent, as explained in the next section.

Table 5: Number of inputs vs. number of classes.

n	# of functions	# of classes			
		P	N	NP	NPN
2	16	12	7	6	4
3	256	80	46	22	14
4	65.536	3.984	4.336	402	222
5	4.294.967.296	37.333.248	134.281.216	1.228.158	616.126

3. EVALUATION OF SET OF CELLS

The second part of this work was implemented in two stages. In the first one, a list of cells obtained from a library-free mapping tool, and defined by their logical equations, is taken into account. Firstly, these equations are converted to the format of logic functions (in hexadecimal). After that, these functions are used as inputs to a developed tool, called LEVE (Logical Equivalence VErifier), written in C, which analyzes the functions, counts the different types of cells and the occurrence of each one of them, generating a report. In the second moment, LEVE tool uses the concepts of the developed algorithms to verify if two or more functions belong to the same equivalency class. The flow is presented in Fig. 6.

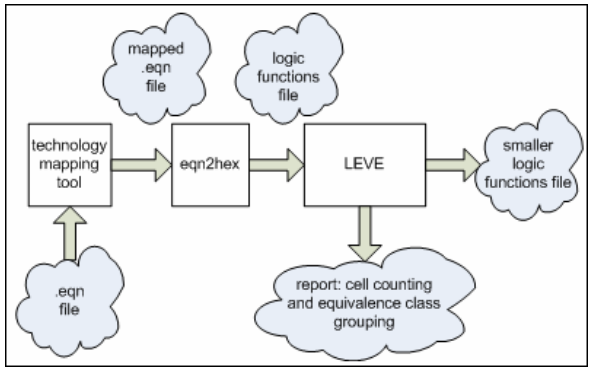

Figure 6: LEVE tool flow, inputs and outputs.

3.1 Library container evaluation

Technology mapping is a process by which a set of optimized logic equations is turned into a design implementation in terms of a cell netlist. A standard cell technology uses a pre-characterized library, while a library-free technology uses a virtual library based on on-the-fly cell generation. SIS [4] is an example of a library-based tool. On the other hand, ELIS [5] and VIRMA [6] tools are examples of library-free tools.

A standard cell library is typically restricted to a few cells. However, the flexibility of the library-free cell generation may produce several distinct cells. Besides, a high occurrence of cells can implicate a worse performance in area and in delay.

3.2 Equivalence in a set of cells

Library-free tools do not explore the fact that several cell types can own some kind of logic equivalence studied in the previous section. That approach is useful to reduce the size of the list of generated cells, choosing one among the functionally equivalent cells to belong to the final circuit.

The algorithm for obtaining the class of a function is shown in Fig. 7. It looks like the one presented in Fig. 2, with the difference that `theLowestFun` returns the lowest function of a class. Here only the permutation operation is presented, but the other algorithms follow strictly the same way.

To verify if two or more functions are equivalent the procedure is the following: given two functions f_a and f_b, if `theLowestFun` returns f_k, the three belong to the same class. The set $A_k = \{f_a , f_b\}$ contains, thus, functionally equivalent cells. This set is increased as new functions equivalent to f_k are found. A specific cell of this set can be chosen to represent it. Therefore, the number of different cells used in an implementation can be smaller, the greater are the sets. For the logical/physical obtainment of the not chosen cells, it is enough to perform the necessary operations on the chosen ones.

The limitation of the algorithm is the number of inputs (n). The greatest difficulty found was the time expensed in the algorithm execution. For cells of 9 inputs, the average time to find its *NPN-class* was of the order of 40 seconds, and for 10 inputs, around 32 minutes. This happens mainly because the search space ($n! \times 2^n$) is increased in $2(n+1)$ when n is incremented. Therefore, the algorithm was limited for up to 10 inputs.

```
long theLowestFun(long f) {
  long f' , lowest;
  lowest = f;
  for (int i = 0 ; i < n! ; i++) {
    f' = kth_permutation(f , i);
    if (f' < lowest)
      lowest = f';
  }
  return lowest;
}
```

Figure 7: Obtaining the lowest function of a class.

3.3 Some results

Table 6 presents the results for a group of well-known benchmarks. Three circuits were mapped, each one, in three different tools. The third column shows the total counting of cells after the mapping. Next, the number of distinct cells found is presented. The following columns show the decrease of the on-the-fly generated library after verification of equivalence classes *P*, *NP* and *NPN*.

Some conclusions can be given by looking the Table 6. First of all, SIS-mapped circuits did not present permutation equivalences, and in the other tools they occur often. It happens because library-free processes may not verify permutations while they analyze the circuit. Finally, as it was seen to come, NPN verification returned the most reduced group of cells.

Table 6: LEVE tool results for ISCAS benchmarks.

ISCAS Circuit	Tech mapping tool	# of cells	# of distinct cells	# of cells after equivalence		
				P	NP	NPN
C499	SIS	174	8	8	8	6
	ELIS	162	6	5	5	4
	VIRMA	212	23	15	10	9
C1908	SIS	346	11	11	11	8
	ELIS	244	16	15	13	10
	VIRMA	187	81	57	28	28
C3540	SIS	500	34	34	34	24
	ELIS	473	46	38	38	23
	VIRMA	545	164	131	69	65

Some aspects should be considered for the observation of the results. On one hand, permutation is an operation that does not modify the internal structure of the circuit, but the decrease in the number of cells is not so evident. On the other hand, reduction using negations in inputs and/or in output would add inverters to the circuit, in spite of a good reduction of cells. It is necessary to affirm that the eventual inverters were not considered in the total number of cells.

4. CONCLUSIONS

This paper discussed the equivalence of classes of logic functions and its use in the context of technology mapping. Some choices of equivalence were presented, and choosing one of them depends on what is desired in the final circuit. Future work will include a better report generation and an investigation of the best cells to be chosen to compose the final circuit.

5. REFERENCES

[1] Sasao, Tsutomu. *Switching Theory for Logic Synthesis*. Springer, 1st edition. 1999.

[2] M. A. Harrison, "The number of equivalence classes of Boolean functions under groups containing negation", *IEEE Trans. Electron. Comput. 12*, pp. 559-561, 1963.

[3] V.P. Correia, and A.I. Reis. "Classifying n-Input Boolean Functions", *VII Workshop IBERCHIP 2001,* IWS 2001, Montevideo, pp. 58, 2001.

[4] E.M. Sentovich, *et al*., "SIS: A system for sequential circuit synthesis", *Technical Report No. UCB/ERL M92/41*,

[5] ELIS Home Page http://www.inf.ufrgs.br/lagarto

[6] F.S. Marques *et al*., "DAG based library-free technology mapping", *Proceedings of the 17th great lakes symposium on Great lakes symposium on VLSI*, ACM Press, New York, pp. 293-298, 2007.