Majority Gate Decomposition

Yo-Chi Lee (李友岐) r06921048, Jia-Shiuan Chen (陳家暄) r06921061
Graduate Institute of Electrical Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT
The decomposition of Boolean function is an important operation in logic synthesis. Finding a good decomposition lead to less circuit complexity and simpler physical design solutions. There are some research focusing on bi-decomposition in past years. [1] In this paper, however, we focus on tri-decomposition. Besides, we don't consider the existence of common variables between three partitions, which means three partitions contain disjoint support. If a function f (X) is tri-decomposable under the variable partition X_A, X_B, X_C on X, then it can be written as h($f_A(X_A)$, $f_B(X_B)$, $f_C(X_C)$) for some functions h, f_A , f_B and f_C, where $X_A \cap X_B = \varnothing$, $X_A \cap X_C = \varnothing$, $X_B \cap X_C = \varnothing$ and $X = X_A \cup X_B \cup X_C$. In this paper, we consider function h to be a 3-input majority gate. In order to check whether a function f (X) can be decompose by a 3-input majority gate, we propose a method based on interpolation and incremental SAT solving.

General Terms
logic synthesis, algorithms

Keywords
majority gate, decomposition, variable partition, quantum-dot cellular automata, satisfiability, Craig interpolation

1. INTRODUCTION
Functional decomposition is a fundamental operation on Boolean functions that decomposes a large function f on variables X into a set of small sub-functions. The trivial case is splitting X into certain sets, but one of them like X_A equal to the origin X. There are some researches about the bi-decomposition in past few years, such as [1]. They tried to use a 2-input logic gate like OR, AND and XOR to represent the given function. According to [1], the definition of decomposability of a given function is as following. If a function f (X) is bi-decomposable under the variable partition X_A, X_B, X_C on X, then it can be written as h($f_A(X_A, X_C)$, $f_B(X_B, X_C)$) for some functions h, f_A , and f_B, where $X = X_A \cup X_B$ and $X_C = X_A \cap X_B$. Usually, it is better that $|X_A| \approx |X_B|$ and $|X_C| \approx 0$. The more the bi-decomposition is balanced and disjoint, the better the quality of a bi-decomposition is. [1]

In this paper, however, we change the problem from bi-decomposition to tri-decomposition. Besides, we only consider the condition that all partitions have disjoint support to be a successful decomposition. Thus, there should be no common variable between each partition. Based on this, we have the following definition. If a function f (X) is tri-decomposable under the variable partition X_A, X_B, X_C on X, then it can be written as h($f_A(X_A)$, $f_B(X_B)$, $f_C(X_C)$) for some functions h, f_A , f_B and f_C, where $|X_A \cap X_B| + |X_A \cap X_C| + |X_B \cap X_C| = 0$ and $X = X_A \cup X_B \cup X_C$. In our research, we use a 3-input majority gate to represent function h, which means the origin function f (X) is the output of this majority gate while $f_A(X_A)$, $f_B(X_B)$ and $f_C(X_C)$ are the three inputs, just like Fig1 show.

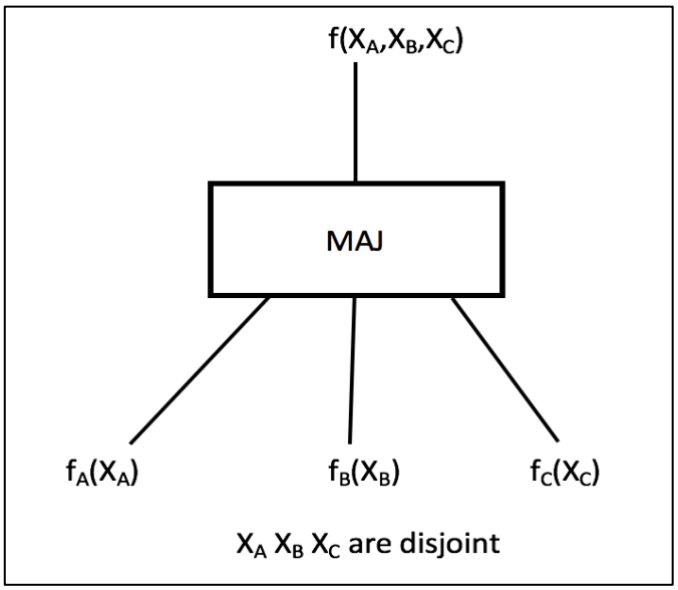

Fig1

However, why we choose majority gate rather than other primitive 3-input logic gates? The reason is related to the new nanotechnology of physical design. It is inefficient to create a majority gate with conventional CMOS technologies. Two decades ago, as a result, there are a few researches towards the logic optimization of circuits based on majority gates. However, a particular technology emerged, which is quantum-dot cellular automata (QCA). Different from CMOS, the primitive logic of QCA is the majority gate. [2] Therefore, more and more logic synthesis research based on majority gate has appeared in past ten years, such as ATPG by Majority [3] and Majority Inverter Graph [4]. There are some advantages of this new nanotechnology. For instance, QCA consume extremely lower power than CMOS, since its working principle is not based on electric current flow but on Coulomb interaction. [5]

Our research is using 3-input majority gate to decompose the given function. We propose a tri-decomposing method based on interpolation and incremental SAT solving. For the remain parts, the word majority gate represents 3-input majority gate specifically. This paper is organized as follows. Section 2 gives the preliminaries. Our proposed method is presented in Section 3. After that, we discuss our experiment results in Section 4. The conclusion of this paper is in Section 5.

2. PRELIMINARIES

2.1 Tri-Decomposition

Definition 1. Given a completely specified Boolean function f, variable x is a support variable of f if $f_x \neq f_{\neg x}$, where f_x and $f_{\neg x}$ are the positive and negative cofactors of f on x, respectively. [1]

Definition 2. A completely specified function f(X) is tri-decomposable under variable partition $X = \{X_A|X_B|X_C\}$, if it can be written as $h(f_A(X_A), f_B(X_B), f_C(X_C))$ for some functions h, f_A, f_B and f_C, where $X_A \cap X_B = \varnothing$, $X_A \cap X_C = \varnothing$, $X_B \cap X_C = \varnothing$ and $X = X_A \cup X_B \cup X_C$. The decomposition is called successful only when three partition contains disjoint support. If we are not able to find a partition method where $|X_A \cap X_B| + |X_A \cap X_C| + |X_B \cap X_C| = 0$ to decompose function f(X), then this function is not tri-decomposable.

2.2 Refutation Proof

Definition 3. Assume literal v is in clause c1 and ¬v in c2. A resolution of clauses c1 and c2 on variable v yields a new clause c containing all literals in c1 and c2 except for v and ¬v. The clause c is called the resolvent of c1 and c2, and variable v the pivot variable. [1]

Theorem 1. [6] If a CNF instance is unsatisfiable, then there exists an empty clause after a sequence of resolution steps.

Usually, not all clauses result in the empty resolvent. We only need some of them to derive an empty clause by several resolution steps. Modern SAT solvers such as MiniSat can produce a refutation proof if a CNF instance is unsatisfiable. Fig2 shows an example of refutation proof of an unsatisfiable instance. [7]

2.3 Craig's Interpolation

Theorem 2 (Craig Interpolation Theorem). [8] For any two Boolean formulas φ_A and φ_B with $\varphi_A \wedge \varphi_B$ unsatisfiable, then there exists a Boolean formula $\varphi_{A'}$ referring only to the common input variables of φ_A and φ_B such that $\varphi_A \Rightarrow \varphi_{A'}$ and $\varphi_{A'} \wedge \varphi B$ is unsatisfiable.

The Boolean formula $\varphi_{A'}$ is referred to as the interpolant of φ_A and φ_B. We assume that φ_A and φ_B are both in CNF. As a result, a refutation proof of $\varphi_A \wedge \varphi_B$ is available from a modern SAT solver. Fig3 is the Boolean space of φ_A, φ_B and $\varphi_{A'}$. [1]

2.4 Majority Gate Logic

Definition 4. A majority gate outputs true if and only if more than 50% of its inputs are true.

Given a 3-input majority gate, the output is true only when more than two inputs are true. As a result, the output is false if there is only one or zero input true, which means more than two inputs are false. The truth table of majority gate is just like Fig2. Observing the truth table, we can find that if f=Maj(a, b, c), then f can be expressed as (ab+bc+ac) in SOP.

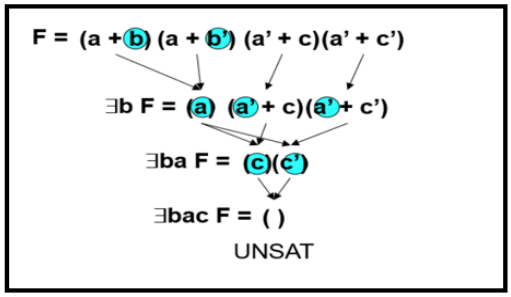

Fig2: An example of refutation proof

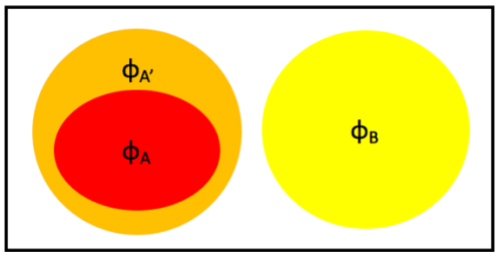

Fig3: The Boolean space of φ_A, φ_B and $\varphi_{A'}$

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Fig4: The truth table of majority gate, f=Maj(a, b, c)

3. PROPOSED METHOD

For the following, we deal with completely specified functions. Since the method of incompletely specified function is very similar to completely specified functions. Besides, we use UNSAT to represent unsatisfiable in the remain sections.

3.1 Decomposition with known variable partition

Given a function f(X) and a non-trivial variable partition X = {$X_A|X_B|X_C$ }, where X_A,X_B and X_C are disjoint. We want to check whether f(X) can be expressed as Maj($f_A(X_A)$, $f_B(X_B)$, $f_C(X_C)$) for some functions f_A, f_B and f_C. The following part describes the foundation of majority gate decomposition. Note that function f is true if and only if more than two inputs are true, which means $f_Af_Bf_C$=011, 101, 110 and 111. Function f is false if and only if more than two inputs are false, which means $f_Af_Bf_C$=000, 001, 010 and 100.

If f(X) is decomposable, then the onset and offset of f_A can be expressed as

$$\text{Onset: } f(X_A, X_B, X_C) \wedge \neg f(X_{A2}, X_B, X_C)$$
$$\text{Offset: } \neg f(X_A, X_{B2}, X_{C2}) \wedge f(X_{A3}, X_{B2}, X_{C2})$$

X_A is the common variable of onset and offset. The second clause set of onset implies $f_B(X_B)$=0 or $f_C(X_C)$=0. Therefore, if the first clause set of onset is satisfied, $f_A(X_A)$ must be 1. In addition, the second clause set of offset implies $f_B(X_{B2})$=1 or $f_C(X_{C2})$=1. Therefore, if the first clause set of offset is satisfied, then $f_A(X_A)$ must be 0. As a result, the conjunction of onset and offset should be UNSAT. If it is not UNSAT, then the decomposition under this partition is failed. After computing the interpolant, we get the function of f_A.

Next, we compute the function of f_B. The method is similar to f_A. When computing the interpolant for f_A, however, we assign value to some don't cares. Therefore, the formula of f_B depends on f_A. It can be expressed as

$$\text{Onset: } f(X_A, X_B, X_C) \wedge [\neg f(X_A, X_{B2}, X_C) \vee \neg f_A(X_A)]$$
$$\text{Offset: } \neg f(X_{A2}, X_B, X_{C2}) \wedge [f(X_{A2}, X_{B3}, X_{C2}) \vee f_A(X_{A2})]$$

X_B is the common variable of onset and offset. The logic is similar to the last part but we add a disjunction in the second clause set for both onset and offset. For the onset, if $f_A(X_A)$ =0, then $f_B(X_B)$ must be 1 in order to satisfy the first clause set. For the offset, if $f_A(X_{A2})$ =1, then $f_B(X_B)$ must be 0 in order to satisfy the first clause set. Computing the interpolant, we get the formula of f_B.

Finally, we compute the last input f_C. The onset and offset of f_C can be expressed as

$$\text{Onset: } f(X_A, X_B, X_C) \wedge [\neg f_A(X_A) \vee \neg f_B(X_B)]$$
$$\text{Offset: } \neg f(X_{A2}, X_{B2}, X_C) \wedge [f_A(X_{A2}) \vee f_B(X_{B2})]$$

X_C is the common variable of onset and offset. For the onset, if either $f_A(X_A)$ =0 or $f_B(X_B)$ =0, then $f_C(X_C)$ must be 1 in order to satisfy the first clause set. For the offset, if either $f_A(X_{A2})$ =1 or $f_B(X_{B2})$ =1, then $f_C(X_C)$ must be 0 in order to satisfy the first clause set. Computing the interpolant, we get the formula of f_C. The decomposition is finished.

3.2 Decomposition with unknown variable partition

The previous construction assumes that a variable partition X = {$X_A|X_B|X_C$ } is given. However, manually partitioning the variables is extremely time consuming. To speed up the process, we further automate variable partition in the derivation of f_A, f_B and f_C as follows. For each variable $x_i \in X$, we introduce two control variables α_i and β_i.

Adding them into the onset and offset of f_A, then the formula become as following.

$$\text{Onset: } f(X) \wedge \neg f(X_2) \bigwedge_i ((x_i \equiv x_{2i}) \vee \beta_i)$$
$$\text{Offset: } \neg f(X_3) \wedge f(X_4) \bigwedge_i ((x_i \equiv x_{3i}) \vee \alpha_i) \bigwedge_i ((x_{3i} \equiv x_{4i}) \vee \beta_i)$$

In the above clause sets, α_i =0 means $x_i \in X_A$, β_i =0 means $x_i \in X_B \cup X_C$. Note that $\alpha_i\beta_i$ can't be 00. If $\alpha_i\beta_i$=11, then x_i belongs to either X_A or ($X_B \cup X_C$).

In SAT solving, we solve the conjunction of onset and offset, and making unit assumptions on the control variables. Under an UNSAT unit assumption, the SAT solver will return a final conflict clause composed of the control variables only. Moreover, all literals in the final conflict clause are in positive phase since the conflict occurs from a subset of the control variables set to 0. This fact indicates that setting the control variables in the conflict clause to 0 is sufficient making the whole formula UNSAT. Therefore, setting the control variables absent from the final conflict clause to 1 don't affect the UNSAT result. [1]

After SAT solving, we partition the variables into two groups but we still need another computation to further partition $X_B \cup X_C$. As a result, we modify the onset and offset of f_B as following.

$$\text{Onset: } f(X_A, X\text{-}X_A) \wedge [\neg f(X_A, X_3\text{-}X_A) \vee \neg f_A(X_A)] \bigwedge_i ((x_i \equiv x_{3i}) \vee \beta_i)$$
$$\text{Offset: } \neg f(X_{A2}, X_2\text{-}X_{A2}) \wedge [f(X_{A2}, X_4\text{-}X_{A2}) \vee f_A(X_{A2})]$$
$$\bigwedge ((x_i \equiv x_{2i}) \vee \alpha_i) \bigwedge ((x_{2i} \equiv x_{4i}) \vee \beta_i)$$

In this formula, α_i =0 means $x_i \in X_B$, β_i =0 means $x_i \in X_C$. Note that $\alpha_i\beta_i$ can't be 00. If $\alpha_i\beta_i$=11, then x_i belongs to either X_B to X_C. After SAT solving again, the automated partition is done. However, this method requires two phases. Instead of solving

SAT two times, we want to partition the variables into three groups by one computation only. Hence, we use another formula describing the majority gate, a different solution is proposed.

If f(X) is decomposable, then the quantified Boolean formula

$$f(X) \wedge \exists X_A.\neg f(X) \wedge \exists X_B.\neg f(X) \wedge \exists X_C.\neg f(X)$$

is UNSAT.

This quantified formula can be rewritten as

$$f(X_A, X_B, X_C) \wedge \neg f(X_{A2}, X_B, X_C) \wedge \neg f(X_A, X_{B2}, X_C) \wedge \neg f(X_A, X_B, X_{C2})$$

This formula consists of four clause sets.

The second clause set implies $(\neg f_B(X_B) \vee \neg f_C(X_C))$, the third clause set implies $(\neg f_A(X_A) \vee \neg f_C(X_C))$, and the fourth clause set implies $(\neg f_A(X_A) \vee \neg f_B(X_B))$. The conjunction of these three implications is

$$(\neg f_A(X_A) \wedge \neg f_B(X_B)) \vee (\neg f_B(X_B) \wedge \neg f_C(X_C)) \vee (\neg f_A(X_A) \wedge \neg f_C(X_C))$$

It indicates that more than two inputs of a majority gate are 0, so the output must be 0. But the first clause set $f(X_A, X_B, X_C)$ indicates the output be 1. Thus, this formula is UNSAT if function f is decomposable.

We modify this formula as following by adding three control variables into the last three clause sets.

$$f(X) \wedge \neg f(X_2) \bigwedge_i ((x_i \equiv x_{2i}) \vee \alpha_i) \wedge \neg f(X_3) \bigwedge_i ((x_i \equiv x_{3i}) \vee \beta_i) \wedge \neg f(X_4) \bigwedge_i ((x_i \equiv x_{4i}) \vee \gamma_i)$$

In this formula, $\alpha_i\beta_i=00$ means $x_i \in X_C$, $\alpha_i\gamma_i=00$ means $x_i \in X_B$, $\beta_i\gamma_i=00$ means $x_i \in X_A$. Note that $\alpha_i\beta_i\gamma_i$ can't be 000. Only $\alpha_i=0$ means x_i belongs to either X_B or X_C. Only $\beta_i=0$ means x_i belongs to either X_A or X_C. Only $\gamma_i=0$ means x_i belongs to either X_A or X_B.

However, there is a trivial partition where $X_A=X$ and $X_B=X_C=\varnothing$. Under this partition, $f_A(X_A)=f(X)$, f_B=constant 1 and f_C=constant 0. In order to avoid this situation, we need to specify three distinct variables into X_A, X_B and X_C, respectively. That is, we choose three variables x_i, x_j and x_k. Assign $\alpha_i\beta_i\gamma_i$ to 100, $\alpha_j\beta_j\gamma_j$ to 010 and $\alpha_k\beta_k\gamma_k$ to 001 initially. These three variables are called seed.

How the automated partition works with control variables? First, we choose the seed variables, assign their corresponding $\alpha\beta\gamma$ to the needed values by unit assumptions. Next, we assign all $\alpha\beta\gamma$ of other variables to 000 by unit assumptions. Since SAT solver tends to return the final conflict clause as small as possible, the literals in conflict clause are the reason of UNSAT result. Observing the literals in conflict clause, we get the variables partition. Note that if final conflict clause contains all three literals α_i, β_i, and γ_i for some x_i, then this partition is failed.

3.3 Decomposition algorithm

MAJ Decomposition

1. While (true)
2. Choose seed variables
3. Automated partition
4. If partition succeed,
5. Go to 8.
6. If all seed combinations are tried
7. Return not decomposable.
8. Build the onset and offset to obtain f_A, f_B and f_C.
9. Return decomposable.

Above is our decomposition algorithm, we use the second method proposed in 3.2 to automated partition. If we can't find a suitable partition under this seed combination, then we try another seed combination. There are C(n,3) seed combinations totally, where n is equal to |X|, the number of the support variables of given function f. If all seed combinations can't result in a successful partition, then function f is not decomposable. If the partition succeeds, then we apply the method proposed in 3.1 to obtain f_A, f_B, and f_C.

4. EXPERIMENTAL RESULTS

The algorithms were implemented in C++ based on ABC [9] with MiniSAT [7] as the SAT solver. All experiments were conducted on a Linux machine with Intel® Core™ i7-7500U 2.7GHz CPU and 12Gb RAM.

			MAJ3			OR2	XOR2
	#in	#out	#dec	#dec/#out	time	#dec	#dec
C17	5	2	2	100%	0.035s	-	-
C432	36	7	3	43%	1.049s	7	0
C499	41	32	0	0%	56.11s	-	-
C880	60	26	14	54%	39.88s	16	11
C1355	41	32	0	0%	70.44s	0	-
C2670	233	140	134	96%	0.322s	-	-
C3540	50	22	4	18%	119.7s	-	-

#in: number of PIs; #out: number of POs; #dec: number of decomposable POs

Fig5: Experiment result

Our experiment result of majority gate decomposition is presented in Fig5. Besides, we obtained the experiment results of OR2 and XOR2

decomposition from [1]. Since we only have three test cases which are same with [1], the comparison is limited. For C432 and C880, it looks like MAJ3 decomposition is harder than OR2 and easier than XOR2 according to the number of decomposable POs. This speculation isn't certainly right, however, since a variable partition that is not disjoint is regarded as successful decomposition in [1], while we consider only disjoint partition to be success. Therefore, it is extremely hard to compare our result with [1]. Thus, we focus on MAJ3 decomposition. Observing the table, we can find that C499 and C1355 both have zero decomposable PO. Among all test cases, there are merely a few POs can be decomposed by MAJ3 under disjoint partition. The proportion of decomposable PO number to total PO number is less than 50% for most circuits. There are three counter examples, which are C17, C880 and C2670. C17 is a simple circuit consisting of merely two POs. We look at C880 and C2670, the proportion of the former is slightly better than 50% while that of the latter is up to 96%. It is interesting that most of the POs of C2670 are decomposable. This fact may be owing to the circuit structure.

5. CONCLUSIONS

In this paper, we presented a method to achieve majority gate decomposition of a Boolean function. Interpolation played an important role in the computation of the sub-functions f_A, f_B and f_C. In addition, we automated the variable partition process. As a result, the majority gate decomposition can be applied to the circuit with large size. Experiment results showed that it is hard to decompose a function by a majority gate. Only few primary outputs are decomposable for most cases. Despite this, there is still a circuit contains 96% decomposable primary outputs. For the future work, we can try other special logic gate to decompose the given Boolean function or explore what the factors of circuit that affect the decomposition are. Moreover, we should add some circuits with larger size to our test cases and the efficiency of our partition algorithm can be tested by this way.

6. REFERENCES

[1] R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung, "Bi-decom posing large Boolean functions via interpolation and satisfiability solving," in Design Automation Conference, 2008, pp. 636–641.

[2] K. Walus, G. Schulhof, G. A. Jullien, R. Zhang, and W. Wang, "Circuit design based on majority gates for applications with quantum-dot cellular automata," in Conf. Rec. 38th Asilomar Conf. Signals, Systems and Computers 2004, vol. 2, pp. 1354–1357.

[3] P. Wohl, J. A. Waicukauski, "Improving test generation by use of majority gates," in VLSI Test Symposium (VTS), 2013 IEEE 31st.

[4] L. G. Amar`u, P.-E. Gaillardon, and G. De Micheli, "Majority inverter graph: A novel data-structure and algorithms for efficient logic optimization," in DAC, pages 194:1–194:6, 2014.

[5] Dayane Alfenas Reis, Caio Araújo T. Campos, Thiago Rodrigues B. S. Soares, Omar Paranaiba V. Neto, Frank Sill Torres, "A Methodology for Standard Cell Design for QCA," in IEEE 2016.

[6] J. A. Robinson, "A machine-oriented logic based on the resolution principle," Journal of the ACM, 12(1):23– 41, 1965.

[7] N. Een and N. Soensson, "An extensible SAT-solver," in Proc. SAT, pages 502–518, 2003.

[8] W. Craig, "Linear reasoning: A new form of the Herbrand-Gentzen theorem," in J. Symbolic Logic, 22(3):250–268, 1957.

[9] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/~alanmi/abc/.