내부 Don't care를 이용한 이차원 셀 배열의 새로운 합성 방법

論 文
49D-2-5

A New Approach to the Synthesis of Two-Dimensional Cellular Arrays Using Internal Don't Cares

李 東 建* · 鄭 美 京** · 李 貴 相***
(Dong-Keon Lee · Mi-Gyung Jung · Guee-sang Lee)

Abstract - This paper presents a new approach to the synthesis of two-dimensional arrays such as Atmel 6000 series FPGAs using internal don't cares. Basically complex terms which fits to the linear array of cells without further routing wires are generated and they are collected by OR/XOR operations. In previous methods, complex terms are collected only by XOR operations, which may not be effective for nearly unate functions. In this paper, we allow complex terms to be collected by OR operations in addition to XOR operations. First, complex terms that lies in the ON-set of the function are generated and collected by OR operations. The sub-function realized by the first stage becomes an internal don't cares and they are exploited in the second stage which generates complex terms collectable by XOR operation. Experimental results shows the efficacy of the proposed method compared to the previous methods.

Key Words : Logic synthesis, FPGA, Two-Dimensional Cellular Array, Complex Term, Don't Care, OR, XOR

1. Introduction

CA(Cellular Architecture)-type devices are characterized by relatively small logic blocks and local connectivity between them. A generic model of CA-type FPGA consists of the array of cells which are connected to their neighbours and to local buses; vertical and horizontal. Consider an example in Figure 1 which shows a multi-output function implemented in a two-dimensional cell array as follows:

$$\begin{aligned} C_0 &= (a+b)\cdot c \\ C_1 &= a\cdot b+c \\ C_2 &= (a\oplus\overline{b})+c \\ f_0 &= C_0\oplus C_1 \\ f_1 &= C_0\oplus C_2. \end{aligned}$$

For simplicity, we assume that the number of inputs to a cell is limited to two, and the number of outputs of a cell is limited to one. Furthermore, only one input is taken from the local bus and the logic blocks can realize an some of their combinations. In Atmel 6000 series inverter, an AND, OR, EXOR, NAND gate, a wire and FPGAs[1], the architecture limitation is the same as the above except that the number of inputs to a logic block is limited to three and outputs to two.

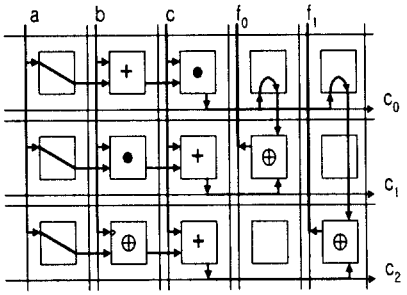

Fig. 1 Example of cell array implementing a multi- output function

For the synthesis of CA-type FPGAs, several approaches have been presented. The approaches from the first group utilize various decision diagrams including FDDs(Function Decision Diagrams)[2], and KFDDs(Kronecker Functional Decision Diagrams)[3] and tree structures[4]. However, these approaches may suffer from the drawback that when the diagrams are directly mapped to a rectangular area, they may lead to a significant waste of cells due to the routing difficulty and sometimes due to its tree like structure. Secondly, an algebraic approach presented by Sarabi et al. [5] provides a well-defined theoretical background for the manipulation of Boolean functions applicable to CA two dimensional arrays.

* 準 會 員 : 全南大 電算學科 博士課程
** 準 會 員 : 全南大 電算學科 理學博士
*** 終身會員 : 全南大 電算學科 副教授 · 理博
接受日字 : 1999年 11月 6日
最終完了 : 2000年 1月 15日

Later Lee[6] improved this result by using ETDDs(EXOR Ternary Decision Diagrams). Our synthesis model is composed of two planes: the complex(input) and collecting(output) plane as in [5], but the collecting plane can use OR operatoins as well as XOR operations as shown in Figure 2. Note that in [5] or [6], it is required

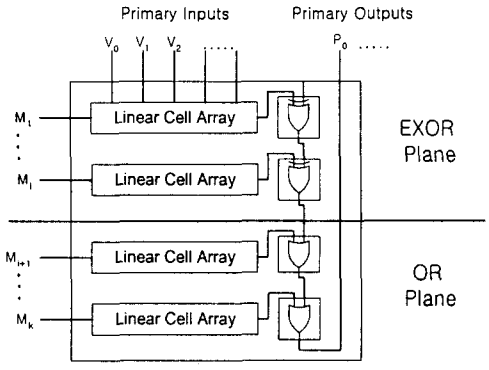

Fig. 2. OR-plane terms are considered as don't cares to EXOR-plane terms

that only XOR operations are used in collecting plane. However, this restriction becomes an obstacle to the efficient generation of complex terms resulting in unoptimized cell arrays. In this paper, we propose a new structure where both OR operations and XOR operations are used to collect complex terms in the synthesis procedure. First, complex terms that lie on the ON-Set of the function are searched and collected with OR-operation if they reduce the cost function. After these ON-Set complex terms are generated, they can be used as internal don't cares to the generation of XOR collectable complex terms. The proposed cell structure is similar to the conventional PLA architecture, but a linear array of cells can implement a broader class of Boolean functions than a simple *product term* in PLA. Since each cell can realize an AND, OR, EXOR or some of their combinations, the outputs of the cell array constitute a special class of Boolean functions called *Maitra terms*(or *complex terms*) which are named from *Maitra Cascade*[7].

ETDDs are similar to BDDs (Binary Decision Diagrams), but each node has one more branch which is an exclusive-OR of the other two branch functions.(Functions with three outgoing edges per node are called TDDs(Ternary Decision Diagrams). For an overview see [8]) Our approach uses ETDD instead of cubes as in [5] and exploits the decompositions of KFDD which are inherent in ETDD to generate complex terms instead of mapping the diagram directly to cell arrays.(A KFDD of a function can be constructed by selecting two of the three branches in the ETDD of the given function.) Our approach can make use of don't care values during the minimization process. Experimental results are described for completely and incompletely specified benchmark circuits. These experiments underline the quality of the approach.

The paper is structured as follows: In Section 2. notations and definitions are given that are of important for the understanding of the article. The minimization method based on ETDDs is described in Section 3 where don't care assignment is studied with its application to the generation of OR/XOR collectable complex terms in Section 4. and our experimental results are described in Section 5. Finally, the results are summarized.

2. Preliminaries

Boolean expressions which can be directly mapped to linear cell arrays are defined and are called complex terms. The following definitions are taken from [5]:

Definition 1. A *complex term* is defined recursively as follows.

1. *a literal is a complex term.*
2. *if M is a complex term and a is a literal then $a \cdot M$, $a \oplus M$, $a + M$ are complex terms, where a variable never appears more than once in the string.*

Note that a specific ordering is imposed on input variables to form complex terms.

It is not difficult to see that a complex term can be implemented in a cell array without extra routing wire.

Example 1.

1. A logic expression (a+b)c + d can be implemented in cell arrays as shown in Figure 3. However, (a+b)(c+d) is not a complex term and it needs another routing wire to be realized in the CA[5]. Note that the input ordering of a, b, c, d is imposed in this example.
2. Three complex terms are implemented in two-dimensional cell arrays of figure 1 to form a multi-output function.

Given a Boolean function $f(V)$ and a variable $v \in V$, $f(V)$ can be expressed as:

$$f(V) = v \cdot f_v \oplus \bar{v} \cdot f_{\bar{v}} \quad (1)$$
$$= v \cdot g \oplus f_{\bar{v}} \quad (2)$$
$$= f_v \oplus \bar{v} \cdot g \quad (3)$$

where $g = f_v \oplus f_{\bar{v}} = f(v=1) = f(v=0)$. We call these

equations as *Davio expansions* and particularly equation 2 and equation 3 are called as *positive Davio decomposition* and *negative Davio decomposition*, respectively. Note that equation 1 is called as *Shannon decomposition* which is used in BDD representations.

3. Generation of complex terms using don't cares

Considering a KFDD which uses Davio expansions without considering don't cares, a path from the root to a leaf constitutes a complex term. Therefore we try to select decompositions which results in the smallest number of different paths in the KFDD of the given function. Note that the KFDD is implicitly contained in the corresponding ETDD which is again easily constructed from the BDD.

In the following we give an example to demonstrate the different representation size resulting from choice of the decomposition formula:

Example 2 *In figure 4, when Shannon decomposition is used it results in 3 complex terms as follows.*

$$\begin{aligned} f &= \bar{a} \cdot f_{\bar{a}} \oplus a \cdot f_a \\ &= \bar{a}\bar{b}(\bar{c} \oplus d) \oplus abcd \end{aligned}$$

However, if negative Davio decomposition is used as in figure5 only two complex terms are needed as follows.

$$\begin{aligned} f &= a \cdot (f_{\bar{a}} \oplus f_a) \oplus f_a \\ &= \bar{a}(\bar{b} \oplus c \oplus d) \oplus bcd. \end{aligned}$$

The first attempt to manipulate don't care terms in Davio expansions appears in [9]. where the basic rules for calculating constrained don't cares in Davio expansions are given. The order of minimizing $f_{\bar{x}_i}$, f_{x_i} and $f_{\bar{x}_i} \oplus f_{x_i}$ affects the final result because the don't cares amongst them are mutually constrained. Consider the example of figure 6 which is given in [9].

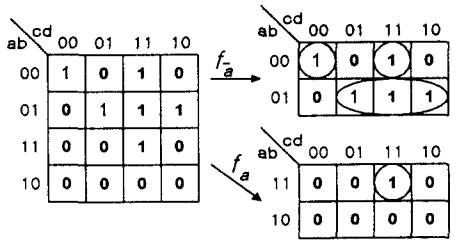

Fig. 4 Function f in which Shannon requires three complex terms

Example 3 *To get complex terms using positive Davio decomposition, $f_{\bar{a}} \oplus f_a$ is calculated first. To do this, the don't cares in $f_{\bar{a}}$ are decided to '0' so that $f_{\bar{a}} \oplus f_a$ is simplified to a constant '1'. Therefore the result becomes,*

$$\begin{aligned} f &= a \cdot (f_{\bar{a}} \oplus f_a) \oplus f_a \\ &= a \cdot 1 \oplus (\overline{bcd} \oplus \overline{bcd} \oplus \overline{bcd}) \end{aligned}$$

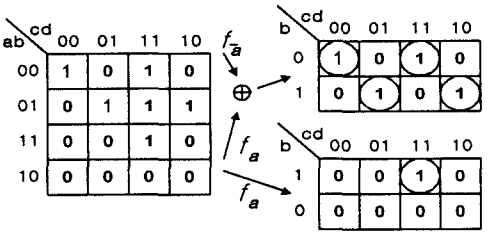

Fig. 5 Function f in which negative Davio expansion needs two complex terms

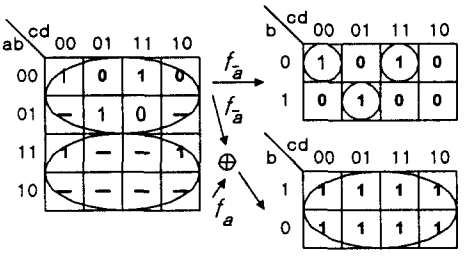

Fig. 6. When $f_{\bar{a}} \oplus f_a$ is calculated first, don't cares in $f_{\bar{a}}$ are fixed to 0 resulting in three complex terms

which includes 3 complex terms.

However, if $f_{\bar{a}}$ is calculated first, the result becomes as in figure 7 where only two complex terms appear as follows:

$$\begin{aligned} f &= a \cdot (f_{\bar{a}} \oplus f_a) \oplus f_{\bar{a}} \\ &= a \cdot .(\bar{c} + d) \oplus (\bar{b} \oplus c \oplus d). \end{aligned}$$

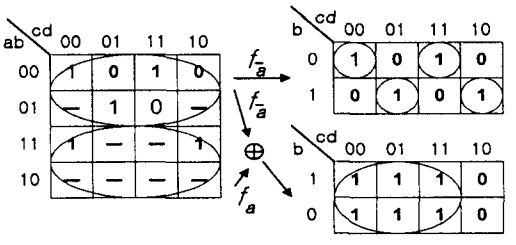

Fig. 7 When $f_{\bar{a}}$ is calculated first, it results in two complex terms finally

Therefore, to generate better results both cases should be considered. In this example, we can observe that the utilization of don't cares at the first part of the calculation of positive(or negative) Davio decomposition puts restrictions on the remaining calculation of the decomposition. In other words, the final output depends on the order of calculations. Moreover, it requires the change of the subfunctions in each step of the minimization procedure, which means heavy costs in computation time and space. To overcome this limitation, an approximation algorithm is proposed as in figure 8 in which don't cares in $f_{\bar{v}}(f_v)$ of positive(negative) Davio decompositions are ignored in the estimation of the number of complex terms for f to make each step of the calculation independent of the other steps. Again, the cost of a new node is calculated by adding smaller two costs. Also note that no changes are made to the functions, and hence no new nodes are added to the existing decision diagram.

```
cost_dc(d, dc)
{
/* returns the no. of complex terms needed */
/* d = an ETDD node, dc = don't care terms */
/* d → then, d → else : cofactors of d */
/* d → xor = d → then ⊕ d → else */
if(d+dc=1) return 0;
if(d implies dc) return 0;
if(d is the last variable in the path) return 1;
  l = cost(d → then);
  r = cost(d → else);
  l_d = cost_dc(d → then, dc → then);
  r_d = cost_dc(d → else, dc → else);
  x_d^1 = cost_dc(d → xor, dc → then);
  x_d^2 = cost_dc(d → xor, dc → else);
/* select two branches with smaller costs */
  return min(l_d + r_d, l + x_d^2, r + x_d^1);
}
```

Fig. 8 Estimation of the number of complex terms with don't cares

For the example in figure 6 , the function *cost_dc()* compares 3 sets of minimization as in figure 9. Note that negative decomposition needs only two complex terms which is optimal in this case.

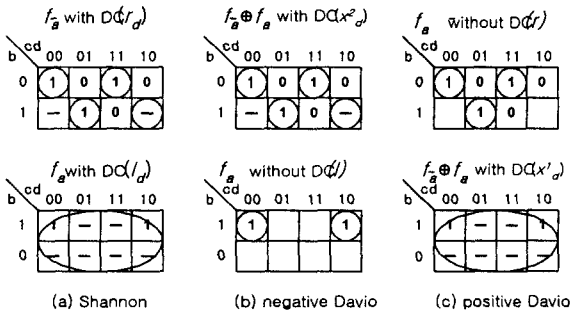

Fig. 9 Result of the approximation in three Davio expansions. Blank boxes in (b) and (c) denote don't care minterms ignored and they are treated as 0's.

4. Generation of OR/XOR collectable complex terms

To exploit the above cost function in the synthesis for complex terms, the collection of complex terms with inclusive-OR operation is considered as shown in figure 2. First, complex terms which lie in ON-set of the function are obtained by expanding SOP cubes. In figure 10, P denotes a set of prime cubes of the function f. For the cubes in P, it is tested whether it reduces *cost_dc* value when it is added to the don't care of f. If so, the prime cube is added into the set of ON-set complex terms and finally the don't care of f becomes its external don't cares and the ON-set complex terms. Then the ON-set complex terms together with external don't cares are applied to *gen_EXOR_terms()* in figure 11 as don't cares to generate complex terms in EXOR-plane.

The procedure for generating EXOR-plane terms is basically same as *cost_dc()* in figure 8 except that the expression for the path from the root to a leaf is generated. Since each node is associated with a specific decomposition type, the expression of the path can be obtained by expanding the expressions of each node. For example, if a node selects $l_d + r_d$(*Shannon decomposition*) *in cost_dc()*, the subexpressions corresponding to the node become 'v · (subexpression for d->then)' and '$\bar{v}$ · (subexpression for d->else)'. However these subexpressions can be further simplified. When a cofactor is a constant, two terms are combined to make one complex term. Without loss of generality, assume that $f_v = 1$

```
gen_OR/XOR_term(f,dc)
{
/* f = a Boolean function */
/* dc = external don't cares of f */
Let P = a prime cube set of f
for all cube c ∈ P {
   if(cost_dc(f,dc) > cost_dc(f,dc+c)) {
      OR_terms = OR_terms Uc;
      dc = dc + c;
   }
}
/* Num of OR_plane terms=| OR-terms | */
/* Num of EXOR_plane terms=cost_dc(f,dc) */
/* Now generate EXOR_plane complex terms */
gen_EXOR_terms(f,dc)
}
```

Fig. 10 Overall algorithm for generating complex terms in OR_plane and EXOR_plane

Then,

$$f = v \cdot f_v + \bar{v} \cdot f_{\bar{v}} = v + \bar{v} \cdot f_{\bar{v}} = v + f_{\bar{v}}$$

In this case, the subexpression for the above example becomes 'v + (subexpression for *d->else*)'. Also if $f_v \oplus f_{\bar{v}} = 1$ the positive and negative Davio decompositions are simplified to get reduced number of complex terms as:

$$f = v \oplus f_{\bar{v}} = f_v \oplus \bar{v}$$

5. Experimental Results

Suggested algorithm in this paper has been implemented in C and run on Pentium 300MHz UNIX system for experimentation. The benchmarks for incompletely specified functions are taken from [10].

First, we consider fully specified functions. Table 1 shows the comparison of the number of complex terms obtained by our algorithm with the results of [5]. These benchmarks do not contain external don't cares, so the terms generated in the OR-plane has been used as don't cares to the terms in EXOR-plane. To compare the results, benchmarks which appear in [5] are listed and it showed about 30% improvement. In Table 1, *vg2* showed considerable reduction in the number of complex terms compared to others. The reason comes from the fact that *vg2* is a nearly unate function and it can be optimized with OR-operations rather than XOR-operations. 14 ON-Set complex terms played a major role in reducing the total number of complex terms. The column OR shows the number of OR-plane terms and the column EXOR denotes the number of EXOR-plane terms. Note that unate or nearly unate functions tend to generate more ON-set complex terms, especially as in *vg2*. More benchmark results are given in Table 2. Since results from other methods are not available, comparison is made to the number of product terms in the input file.

```
gen_EXOR_term(d,dc)
/* generates EXOR_terms for a ETDD node d */
/* dc = ETDD node of don't cares */
{
   if(d is terminal node) {
        generate the path from the root to d;
        return;
   }
   l = cost(d → then);
   r = cost(d → else);
   x = cost(d → xor);
   ld = cost_dc(d → then, dc → then);
   rd = cost_dc(d → else, dc → else);
   xd1 = cost_dc(d → xor, dc → then);
   xd2 = cost_dc(d → xor, dc → else);
   select minimum of (ld + rd, l + xd2, r + xd1);
   if(ld + rd is selected) {
       /* use Shannon expansion */
       gen_EXOR_terms(d → then, dc → then);
       gen_EXOR_terms(d → else, dc → else);
   } else if(l + xd2 is selected) {
       /* use negative Davio expansion */
       gen_EXOR_terms(d → then, 0);
       gen_EXOR_terms(d → xor, dc → else);
    } else if(l + xd1 is selected) {
       /* use positive Davio expansion */
       gen_EXOR_terms(d → else, 0);
       gen_EXOR_terms(d → xor, dc → then);
    }
}
```

Fig. 11 Algorithm for the generation of EXOR-plane complex terms

6. Discussions and future research

In this paper, the manipulation of ETDDs to minimize functions with don't cares for the generation of complex terms is discussed. Davio decompositions with don't cares generate different results by the method (and/or the order) of matching specific values to the don't care terms in each of f_v, $f_{\bar{v}}$ and $f_v \oplus f_{\bar{v}}$. Moreover, it requires changes in the functions in each step of the minimization procedure which results in large overhead in computation time and space.

To overcome this problem, an approximation algorithm

Table 1 Comparison of the number of complex terms for completely specified functions

name	[5]	our result			
		OR	EXOR	total	time (sec)
5xp1	33	0	24	24	0.41
card4	30	0	20	20	0.46
clip	57	23	29	52	1.86
clog8	84	2	74	76	3.04
cmlp4	54	6	63	69	2.42
cnrm	52	0	55	55	1.83
cu	15	1	16	17	0.62
f51m	30	0	21	21	0.42
inc	26	4	24	28	0.47
mlp3	17	0	17	17	0.24
rd53	13	0	9	9	0.15
rd73	36	7	17	24	0.63
sao2	26	0	28	28	0.80
t481	18	0	10	10	15.22
vg2	179	14	15	29	0.50
total	670	56	423	479	

Table 2 Experimental results for more benchmarks

name	#product terms	our result			
		OR	EXOR	total	time (sec)
apex4	1732	544	270	814	28.97
con1	9	4	3	7	0.07
duke2	242	53	111	164	10.92
misex1	32	5	15	20	0.17
misex2	29	0	27	27	1.00
misex3c	255	196	21	217	68.37
sqrt8	30	0	14	14	0.25
squar5	85	0	19	19	0.18
table3	645	305	214	519	61.23
table5	606	343	66	409	95.71
xor5	16	0	1	1	0.05

is proposed with its applications to the synthesis of complex terms. The experiment shows about 30% improvement compared to existing methods and currently the algorithm is being tested with broader class of benchmarks. With further refinements in the implementation, more improved results are expected.

For future works, refinement of generating OR/EXOR-plane complex terms for multi-output functions will be studied. In this paper, only previously generated complex terms of other subfunctions are considered exhaustively which can be time consuming and may not lead to an optimized result. Also the development of optimal ordering techniques and their application to the problem of complex term generation can be studied. Even though the variable ordering for completely specified functions has been studied in [6], the ordering for incompletely specified functions could be much more difficult.

Also, to reduce the delay time of the output signal, the cells for collecting the complex terms are to be modified. When the complex terms are collected in a tree format instead of collecting serially, the time for the collection could be reduce from $O(n)$ to $O(\log n)$, where n is the number of complex terms.

감사의 글

본 논문은 정보통신부의 '99년도 정보통신 우수시범학교 지원사업에 의해 연구되었음

참 고 문 헌

[1] ATMEL Corporation CMOS Integrated Circuit Data Book. ATMEL, 1994.

[2] U. Kebschull, E. Schubert, and W. Rosentiel. "Multilevel logic synthesis based on functional decision diagram", In EDAC, page 43-47 ,1992.

[3] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. "Efficient representation and manipulation of switching functions based on ordered kronecker functional decision diagrams", In Design Automation Conference, pages 415-419, 1994.

[4] L. F. Wu and M. A. Perkwski, "Minimization of permuted Reed-Muller trees for cellular logic programmable gate arrays", H. Gruenbacher and R, Hartenstein (eds.), LNCS, pages 78 -87, 1993.

[5] A. Sarabi, N Song, M.Chrzanowska-jeske, and M. A . perkowski, "A comprehensive approach to logic synthesis and physical design for two-dimensional logic arrays", In Design Automation Conference, pages 32 1-326, June 1994.

[6] G. Lee, "Logic synthesis for cellular architecture FPGAs using BDDs", In ASP-DAC'97, pages 253-258, january 1997.

[7] K. K. Maitra, "Cascaded switching networks of two-input flexible cells", IRE Trans. Electron. Com., pages 136-143, 1962.

[8] T. Sasao, editor, "AND-EXOR Expressions and Their Optimization", Kluwer Academic Publisher, 1993.

[9] M. A. Perkowski, M. Chrzanowska - Jeske, A. Sarabi and I, "Multi-level logic synthesis based on kronecker and boolean ternary decision diagrams for incompletely specified function", In VLSI Designs, 3(3), pges 301-313, 1995.

[10] T. Kozlowski, E. L. Dagless, and J. M. Saul, "An enhanced algorithm for the minimization of exclusive-or sum-of-products for incompletely specified functions", In ICCD, pages 244-249, 1995.

저 자 소 개

이 동 건 (李 東 建)

1989년 호남대 전산통계학과 졸업(학사). 1994년 전남대 대학원 전산통계학과 석사. 1998년-현재 전남대 대학원 전산통계학과 박사과정. 1995 - 현재 광양대학 전산과 조교수. 관심분야는 VLSI/CAD, 논리합성.

Tel : 0667-760-1471, Fax : 0667-763-9009

E-mail : dklee96@chollian.net

정 미 경 (鄭 美 京)

1987년 전남대 전산통계학과 졸업(학사). 1989년 전남대 대학원 전산통계학과 석사. 2000년 전남대 대학원 전산통계학과 박사. 관심분야는 VLSI/CAD, 논리합성, 인공지능.

Tel : 062-530-0147, Fzx : 062-530-3439,

E-mail : mgjung@chonnam.chonnam.ac.kr

이 귀 상 (李 貴 相)

1980년 서울대 공대 전기공학과 졸업(학사). 1982년 서울대 대학원 전자계산기공학과 석사. 1982년 금성통신 연구소 근무 1991년 Pennsyivania 주립대학 전산학과박사. 1984년~현재 전남대 전산학과 부교수. 관심 분야는 VLSI/CAD, 멀티미디어 시스템 및 통신, 테스팅, 논리합성.

Tel : 062-530-3425, Fzx : 062-530-3439,

E-mail : gslee@chonnam.chonnam.ac.kr