HART: A HARDWARE FOR LOGIC MINIMIZATION AND VERIFICATION

Tsutomu SASAO

Reprinted from IEEE Proceedings of the
INTERNATIONAL CONFERENCE ON COMPUTER DESIGN:
VLSI IN COMPUTER
October 7 - 10, 1985 Port Chester, NY

HART: A Hardware for Logic Minimization and Verification

Tsutomu SASAO

Department of Electronic Engineering
Osaka University
Suita 565, Japan

Abstract: An experimental logic minimization system using a hardware tautology checker (HART) have been developed, and successfully minimized large PLA's. In the proposed system, the minimization algorithm (TMINI) is similar to that of MINI, but no complement is generated. Instead, a HART is used to check the implication relation quickly. Generation of prime implicants, detection of all the essential prime implicants, detection of the redundant implicants, and verification of the correctness of the minimization are performed efficiently by using the HART. The system consists of PC-9800E (a personal computer utilizing a 8086 micro-processor with 256 kilo bytes memory), and a HART for n=8 which is composed by 43 field programmable logic arrays and some interface TTL IC's.

I. Introduction

Logic minimization of expressions with many variables is necessary not only for programmable logic array (PLA) design[1], but also for automatic logic synthesis of random logic networks[2]. Practical heuristic minimization algorithms for large scale problems can be roughly divided into two classes.

The first type of the logic minimization algorithms include PRESTO[3], POP[4] and TAU[5]. They generate prime implicants by tautology checking. Because the tautology checking by software is quite time-consuming, these algorithms require much computation time. For this reason, no iterative improvement is attempt in these algorithms. The quality of the solution of these algorithms are sometimes far from minimum[6], and so the minimized results are unreliable.

The second type of the logic minimization algorithms include MINI[7], MINI-II[8], and ESPRESSO-II [6]. They are faster then the first types of algorithms for medium size problems. These algorithms first generate a complement of a given function, and then generate prime (or near prime) implicants. The complement is effectively used to generate the prime implicants. Because the generation of prime implicants is less time-consuming than the first type of the algorithms, iterative improvement method is used to generate near optimum solutions. However, for some class of functions, the size of the complement increase exponentially with the number of inputs[9]. In such a case, the first step of the complementation cannot be completed due to the memory overflow or the computation time over.

In the proposed system, the minimization algorithm (TMINI) is similar to that of MINI-II, but no complement is generated. Instead, a hardware tautology checker (HART) is used to check the implication relation ($c \prec \mathcal{F}$ or $c \not\prec \mathcal{F}$) quickly. Generation of prime implicants, detection of all the essential prime implicants, detection of the redundant implicants, and the verification of the correctness of the minimization are performed efficiently by using a HART.

The most time-consuming part in the minimization algorithm is a implication relation checking (i.e., to decide $c \prec \mathcal{F}$ or $c \not\prec \mathcal{F}$). In the proposed minimization system, it is converted to a tautology problem (i.e., to decide $\mathcal{F} \equiv 1$ or $\mathcal{F} \not\equiv 1$). By divide and conquer method, the tautology problem is repeatedly decomposed into ones with fewer variables until the size of the sub-problems become the specified value (n=8 for the experimental system). Then the sub-problems are sent to the HART through the I/O port of the host processor.

The HART for n-variable function consists of 2^n copies of n-input AND gates, 2^n copies of latches and a 2^n-input AND gate. It makes a truth table of the given function, and checks if it is tautology ($\mathcal{F} \equiv 1$) or not ($\mathcal{F} \not\equiv 1$) by a logic circuit. In the experimental system, the HART for n=8 is composed by 43 FPLA's (Field Programmable Logic Arrays) and some interface TTL IC's.

Because the proposed algorithm (TMINI) uses similar heuristic as that of MINI, it obtains better solution than PRESTO, POP, and TAU. Because TMINI needs no complement of the function, it can minimize large PLAs for which the simple application of MINI or ESPRESSO fails, on a personal computer with small memory storage.

II. Minimization of Logical Expression

In order to explain the idea of the minimization method by using a tautology checker as simply as possible, the given function is supposed to be a single-output two-valued input function. In addition, we use the following simple minimization algorithm to show the idea. However, the real system treats multiple-valued input binary function, and uses a more complicated algorithm to produce a better solution[10].

Algorithm 2.1:(Minimization of logical Expression)

S1. Expand each cube into prime implicant.
S2. Delete redundant cubes.

In the proposed system, we use Theorem 2.1 for S1, and Theorem 2.2 for S2.

Theorem 2.1: (Prime Implicants)

Let $\mathcal{F}=c \vee \mathcal{G}$, where c is a product and $\mathcal{G}$ is sum of the other products in $\mathcal{F}$. Suppose that c can be written as $c=x_i^* e$, where e is a product and x_i^*

denotes either x_i or $\bar{x}_i$. Let $c(i)=\bar{x}_i^*e$.

1) If $c(i) \subset \mathcal{G}$, then the literal x_i^* can be removed from c, i.e., $F \equiv e \vee \mathcal{G}$.
2) If $c(i) \not\subset \mathcal{G}$ for all possible i, then c is a prime implicant of $\mathcal{F}$.

Theorem 2.2:
(Irredundant sum-of-products expression)
Suppose that an expression $\mathcal{F}=c_1 \vee c_2 \vee \ldots \vee c_m$ is given where, c_i(i=1,2,...,m) is a prime implicant of $\mathcal{F}$. Let $\mathcal{F}=c_i \vee \mathcal{G}_i$, where $\mathcal{G}_i$ is a sum of the prime implicants other than c_i.

1) If $c_i \subset \mathcal{G}_i$, then c_i can be deleted from $\mathcal{F}$, i.e., $\mathcal{F} \equiv \mathcal{G}_i$.
2) If $c_i \not\subset \mathcal{G}_i$ for i=1,2,...,m, then $\mathcal{F}$ is irredundant sum-of-products expression (minimal).

Note that in applying Theorems 2.1 and 2.2, we have to check the implication relation ($c \subset \mathcal{F}$ or not) many times. As shown in the next section, implication relation checking is quite time-consuming. This implication relation can be converted into a corresponding tautology problem, and can be solved by a HART.

In Algorithm 2.1 , the order of the expansion directions (in step S1), and the order of the elimination of the redundant prime implicants (in step S2) greatly influence the quality of the solution. So, the real algorithm (TMINI) use several heuristics to choose near optimal ordering.

III. Software Tautology Checker

In this section, we show that the implication relation can be examined by a tautology checker.

Definition 3.1: Let $\mathcal{F}$ be a sum-of-products expression. If $\mathcal{F} \equiv 1$, i.e., $\mathcal{F}$ is equal to 1 for all the input combination, then $\mathcal{F}$ is said to be tautology. The problem to decide whether a given sum-of-products expression is the tautology or not is said to be a tautology problem.

The following theorem shows that there is virtually no hope to solve the tautology problem in a polynomial time[16].
Theorem 3.1:The tautology problem is co-NP complete.

Next, we introduce a restriction operation which converts an implication question into a tautology question.

Definition 3.2: Let $c=x_1^{S_1}x_2^{S_2}\ldots x_n^{S_n}$ be a cube.
A cube restriction of $\mathcal{F}$ to c is obtained as follows and denoted by $\mathcal{F}(|c)$.
1) Compute the Boolean intersection of $\mathcal{F}$ and c, and delete null products.
2) Replace each product $x_1^{T_1}x_2^{T_2}\ldots x_n^{T_n}$ in the expression obtained in 1) with $x_1^{(T_1 \cup \bar{S}_1)}x_2^{(T_2 \cup \bar{S}_2)}\ldots x_n^{(T_n \cup \bar{S}_n)}$.

For a product term of a switching function, $x_i=x_i^1$, $\bar{x}_i=x_i^0$, and a missing variable (or don't care) is denoted by $x_i^{(0,1)}=1$.
Theorem 3.2: Let c be a cube and $\mathcal{F}$ be an expression. Then, $c \subset \mathcal{F} \leftrightarrow \mathcal{F}(|c) \equiv 1$.

The above theorem shows that the implication relation can be examined by a tautology checker. When the tautology problem is small enough, it can be directly checked by a HART. However, when the problem is large and cannot be solved directly by a HART, we have to reduce and decompose the problem into smaller ones until a HART can check them. [10]

IV. Hardware Tautology Checker(HART)

Fig.4.1 shows the HART for three-variable switching function. It consists of a minterm generator part, a latch part, and an AND gate part[11].

Minterm Generator Part has n-input and 2^n-output. Each output of the minterm generator corresponds to a minterm of the function. When a cube c is applied to the input to the minterm generator part, all the outputs which correspond to the minterms of c become one.

Latch Part consists of n-latches. Every Latch is reset to zero at the initial state. When a cube is applied to the minterm generator, all the latches which correspond to the minterms of c will be set to one. We have to apply all the cubes sequentially.

AND gate Part has an 2^n-input AND gate. The output becomes one when all the inputs are one, which shows that the given function is tautology.

The operation of the HART is illustrated in the following example.

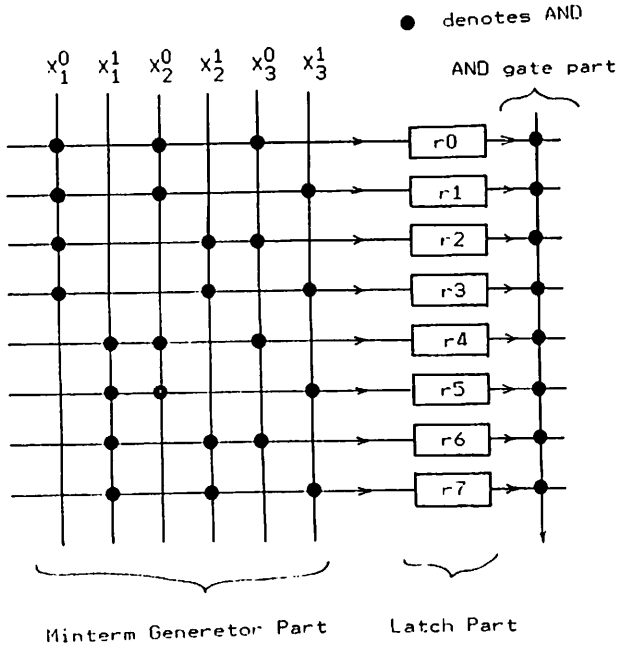

Fig. 4.1 Hardware Tautology Checker for Three-Variable Switching Function

Example 4.1: Suppose that we have to check whether

$\mathcal{F}=x_1\bar{x}_3 \vee x_2x_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2 \vee \bar{x}_1x_2\bar{x}_3$

is tautology or not. First, convert the expression into positional cube as follows:

Positional Cube						Product term
x_1^0	x_1^1	x_2^0	x_2^1	x_3^0	x_3^1	
0	1	1	1	1	0	$c_1=x_1\ 1\ \bar{x}_3$
1	1	0	1	0	1	$c_2=1\ x_2\ x_3$
0	1	1	0	0	1	$c_3=x_1\ \bar{x}_2\ \bar{x}_3$
1	0	1	0	1	1	$c_4=\bar{x}_1\ \bar{x}_2\ 1$
1	0	0	1	1	0	$c_5=\bar{x}_1\ x_2\ \bar{x}_3$

Note that x is denoted by 01, $\bar{x}$ is denoted by 10, and missing variable (don't care) is denoted by 11.

At the initial state, all the latches are reset to zero.
When c_1 is applied, r4 and r6 are set to one.
When c_2 is applied, r3 and r7 are set to one.
When c_3 is applied, r5 is set to one.
When c_4 is applied, r0 and r1 are set to one.
When c_5 is applied, r2 is set to one.
In the following table, X marks show the latches which are set by the application of each cube.

	x_1	x_2	x_3	c_1	c_2	c_3	c_4	c_5
r0	0	0	0				X	
r1	0	0	1				X	
r2	0	1	0					X
r3	0	1	1		X			
r4	1	0	0	X				
r5	1	0	1			X		
r6	1	1	0	X				
r7	1	1	1		X			

When all the cubes are applied to Fig.4.1, the output of the AND gate will be one, which shows that $\mathcal{F}$ is tautology. (End of example).

Table 4.1 Number of 2-input AND gates to realize a Tautology checker

Number of Inputs n	6	8	10	12	14
Minterm Generator	88	304	1120	4272	16712
Latch part	128	512	2048	8192	32768
ANDgate part	63	255	1023	4095	16383
Total	279	1071	4191	16559	65863

Table 4.1 shows the number of two-input gates to realize a HART[10].
In this experiment, a HART for n=3 (Fig.4.1) is realized by a FPLA(82S153 compatible) and constitutes a latch module. The HART for n=8 consists of 32 latch modules and other FPLA's for the AND modules, which are used for choosing latch modules and the control module which is used for dynamically changing the configulation of the HART to check either an 8-input 1-output function or a 6-input 4-output functions. In the experimental system, there are also the mask modules which are used to make HART to check implication relation directly (which are not used in the experiments).

V. Description of the Algorithm (TMINI)

In this section, we use the terminology of MINI[7],[8], and that of tautology checker for multiple-valued input binary function[10].

TMINI (Minimization of Logical Expression)

M0. Let $\mathcal{F}$ be a care specification, and DC be a don't care specification.
M1. (Expand) Expand each cube of $\mathcal{F}$ against the other cubes in $\mathcal{F} \vee DC$.
M2. (Delete) Delete the redundant cubes in $\mathcal{F}$.
M3. (Essential Prime Implicant Detection) Let ESS be the set of the essential prime implicants in $\mathcal{F}$. Let $\mathcal{F}1$ be the set of the other implicants in $\mathcal{F}$.
M4. If $\mathcal{F}1=\phi$ then go to M10.
M5. (Reduce) Reduce each cube of $\mathcal{F}1$ against the other cubes in $\mathcal{F}1 \vee ESS \vee DC$.
M6. (Reshape) Reshape $\mathcal{F}1$.
M7. (Expand) Expand each cube of $\mathcal{F}1$ against the other cubes in $\mathcal{F}1 \vee ESS \vee DC$.
M8. If the size of the new solution is smaller than the size of the solution immediately before the last execution of M5, then go to M5.
M9. (Input Fat) Expand the input parts of the each cube in $\mathcal{F}1$.
M10. Let $\mathcal{F}$ be $ESS \vee \mathcal{F}1$.
M11. (Output Slim) Reduce the output parts of each cube in $\mathcal{F}$.
M12. (Input Fat) Expand the input part of the cubes which are changed by M11.
M13. (Verification) Check if the obtained cover $\mathcal{F}$ is equivalent to the original specification.

Delete: For each cube c in $\mathcal{F}$, do the followings:
If $c \subset ((\mathcal{F}-c) \cup DC)$ then delete c from $\mathcal{F}$.
Expand: The heuristic of the expansion is the same as that of MINI, but the over-expanded cubes and the expanded cubes are generated by using the tautology checking algorithm[10].

Reduce: The same routine as MINI2[8],[12] is used. It is also possible to use tautology algorithm.

Reshape: Exactly same as that of MINI.

Essential Prime Implicant Detection

E1. For each cube c in $\mathcal{F}$ do the following.
E2. Expand c into prime implicant.
E3. Partition the cubes in $\mathcal{F}-(c)$ into the following three arrays:
$\mathcal{F}0$: Set of the cubes which have common element with c.
$\mathcal{F}1$: Set of the cubes which are distance one from c.
$\mathcal{F}2$: Set of the other cubes in F.
E4. Let $\mathcal{G}$ be $(\mathcal{F}0 \circledast c) \vee \mathcal{F}1$.
Let $\mathcal{H}$ be cons(c, $\mathcal{G}$).
E5. If $c \subset \mathcal{H}$, then c is non-essential.

Verification

Check if $\mathcal{F}1$ is equivalent to $\mathcal{F}2$.
V1. Let $\mathcal{F}1$ be the original array, $\mathcal{F}2$ be the minimized array, and DC be the don't care array.
V2. For each cube c in $\mathcal{F}1$, do the following: if $c \subset (\mathcal{F}2 \vee DC)$ then go to V5.
V3. For each cube c in $\mathcal{F}2$, do the following: if $c \subset (\mathcal{F}1 \vee DC)$ then go to V5.
V4. $\mathcal{F}1$ is equivalent to $\mathcal{F}2$. Stop.
V5. $\mathcal{F}1$ is not equivalent to $\mathcal{F}2$. Stop.

Tautology Checking

T1. Let $\mathcal{F}$ be an expression represented by cubical notation. This routine check if $\mathcal{F} \equiv 1$ ($\mathcal{F}$ is tautology) or $\mathcal{F} \not\equiv 1$ ($\mathcal{F}$ is non-tautology).

T2. If ((the number of the parts $\leq$ 8) and (the number of bits in all the parts=2)) or ((the number of the input parts $\leq$ 6) and (the number of the bits in the output part $\leq$ 4)) then use a hardware tautology checker. When n $\geq$ 10, go to T7 or T8 depending on the shape of expression[12].

T3. (Reduce the Tautology Problem) Apply the following operations repeatedly.
1) Delete a cube which has a part with all 0's.
2) Delete a column with all 1's.

T4. If $\mathcal{F}$ is reduced, then go to T2.

T5. If $\mathcal{F}$ has a column with all 0's, then $\mathcal{F}$ is non-tautology and return.

T6. Compute the volume of $\mathcal{F}$. If the volume of $\mathcal{F}$ is less than the volume of the universal space for $\mathcal{F}$ then $\mathcal{F}$ is non-tautology and return.

T7. (Split-by-variable Decomposition) If the j-th part of $\mathcal{G}$ has more than 2 bits, then decompose $\mathcal{F}$ into $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k$ so that the j-th part of $\mathcal{F}_i$ (i=1,2,...,k) has at most 2 bits, where $\mathcal{F}_i = \mathcal{F}(\mid X_j^{S_i})$. Apply this algorithm to $\mathcal{F}_i$. If $\mathcal{F}_i \not\equiv 1$ for some i, then $\mathcal{F}$ is non-tautology and return. Otherwise, $\mathcal{F}$ is tautology and return.

T8. (Split-by-term Decomposition) Suppose that $\mathcal{F}$ can be written as $\mathcal{F} = \mathcal{G} \vee c$, where $c = X_1^{S_1} X_2^{S_2} \ldots X_m^{S_m}$. Decompose $\mathcal{F}$ into $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$, where $\mathcal{F}_i = \mathcal{G}(\mid c_i)$, $c_1 = X_1^{S_1}$, $c_2 = X_1^{S_1} X_2^{\bar{S}_2}$, $c_3 = X_1^{S_1} X_2^{S_2} X_3^{\bar{S}_3}$, ... and $c_m = X_1^{S_1} X_2^{S_2} \ldots X_m^{\bar{S}_m}$. Apply this algorithm to $\mathcal{F}_i$. If $\mathcal{F}_i \not\equiv 1$ for some i, then $\mathcal{F}$ is non tautology and return. Otherwise, $\mathcal{F}$ is tautology and return.

VI. Experimental Results

6.1 HART versus Software

Computation time for the tautology checker with and without HART are compared by using the following expressions:

$S(n) = x_1 \vee x_2 \vee \ldots \vee x_n$.

$T(n) = S(n) \vee \bar{x}_1 \bar{x}_2 \ldots \bar{x}_n$.

Table 6.1 Computation time of tautology checker with and without HART(milli-seconds)

n	without HART S(n)	T(n)	with HART S(n)	T(n)
6	116	123	27	31
7	268	277	34	38
8	639	653	40	45
9	767	792	168	184
10	932	970	334	363
11	287	1435	287	826
12	335	1986	334	1378
13	385	2632	384	2024

In Table 6.1, the computation time with HART includes the time for converting data and sending them into HART. This table shows that the tautology checker with HART is about 15 times faster than one without HART when n=8. For S(11), they took almost same computation time because when n $\geq$ 11, both tautology checkers changes their strategies to delete unate variables.

6.2 Effect of HART on Minimization.

Minimization time with and without HART is analyzed by using a function SPL10 (See 6.5). Table 6.2 shows the minimization process, where DELETE, EXPAND, ESSENTIAL, INPUT_FAT and VERIFICATION are accelerated by HART. INPUT_COVER, RESHAPE and REDUCE are performed by the same routine, so the computation time are equal.

Table 6.2 Minimization process of TMINI

Process	# of Cubes	Time (milli-sec) Without HART	With HART
INPUT_COVER	18	655	654
EXPAND	11	13034	7829
DELETE	11	2923	905
ESSENTIAL	0	17324	4579
REDUCE	11	7881	7882
RESHAPE	11	490	490
EXPAND	10	8892	2763
REDUCE	10	8973	8973
RESHAPE	10	461	462
EXPAND	10	8124	1895
INPUT_FAT	10	7773	1462
VERIFICATION	10	4223	1678
TOTAL		81421	40190

6.3 Minimization of various PLA's .

Table 6.3 shows number of cubes after minimization, number of used words, and cpu time of MINI2 and TMINI for various PLA's. MINI2 is an enhanced version of MINI, with essential prime implicant detection[8] and a fast recursive complementation algorithm[12].

Both MINI2 and TMINI use the same data structure. Each cube is represented by a cell: it consists of n+3 words for n-input single-output function, and n+k+3 words for n-input m-output function where k=(m-1)/16 +1: it has two pointers and one extra words for miscellaneous use. Each cover is represented by a linked list of cells. In the FORTRAN compiler we used, the size of each array must be smaller than or equal to 64k bytes. Therefore, the maximum size of the LIST we can declare is DIMENSION LIST(32767). The memory size shown in Table 6.3 denotes the maximum number of words used during minimization, which must be smaller than 32767. As shown in Table 6.3, TMINI minimized a large PLAs for which MINI2 failed due to memory overflow.

6.4 Comparison with other programs.

Table 6.4 compares the minimization quality and computation time for TMINI with ESPRESSO-II C, ESPRESSO-II APL, MINI APL, and POP C[6]. ESPRESSO's and MINI were run on IBM3081K, and POP C was run on VAX-11/780. Table 5.4 shows that TMINI produces as good solutions as ESPRESSO's and MINI, and better solutions than POP. MIN shows the minimum solution. For PLA's in this table, minimization time with HART were only a few percents faster than the time without HART. This is because the volumes of cubes in both the original and minimized covers are too small to be efficiently checked by the HART.

Table 6.3 Comparison with MINI2

	INPUT			MINI2			TMINI2			
	In	Out	Cubes	Memory (words)	Cubes	Time (sec)	Memory (words)	Cubes	Time(sec) w/o HART	Time(sec) with HART
SPL10	10	1	18	1144	10	80	884	10	81	40
SPL12	12	1	22	1590	12	158	1560	12	199	126
SYE12	12	1	126	12045	68	2736	8130	66	3072	2023
ACH24	24	1	8	memory overflow			2268	8	31	31
M110	10	4	511	7182	58	843	16240	58	467	453
SKA	12	7	77	6400	45	413	4912	46	369	304
SKB	12	8	132	9568	111	875	7152	113	808	777
OK672	17	13	672	memory overflow			16695	186	4129	3959
SEG7	10	23	40	3375	35	196	3780	35	597	499
AUG1	16	8	147	13720	54	725	6720	55	480	462
HOASI	14	8	18	3672	18	79	864	18	20	22

Table 6.4 Comparison with other programs

	Input			# of cubes					
Name	In	Out	Cubes	Ess C	ESS APL	Min APL	POP C	Tmi FOR	MIN
ADR4	8	5	255	75	75	75	75	75	75
MLP4	8	8	225	127	127	126	130	125	123*
ROT8	8	5	255	57	57	58	58	57	57
SQR6	6	12	63	50	49	49	53	49	47
SYM9	9	1	420	87	85	85	148	86	84
total				396	393	393	464	392	386

	Computation time (sec)				
Name	Ess-C 3081	Ess-A 3081	Mini-A 3081	Pop-C VAX	Tmini 8086
ADR4	3.0	9.3	52.6	114.9	379.8
MLP4	9.0	33.2	269.3	121.0	1345.0
ROT8	3.4	12.6	28.8	53.4	290.3
SQR6	2.1	11.8	56.3	20.5	263.2
SYM9	4.6	16.3	226.3	212.3	991.7
total	22.1	83.2	633.3	522.1	3270.0

6.5 Description of example functions.

The following functions are given by the truth table with minterms.

ADR n : n-bit adder, 2n-input (n+1)-output.
MLP n : n-bit multiplier, 2n-input 2n-output.
SQR n : n-bit square circuit, n-input, 2n-output.
ROT n : n-bit square root circuit, n-input ((n/2)+1)-output.
SYM n : b-bit symmetric function (n=3m).
SYM n=1 if ($m \leq \sum x_i \leq 2m$).
=0 otherwise.

The following functions are generated by a HDL translator developed by the author[19].

SPL n : n-bit symmetric function.

SPL n=$\overline{g_1 \vee g_2}$, where

$g_1(n)=x_1x_2\cdots x_n$. $g_2(n)=\bar{x}_1\bar{x}_2\cdots\bar{x}_n$.

SYE n : n-bit symmetric function.

SYE n= $\overline{g_3 \vee g_4}$, where

$g_3(n)=x_2x_3\cdots x_n \vee x_1x_3\cdots x_n \vee x_1x_2\cdots x_{n-1}$.

$g_4(n)=\bar{x}_2\bar{x}_3\cdots\bar{x}_n \vee \bar{x}_1\bar{x}_3\cdots\bar{x}_n \vee \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-1}$.

ACH n : n-bit Achilles' heel function (n=3m).
ACH n $=x_1x_2x_3 \vee x_4x_5x_6 \vee \cdots \vee x_{n-2}x_{n-1}x_n$.

The following are industrial PLA's.

M110 : Control part of computer.
SKA,SKB : Circuits for clock.
OK672:Mapping table of microprogram for a 16-bit microprocessor.
SEG7 : 7-segment display [17].
AUG1:[18].
HOASI: Hollerith-ASCII converter.

7. Conclusion

1. The tautology checker with HART for n=8 is about 15 times faster than one without HART for an expression of 8-variables.
2. Minimization using tautology checker with HART is about two times faster than one without HART for an expression of 10-variables.
3. TMINI produced as good solutions as MINI and ESPRESSO, and minimized large PLA's for which MINI2 failed due to memory overflow on a personal computer.
4. For larger PLA's with many don't cares in the input parts, TMINI with HART for more than 8-variables is promising.
5. For PLA's shown in Table 6.4, HART was virtually of no effect on computation time. This is because the volume of cubes are too small and the most computation time are spent by software.

8. Future Project.

1. Increase the size of the HART.

A HART for n=8 consists of 43 FPLA's, and each FPLA dissipates 650 mW. So total power dissipation is about 28W. A HART for n=12 using same FPLA's would dissipates about 450W, and so we need to use CMOS FPLA's to reduce the power. A HART for n=8 can also be easily realized by a LSI chip if we use gate array technology. In such a case, a HART for n=12 can be build by using 16 such LSI's.

2. HART for multiple-output function.

It is easy to change the configuration of HART dynamically. A HART with 256 latches check 8-input 1-output function, 7-input 2-output function, 6-input 4-output function, or 5-input 8-output function, etc.

3. Application to other minimization system.

It is possible to use HART to accelerate ESPRESSO's, PRESTO, POP, TAU, ESPRESSO-MV[14], or PRESTOL-II[15].

4. Hardware for test generation.

In the test generation algorithm, we have to find an input vector $\underline{a}$ such that $\mathcal{F}(\underline{a})=0$, where $\mathcal{F}$ is a sum-of-products expression. The problem to find such an input vector is NP-complete and is quite time consuming. We can make a hardware for this problem in a similar way to HART. Fig.8.1 shows a hardware for test generation. The minterm generator part, and the latch part are same as that of HART.

Zero detection part finds a latch whose output is zero, and represents its position by a binary code. If the output T is equal to 1, then the given function is tautology and there is no test. Table 8.1 shows the truth table of the PLA for this part.

Table 8.1 Zero detection part

y0	y1	y2	y3	y4	y5	y6	y7	T	Y4	Y2	Y1
0	-	-	-	-	-	-	-	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1	1
1	1	1	1	0	0	0	0	0	1	0	0
1	1	1	1	1	0	0	0	0	1	0	1
1	1	1	1	1	1	0	0	0	1	1	0
1	1	1	1	1	1	1	0	0	1	1	1
1	1	1	1	1	1	1	1	1	0	0	0

Acknowledgement

The author thanks to the engineers of KUBOTA Ltd. for their co-operation to build HART. He also thanks to Prof. G.Hachtel for his comments on the tautology checker[13]. This work was supported by Grant in Aid for Scientific Research of the Ministry of Education, Science , and Culture of Japan.

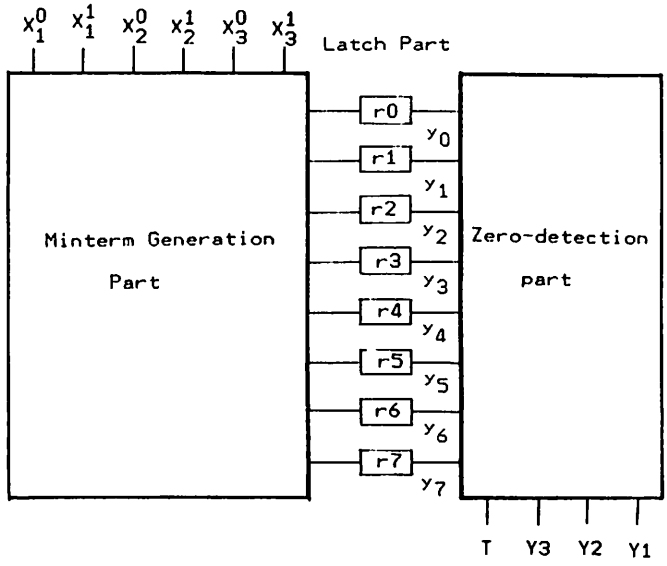

Fig.8.1 Hardware for test generation

Reference

[1] H-F.S. Law and M.Shoji,'PLA design for the BELLMAC-32A microprocessor',ICCC-82,pp.161-164, September 1982.

[2] R.K. Brayton and C. McMullen,'Synthesis and optimization of multistage logic,' ICCD-84, pp.23-28, 1984.

[3] D.W.Brown, 'A state-machine synthesizer -- SMS,' Proc. of 18-th Design Automation conference , June 1981.

[4] G. DeMicheli, M.Hofmann, R. Newton, and A. Sangiovanni-Vincentelli,'A system for the automatic synthesis of programmable logic arrays,' in Advances in Computer-Aided Engineering. A. Sangiovanni-Vincentelli editor, Jay Press, 1984.

[5] A. Poretta, M.Santomauro, and F.Somenzi,'TAU: A fast heuristic logic minimizer,' ICCAD-84, November 1984.

[6] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, 1984.

[7] S.J.Hong, R.G.Cain and D.L.Ostapko, 'MINI: A heuristic approach for logic minimization,' IBM J. of Res. and Dev., vol,18, pp.443-458, September 1974.

[8] T. Sasao, 'Input variable assignment and output phase optimization of PLA's' IEEE Trans. on Comput., vol. C-33, No.10, pp.879-894, October 1984.

[9] T.Sasao, S.J.Hong, and R.K. Brayton,'Minimization of PLA's by decomposition,' (in preparation).

[10] T.Sasao, 'Tautology checking algorithm for multi-valued input binary functions and their application,' Inter. Sympo. on Multiple-valued Logic, pp.242-250, May 1984.

[11] T.Sasao, 'A hardware for logic minimization,' (in Japanese), The 9-th Workshop on FTC, July 1983.

[12] T.Sasao, 'An algorithm to derive the complement of a binary function with multiple-valued input,' IEEE Trans. on Comput., vol. C-34, No.2, pp.131-140, Feb. 1985.

[13] G.Hachtel,'Algorithms for multi-level tautology and equivalence,'ISCAS 85,pp.1277-1280, June 1985.

[14] R.L.Rudell and A.L.M.Sangiovanni-Vincentelli 'ESPRESSO-MV: Algorithms for multiple-valued logic minimization,' Custom Integrated Circuit Conference, pp.230-234, May 1985.

[15] M.Bartholomeus and H.De Man,'PRESTOL-II: Yet another logic minimizer for programmed logic arrays', ISCAS 58, pp.447-450, June 1985.

[16] M.R.Garey and D.S.Johnson, Computers and Intractability, W.H.Freeman and Company, San Francisco, 1979.

[17] W.N Carr and J.P.Mize, MOS/LSI Design and Application, McGraw-Hill, 1972,New York p.241.Fig8.9(c).

[18] M.Auguin, F.Boeri, and C.Andre,'An algorithm for designing multiple Boolean functions: application to PLA's',Digital Process 4,3-4, p.227, 1978.

[19] T.Sasao, 'A logic synthesis and analysis system on a personal computer,'(in Japanese) The 13-th Workshop on FTC, July 1984.