Logic Synthesis for a Fine-grain, Dynamically Reconfigurable FPGA

Nan Zhuang and Peter Y.K. Cheung[1]

Abstract

This paper describes an algorithm that combines logic synthesis and technology mapping specifically for Xilinx's XC6200, a new family of fine-grain, dynamically reconfigurable FPGA. The algorithm employs a BDD representation of the logic function, and Genetic Algorithm (GA) is used to find a good decomposition variable ordering. It also exploits the architectural features of the XC6200 to minimise the number of cells required to implement a given function. Results on benchmark circuits show that the new algorithm performs significantly better than other synthesis tools in the majority of cases.

Indexing terms: *Logic synthesis, FPGAs, Technology Mapping, Genetic Algorithm*

1. Introduction

The use of FPGAs in many applications has been increasing rapidly in recent years. In addition to the two popular types of commercial FPGAs, namely, look-up-table based and multiplexer based, Xilinx recently introduced a new fine-grain, dynamically reconfigurable FPGA family, the XC6200. The function unit of the XC6200 architecture is as shown in Fig.1[1]. The combinational logic part of the function unit is a universal logic gate. Unlike normal multiplexers, it can implement all two variable functions and four of the three variable functions because the inputs to the multiplexers can be complemented if necessary. The most obvious advantage of this architecture is that it allows direct implementation of the XOR function $x \oplus y$, leading to efficient synthesis. Some works on the problem of synthesising Boolean networks, and expressing each node as a network of

[1]The authors are with the Dept. of Electrical and Electronic Engineering, Imperial College London, UK, SW7 2AZ

multiplexers, have been proposed[2-10]. Among these algorithms, [2,3] operated on minterms, and $mux\_synthesis$[4] is based on disjoint cubes. In $MIS-pga$[5], the multiplexer structure was exploited, thus leading to the representation of Boolean functions by Binary Decision Diagrams(BDD). Eight types of BDD-patterns corresponding to the basic functionalities associated with the Actel architecture were used to cover these BDDs. In $ASYL$[6,7], the Boolean functions are represented with ROBDDs for area optimization, and a direct mapping was performed, which had shown high benefit when compared with library based approach. $Amap$[8] and $MIS-pga(new)$[9] introduced If-Then-Else Direct Acyclic Graphs(ITE DAG) to replace BDD. The main advantage of ITE DAGs over BDDs is that duplication of cubes can be avoided. mux_decomp_opt[10] employed more general DAGs to represent the Boolean functions, in which different selector function can be used at the same level. However, all of above algorithms did not consider the availability of complementary inputs of the multiplexers except [4]. These algorithms were designed for the synthesis of Actel ACT series FPGAs. A logic synthesis package targeted specifically to XC6200 FPGAs is proposed in the paper. In our algorithm Boolean networks are represented by ROBDDs because it is more memory efficient than minterm and disjoint cube representations. Different from other published algorithms[2-10], a Genetic Algorithm (GA) is introduced to find a good decomposition variable ordering.

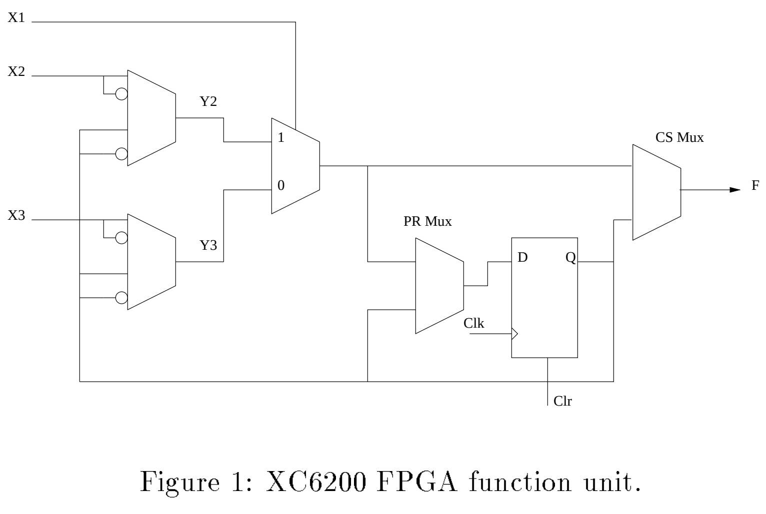

Figure 1: XC6200 FPGA function unit.

The rest of the paper is organized as follows: Section 2 describes the BDD size minimization algorithm using GA, and explains how the BDD is optimally mapped to XC6200 FPGA function units. Section 3 explains the XC6200 FPGA specific logic synthesis algorithm, $XC6200\_Syn$. Section 4 shows the results of the new algorithm when applied to published benchmark circuits, and the conclusions are given in Section 5.

2. BDD size minimization and XC6200 technology mapping

2.1 BDD size minimization

It is well known that multiplexer based structures can be directly related to BDDs[11]. [4,12] indicated that finding the optimal data-select variable ordering of the multiplexer network is strongly equivalent to finding the good variable ordering for BDDs. Since reduced ordered BDD (ROBDD) has been shown to be an efficient representation of Boolean functions for logic synthesis and verification purposes[11], it is employed in our algorithm. The first step of our synthesis algorithm is to minimize the size of the ROBDD, and then the ROBDD is mapped to Boolean networks based on XC6200's function units.

Definition 1. An OBDD is a rooted directed graph $G = (V, E)$. The vertex set V is composed of two kinds of vertex, non-terminal and terminal. Each non-terminal vertex has as attributes a pointer $index(v) \in \{1, 2, ..., n\}$ to an input variable in the set $\{x_1, x_2, ..., x_n\}$, and two children $low(v), high(v) \in V$. A terminal vertex v has as an attribute a value $value(v) \in \mathbf{B}(\mathbf{B} \in \{0, 1\})$. The edge set E is composed of negative edges and positive edges.

Definition 2. BDD size, $|BDD|$, is given by its number of non-terminal nodes. For a certain variable ordering, BDD size can be reduced with following three rules:

Deletion Rule (R1): The node v with label x_i can be deleted, if and only if the two successors $low(v)$ and $high(v)$ of v represent the same subfunction of f, i.e. the subfunction represented a v does not depend essentially on x_i.

Merging Rule (R2): Nodes v and w with label x_i can be shared, if and only if they represented the same subfunction of f.

Complement Rule (R3): For nodes v and w with label x_i, w can be replaced with v', if and only if $v = w'$.

Definition 3. A BDD is called a reduced ordered binary decision diagram(ROBDD), if no reductions can be achieved using the above rules.

Among the above rules, R3 is particular suitable for XC6200 synthesis. Unlike other multiplexer based FPGAs such as Actel, the complement multiplexer inputs can be fully exploited without using any additional logic cell as an inverter[4]. R3 is used very frequently to minimise near linear functions.

ROBDD size is extremely sensitive to the choice of the variable ordering. The existing methods of finding a good variable ordering can be classified into three categories: heuristic methods[13-15], exact methods[12,16], and GA methods[17-21]. Previous research indicated that GA methods can produce better results with reasonable CPU time for most of the benchmark circuits. In this paper a novel GA[21], which uses dynamic parameters, is employed to search for near optimal variable ordering.

Definition 4. For any Boolean function $f(x_1, x_2, ..., x_n)$, the decision variable ordering (from the root to the terminals of the data structure) can be defined with a vector $Order[n] = \{x_{k_1}, x_{k_2}, ..., x_{k_n}\}$. For a certain $Order[n]$, the size of the ROBDD, $BDD\_Size()$, is the number of the non-terminal vertex in the data structure. Our goal is to find a good variable ordering $\{x_{k_1}, x_{k_2}, ..., x_{k_n}\}$ for a given function $f(x_1, x_2, ..., x_n)$, such that $BDD\_Size()$ is minimized. The pseudocode for the dynamic GA is depicted as follows. Details of the algorithm can be found in [20,21].

```
Dynamic_Genetic_Algorithm(benchmark)
{
    for(K1 = 0, K1 <= MAX_POPULATION) {
        Random_Order();
        BDD_Size();
        Initial_Roulette_Wheel_Selection();
    }
```

```
    while(Generation_Remaining) {
        Crossover();
        Mutation();
        BDD_Size();
        random_number = rand()%100;
        if(random_number < HYBRID_RATE)Heuristic_Parent_Selection();
        else Roulette_Wheel_Parent_Selection();
        Elitism();
        if(improvement found in new generation) {
            decrease Population_Size by δ1;
            decrease Mutation_Rate by μ;
            increase Generation_Remaining γ1;
        } else {
            increase Population_Size by δ2;
            increase Mutation_Rate by μ;
            decrease Generation_Remaining by γ2;
        }
        if(Population_Size >= MAX_POPULATION)
            decrease Generation_Remaining by γ3;
        Population_Size = max(MIN_POPULATION, min(Population_Size,
                              MAX_POPULATION));
        Mutation_Rate = max(MIN_RATE, min(Mutation_Rate, MAX_RATE));
    }
    Copy_Best_Ordering();
    BDD_2_Network();
}
```

2.2 Mapping BDD to Boolean network based on XC6200 function units

Most of the functions described in the above pseudocode have been discussed in [17,20,21]. The $BDD\_2\_Network()$ function is the new contribution described in this paper. It maps the ROBDD produced with the GA to a Boolean network specifically for XC6200 FPGA cell architecture. The relationship between a ROBDD and the Boolean function it represents can be described as follow:

Corollary 1. An OBDD with root v denotes a Boolean function $f_v : \mathbf{B}^n \to \mathbf{B}$ such that:

(a) If v is a terminal vertex with $value(v) = 1$, then $f_v = 1$.

(b) If v is a terminal vertex with $value(v) = 0$, then $f_v = 0$.

(c) If v is a non-terminal vertex and $index(v) = i$, then $f_v = x'_i \dot{f}_{low(v)} + x_i \dot{f}_{high(v)}$, where, $\dot{f}_{low(v)} = f_{low(v)}$ if $edge(v, low(v))$ is positive, and $\dot{f}_{low(v)} = f'_{low(v)}$ if $edge(v, low(v))$ is negative. $\dot{f}_{high(v)}$ can be defined in a similar way.

According to Corollary 1, a BDD vertex v can be mapped to a Boolean node based on XC6200 FPGA architecture as follows:

$Node\_Mapping(v)$
{
    $if(v\ is\ a\ root\ of\ the\ BDD\ and\ the\ edge(po^2, v)\ is\ negative)$ {
        $if(edge(v, low(v))\ is\ negative)\ f_{x'_{index(v)}} = f_{low(v)};$
        $else\ f_{x'_{index(v)}} = f'_{low(v)};$
        $if(edge(v, high(v))\ is\ negative)\ f_{x_{index(v)}} = f_{high(v)};$
        $else\ f_{x_{index(v)}} = f'_{high(v)};$
    } $else$ {
        $if(edge(v, low(v))\ is\ negative)\ f_{x'_{index(v)}} = f'_{low(v)};$
        $else\ f_{x'_{index(v)}} = f_{low(v)};$
        $if(edge(v, high(v))\ is\ negative)\ f_{x_{index(v)}} = f'_{high(v)};$
        $else\ f_{x_{index(v)}} = f_{high(v)};$
    }
    $f_v = x'_{index(v)} f_{x'_{index(v)}} + x_{index(v)} f_{x_{index(v)}};$
}

$Node\_Mapping(v)$ is the function that is called repeatedly in $BDD\_2\_Network()$ to map the BDD to the Boolean network. For example, a BDD vertex with root v as shown in Fig. 2(a) can be mapped to a Boolean node, and implemented with a single XC6200 FPGA unit as shown as in Fig. 2(b). The computational complexity of $BDD\_2\_Network()$ is $O(|BDD|)$, which is the complexity of traversing the BDD. The mapping algorithm above is designed specifically for the XC6200 FPGAs. For a given root node v, $Node\_Mapping()$ not only collapses the ancestor negative edges, it also absorbs the descentant negative edges.

3. Synthesis procedures for XC6200 FPGAs

After map the minimized ROBDD to a Boolean network, every node can be mapped to a single XC6200 FPGA unit. However, the timing performance of the network often need further optimisa-

[2] po is primary output.

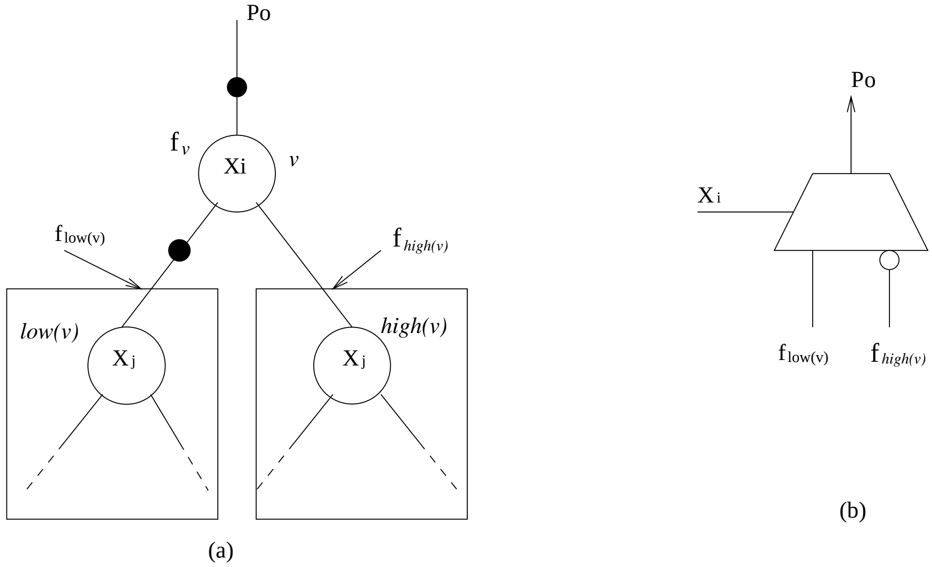

Figure 2: Realize a BDD vertex with a XC6200 unit.

tion. To describe our algorithm, the following Boolean function is used as an example throughout.

$$f(x_1, x_2, x_3, x_4, x_5) = x_1 x_2' + x_1 x_3 + x_1 x_4' x_5 \quad (1)$$

After variable ordering using GA, the ROBDD representation of Eq. (1) is as shown in Fig. 3(a). Using The function $BDD\_2\_Network()$, the ROBDD representation of $f(x_1, x_2, x_3, x_4, x_5)$ is mapped to XC6200 units as shown in Fig. 3(b). To improve the timing performance of this circuit, the following network refinements were introduced.

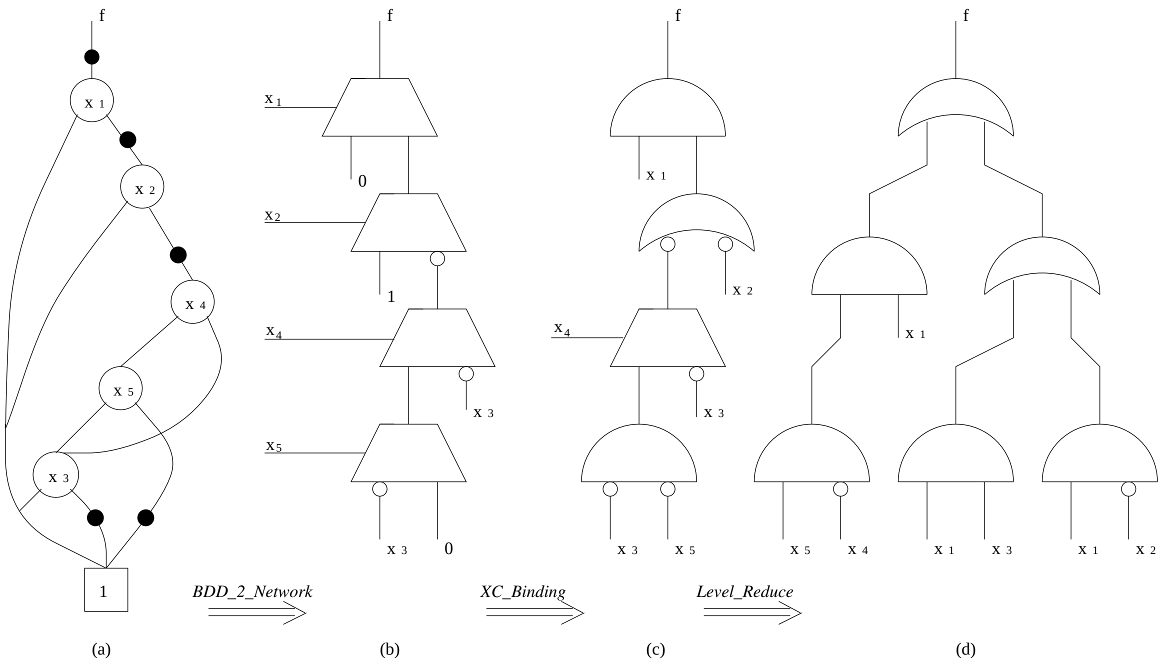

Figure 3: Performace refine based on XC6200 FPGAs

3.1 Library binding

$XC\_Binding()$ maps all multiplexers with a constant input to an equivalent two input gate from XC6200's basic function library. This results in more efficient routing when using the current version of Xilinx tools. The result of $XC\_Binding()$ is shown in Fig. 3(c).

3.2 Level reduction

Fig. 3(c) shows that x_1 has been factorized. Factorization can often reduce the size of the Boolean network at the expense of increased levels of logic. The level reduction algorithm, $Level\_Reduce()$, locates nodes on the critical path that are in factorized form. The factored variables are then distributed. Using the distributive property, the circuit in Fig. 3(c) can be improved to that in Fig. 3(d), with the logic level reduced by one at the expense of two extra logic cells. If a node is located on the critical path but not in a factorized form, it is collapsed and checked for the following conditions: (i) if it is composed of a single cube; or (ii) if it is composed of single-literal cubes; or (iii) if it is composed of cubes of disjoint support. Nodes that satisfy one of the above conditions are decomposed with AND-OR decomposition[23]. This produces an AND-OR tree structure which has fewer levels of logic. As will be seen later, for circuits such as the benchmark $misex2$, the function $Level\_Reduce()$ can decrease the levels of the critical path from 11 to 7, while increasing the number of XC6200 cells by only 12%. This algorithm can be used iteratively until no improvement is found.

After level reduction, the Boolean network is verified against the input circuit using SIS's verification routines.

The entire synthesis procedure for XC6200 FPGA as shown in Fig. 4 is collected into $XC6200\_syn$ as an add-on package to SIS.

4. Results

The numerous steps described in the previous section were combined together as $XC6200\_Syn$, a XC6200 FPGA logic synthesis add-on package to Berkeley's SIS environment. Since no other

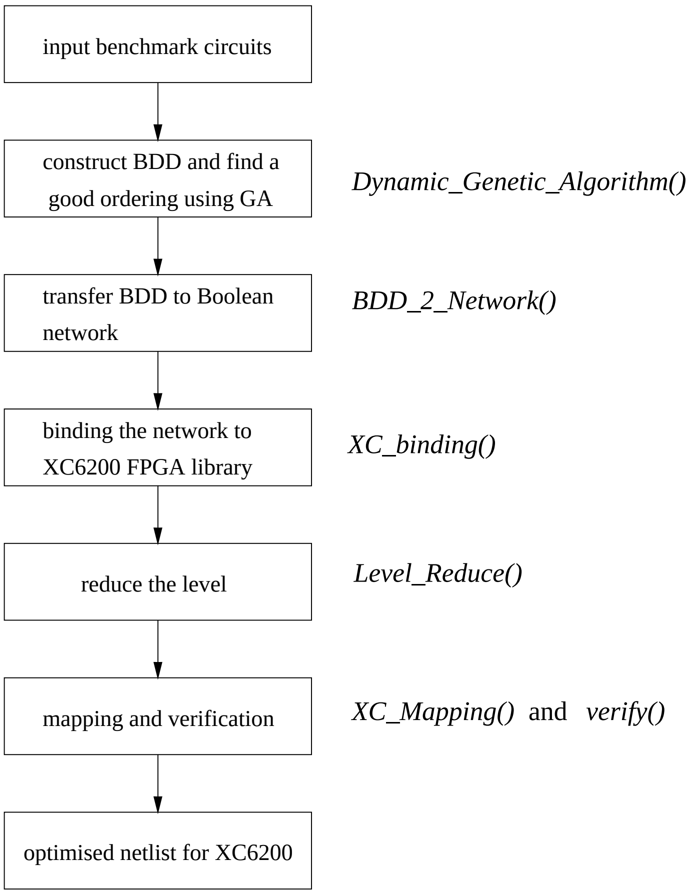

Figure 4: Schematic diagram of XC6200 synthesis procedure

synthesis results on XC6200 has been published, the performance of $XC6200\_Syn$ is compared to that of SIS (without our package) and Synopsys. All 39 LGSynth93 benchmark circuits were tested. For 24 of them, $XC6200\_Syn$ yielded better results than both SIS and Synopsys.

To understand better the type of circuits that $XC6200\_Syn$ is more suited for, all LGSynth93 benchmark circuits were divided into three catagories according to the concepts of *circuit density* as defined below.

Definition 6. The density of a circuit is defined as follows:

$$\rho = \frac{|BDD|}{i * o} \qquad (2)$$

Where, $|BDD|$ is the ROBDD size of the circuit. i and o is the number of the inputs and outputs of the circuit.

Definition 7. A circuit is

(a) a *simple* circuit if $|BDD| \leq 80$;

(b) a *high density* circuit if $\rho \geq 0.7$ and $|BDD| > 80$;

(c) a *low density* circuit if $\rho < 0.7$ and $|BDD| > 80$.

Table 1 shows the results for all simple circuits in the LGSynth93 benchmarks. Column "U" indicates the number of XC6200 units used; "L" is the level of logic on the critical path; "D" shows the estimated critical path delay of the mapped circuit; "CPU" is the CPU time running on a SUN Sparc 10 workstation. From Table 1, it can be seen that $XC6200\_Syn$ yields better or similar results to Synopsys and SIS except $rd84$. For very simple circuits such as $con1$ and $xor5$, $XC6200\_Syn$ performs identically to Synopsis.

Table 1. Testing results for small benchmarks

Circuits			**XC6200_Syn**				**SIS**				**Synopsys**		
	i	o	U	L	D	CPU	U	L	D	CPU	U	L	D
5xp1	7	10	**40**	**6**	**27.8**	**3.4**	73	18	78.9	12.5	69	8	36.3
9sym	9	1	**23**	**8**	**35.2**	**8.7**	169	15	77.9	64.8	30	8	33.3
b12	15	9	**53**	**8**	**27.8**	**31.5**	54	7	29.6	6.5	57	6	24.0
clip	9	5	**73**	**8**	**38.9**	**15.7**	100	19	88.3	27.1	79	10	43.5
con1	7	2	**12**	**4**	**14.8**	**0.7**	12	4	16.6	2.7	12	4	14.8
cordic	22	2	**47**	**22**	**98.4**	**231.6**	40	9	36.1	2025.3	49	8	30.5
inc	7	9	**68**	**6**	**27.8**	**6.9**	82	19	91.6	15.8	90	8	31.7
misex1	8	7	**32**	**5**	**21.3**	**2.5**	36	12	56.8	6.1	46	6	25.0
rd53	5	3	**15**	**4**	**18.5**	**3.7**	22	7	28.7	4.2	12	5	21.3
rd73	7	3	**29**	**6**	**28.7**	**4.4**	40	12	53.7	13.7	20	7	29.6
rd84	8	4	**40**	**7**	**33.3**	**14.0**	88	19	87.7	37.3	26	7	30.5
sqrt8	8	4	**31**	**7**	**29.6**	**2.7**	49	12	55.0	5.8	37	8	31.4
squar5	5	8	**30**	**4**	**18.5**	**5.1**	46	10	48.4	5.4	40	7	25.9
t481	16	1	**19**	**15**	**69.6**	**41.9**	15	4	14.8	33.1	19	5	19.4
xor5	5	1	**4**	**3**	**11.1**	**1.2**	4	4	17.6	0.9	4	3	11.1

The density ρ in Definition 6 indicates the relative complexity of a circuit. For high density circuits as defined here, $XC6200\_Syn$ performs better; for low density circuits, $XC6200\_Syn$ always performs worse than either Synopsys or SIS except $ex5p$. The test results for all the high density benchmark circuits and three of the low density circuits, $ex5p$, $duke2$, and $misex2$ are listed in Table 2. For example, for $apex4$, which has a density of 5.18, $XC6200\_Syn$ is considerably better than both Synopsys and SIS. On the other hand, for $misex2$ and $duke2$, with densities of 0.18 and 0.52 respectively, $XC6200\_Syn$ performs worse than Synopsys. Another 11 benchmarks with the property similar to $misex2$ and $duke2$ are omitted from Table 2.

Table 2. Testing results for large benchmarks

Circuits				**XC6200_Syn**				**SIS**				**Synopsys**		
	i	o	ρ	U	L	D	CPU	U	L	D	CPU	U	L	D
alu4	14	8	5.03	**555**	**13**	**69.1**	**399.4**	183	20	92.1	1323.3	761	17	86.6
apex4	9	19	5.20	**885**	**8**	**83.0**	**390.9**	1919	123	776.3	3415.5	1918	15	92.3
bw	5	28	0.70	**95**	**4**	**41.2**	**10.3**	123	22	119.8	32.2	138	7	42.6
duke2	22	29	0.52	**333**	**17**	**86.7**	**78.4**	320	20	110.4	97.7	294	7	53.4
ex1010	10	10	10.5	**1042**	**9**	**109.7**	**409.4**	2379	17	78.5	4409.6	1925	16	102.2
ex5p	8	63	0.47	**240**	**7**	**47.1**	**90.1**	274	16	78.4	280.6	289	7	53.7
misex2	25	18	0.18	**90**	**7**	**31.5**	**7.8**	81	8	35.2	11.1	82	8	33.3
misex3	14	14	2.43	**474**	**13**	**74.9**	**582.4**	522	23	108.1	887.1	686	16	79.3
misex3c	14	14	1.95	**372**	**13**	**67.4**	**147.9**	389	51	223.5	993.3	461	15	60.1
sao2	10	4	2.00	**78**	**9**	**58.3**	**7.8**	99	21	88.1	24.7	114	12	47.2
seq	41	35	0.88	**1259**	**21**	**119.4**	**3367.3**	1399	28	146.9	1748.5	1321	19	107.1
table3	14	14	3.83	**747**	**13**	**99.4**	**142.4**	693	87	463.8	1314.1	891	13	88.5
table5	17	15	2.61	**666**	**16**	**100.3**	**151.7**	703	25	131.2	1162.5	782	17	78.6

It is obvious from Table 2 that $XC6200\_Syn$ is suitable for synthesizing high density circuits where the product $i * o$ is small. This is because $XC6200\_Syn$ represents Boolean networks as BDDs, and the optimization procedure relies heavily on the ability of the algorithm to find an optimal variable order. This becomes increasingly difficult as $i * o$ product is large. In contrast, both SIS and Synopsys use an algorithm that is independent of the number of input and output nodes, but yields good results as long as the number of prime implicants is reasonably small. Note also that $XC6200\_Syn$ is considerably better than SIS on timing performance. This is because SIS initially performs multilevel minimization, then performs IF-THEN-ELSE decomposition one node at a time. For complex functions, this may result in a very long path. The results here also show that the number of logic levels only provides a rough estimate on timing. Delay in a mapped circuit includes fanout and routing delays that are not reflected in the levels of logic.

Finally, all benchmark circuits reported in [4] were tested and $XC6200\_Syn$ was found to use 39.5% fewer cells with 1% improvement in the total levels of logic.

5. Conclusion

$XC6200\_Syn$, an additional package to Berkeley's SIS environment for XC6200 FPGA logic synthesis, is proposed in this paper. Since Boolean functions are represented as BDDs, memory usage is more efficient and the quality of synthesis is no longer limited by the number of prime implicants. Genetic Algorithm is employed to solve the NP-complete problem of BDD variable ordering. This enables a near-optimal result to be obtained within reasonable CPU time. Finally, by combining logic synthesis with technology mapping, and by exploiting the inherent architecture of the $XC6200$ function unit, our algorithm yields better results for the major of the LGSynth93 benchmark circuits in terms of size and delay when compared with SIS and Synopsys.

6. Acknowledgement

This project is supported by Mentor Graphics Corparation, USA.

References

[1] Xilinx Inc., 'XC6200 FPGA Family', 1995.

[2] Gorai, R.K., and Pal, A., 'Automated synthesis of combinational circuits by tree network of multiplexers', *Proc. of IEEE 3rd Intl. Conf. VLSI design*, pp. 380–386, 1990.

[3] Wu, H., Zhuang, N., Perkowski, M.A., 'Synthesis of Multiplexer Directed-Acyclic-Graph Network with Application to FPGAs & BBDs', *International Workshop on Logic Synthesis*, U.S.A., May, 1993.

[4] Schafer, I., and Perkowski, M.A., 'Synthesis of multilevel multiplexer circuits for incompletely specified multioutput Boolean functions with mapping to multiplexer based FPGA's', *IEEE Trans. on CAD*, Vol. 12, No. 11, pp. 1655–1664, Nov., 1993.

[5] Murgai, R., Nishizaki, Y., Shenoy, N., Brayton, R.K., and Sangiovanni-Vincentelli, A.L., 'Logic synthesis for programmable gate arrays', *Proc. of 27th ACM/IEEE Design Automat. Conf.*, pp. 620–625, 1990.

[6] Besson, T., Bousouzou, H., Crates, M., and Saucier, G., 'Synthesis on multiplexer-based programmable devices using(ordered) binary decision diagrams', *Proc. EUROASIC*, pp. 8–13, 1992.

[7] Besson, T., Bousouzou, H., Crates, M., and Saucier, G., 'Synthesis for FPGA using binary decision diagrams', *IEEE 1992 Int. Conf. on Computer Design: VLSI in Computers and Processors*, pp. 163–167, 1992.

[8] Karplus, K., Amap: 'A technology mapper for selector-based Field-Programmable gate array', *Proc. of 28rd ACM/IEEE Design Automat. Conf.*, pp. 244–247, 1991.

[9] Murgai, R., Shenoy, N., Brayton, R.K., and Sangiovanni-Vincentelli, A.L., 'An improved synthesis algorithm for multiplexer-based PGAs', *Proc. of 29th ACM/IEEE Design Automat. Conf.*, pp. 380–386, 1992.

[10] Thakur, S., Wong, D.F., and Krishnamoorthy, S., 'Delay minimal decomposition of multiplexers in technology mapping', *Proc. of 33rd ACM/IEEE Design Automat. Conf.*, June, 1996.

[11] Bryant, R., 'Graph-based algorithms for Boolean function manipulation', *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug., 1986.

[12] Ishiura, N., Sawada, H., and Yajima, S., 'Minimization of binary decision diagrams based on exchanges of variable', *Proc. of IEEE Int. Conf. Comput.-Aided Design*, pp. 472–475, Nov., 1991.

[13] Wegener, I., 'The size of reduced OBDD's and optimal read-once branching programs for almost all boolean functions', *IEEE Trans. Comput.*, Vol. C-43, No. 11, pp. 1262–1269, Nov., 1994.

[14] Aziz, A., Tasiran, S., and Brayton, R.K., 'BDD variable ordering for interating finite state machines', *Proc. of 31st ACM/IEEE Design Automat. Conf.*, pp. 283–288, June, 1994.

[15] R. Rudell, 'Dynamic variable ordering for ordered binary decision diagrams', *Proc. of IEEE Int. Conf. Comput.-Aided Design*, pp. 42–47, Nov., 1993.

[16] Friendman, S.J., and Supowit, K.J., 'Finding the optimal variable ordering for binary decision diagrams', *IEEE Trans. Comput.*, Vol. C-39, No. 5, pp. 710–713, May, 1990.

[17] Almaini, A.E.A., and Zhuang, N., 'Using genetic algorithm for the variable ordering of Reed-Muller binary decision diagrams', *Microelectronics Journal*, Vol. 26, No. 5, pp. 471–480, 1995.

[18] Becker, B., and Drechsler, R., 'OFDD based minimization of fixed polarity Reed-Muller expressions unsing hybrid genetic algorithm', *Proc. of IEEE Int. Conf. Comput.-Aided Design*, pp. 106–110, Nov., 1994.

[19] Drechsler, R., Becker B., and Göckel, N., 'A genetic algorithm for variable ordering of OBDDs', *Int Workshop on Logic Synthesis*, pp. 5.55–5.64, 1995.

[20] Almaini, A.E.A., Zhuang, N., and Bourset, F., 'Minimisation of multioutput Reed-Muller binary decision diagrams using hybrid genetic algorithm', *Electronics Letters*, Vol. 31, No. 20, pp.1722-1723, 1995.

[21] Zhuang, N., Benten, M.S.T., and Cheung, P.Y.K., 'Improved variable ordering of BDDs with novel genetic algorithm', *Proc. of IEEE Int. Symposium on Circuits and Symtems*, pp. III-414–417, May, 1996.

[22] De Micheli, G., *Synthesis and Optimisation for Digital Circuits*, McGraw-Hill, 1994.

[23] Sentovich, E.M., Singh, K.J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P.R., and Brayton, R.K., 'SIS: A system for sequential circuit synthesis', Technical report UCB/ERL M92/41, University of California, Berkeley, May 1992.