Whirlpool PLAs: A Regular Logic Structure and Their Synthesis

Fan Mo and Robert K. Brayton
Department of EECS, University of California, Berkeley
{fanmo,brayton}@eecs.berkeley.edu

Abstract *¾ A regular circuit structure called a Whirlpool PLA (WPLA) is proposed. It is suitable for the implementation of finite state machines as well as combinational logic. A WPLA is logically a four-level Boolean NOR network. By arranging the four logic arrays in a cycle, a compact layout is achieved. Doppio-ESPRESSO, a four-level logic minimization algorithm is developed for WPLA synthesis. No technology mapping, placement or routing is necessary for the WPLA. Area and delay trade-off is absent, because these two goals are usually compatible in WPLA synthesis.*

1. Introduction

A conventional method of implementing FSMs is to use standard-cells and a design flow (possibly iterated) of logic synthesis, technology mapping, and physical design, such as placement and routing. In deep-sub-micron (DSM) designs, such a design flow exacerbates the timing closure problem [6]. It has become widely accepted that regular circuit and layout structures are a means of alleviating the problem [14]. Generally, regular structured circuits, such as memory [8,9] and array structures [10,13] are more predictable. Various regular structures are being explored [8-13]. Arranging cells in rows as in standard-cell designs is not enough, because routing is unpredictable. Although regular structures may aid timing closure, area and/or delay penalties (if any) should be minimized. In addition, regular structures are also favorable from the manufacturing point of view.

A new regular structure is proposed and synthesis methods for this are given. It is a cyclic four-level programmable array, called a Whirlpool Programmable Logic Array (WPLA). Since this cascaded NOR structure allows binary inputs to each plane, it extends the conventional Sum-of-Products (SOP) form. An algorithm called *Doppio-ESPRESSO* is developed to synthesize logic into WPLAs. Unlike *ESPRESSO* [2], which could be used to minimize two two-level circuits separately, *Doppio-ESPRESSO* uses the extra structural flexibility in WPLAs for further optimization. An important feature is that after logic minimization, the layout is completely determined. No technology mapping, placement or routing is needed for the WPLA; neither is prediction necessary, because area and delay are solely determined by the logic embedded in the WPLA. Another interesting feature is that area and delay minimization rarely conflict in this structure (primarily because 4-level logic is required). If the delay requirements are not satisfied, the user's only option is to modify the specification rather than re-running many optimizations with different synthesis parameters.

The WPLA structure is suitable for circuits with up to several thousand gates. Therefore it can be a building block on the chip. Block-level placement and routing are still needed to complete the interconnections between the WPLAs. To maintain global regularity, regular global interconnections are desired. Fortunately, both a block-level placer and a regular global wiring scheme have been reported [16,14]. However, the question remains of how to optimally partition the circuit into pieces that can fit the size requirements of WPLAs. In this paper, we focus on the synthesis of WPLAs.

The paper is organized as follows. In Section 2, the WPLA structure is described, and area and delay computations are given. In Section 3, a synthesis algorithm for the WPLA structure is detailed. Section 4 gives some experimental results, and Section 5 concludes.

2. The circuit structure of the WPLA

The WPLA structure is shown in Figure 1. The four programmable planes, labeled 0, 1, 2 and 3, are organized in a cycle. In each plane, input signals consist of external inputs as well as outputs from the preceding plane. Placing latches between planes 3 and 0 breaks the combinational loops. A plane is logically one level of NOR gates. Positive and/or negative (inverters) buffers are inserted between two neighboring planes; hence inputs to a NOR plane can have both polarities. WPLA circuits consume only 2 metal layers.

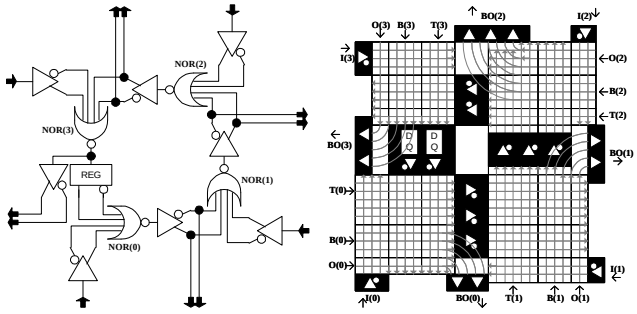


Figure 1. Schematic and layout view of a WPLA.

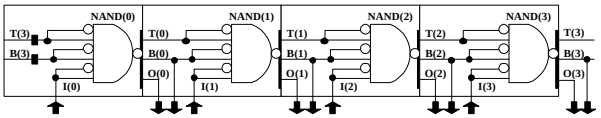


Figure 2. Abstract view of a WPLA.

Two cascaded NOR gates, together with the buffers, can be modeled as a NAND-NAND structure, shown in Figure 2, which is equivalent to a SOP. However, the inputs to the NAND gates can have both polarities available by choosing buffer polarities.

The signals of a WPLA are divided into four categories. *T*(.) denotes the set of signals used only internally by the next NAND. *B*(.) denotes the set of primary outputs that also feed the next NAND. *O*(.) are the primary outputs that do not fanout to the next NAND. *I*(.) are the primary inputs. The union of *T*(.) and *B*(.) is abbreviated by *TB*(.). Similar abbreviations include *BO*(.) and *TBO*(.).

The width and height of a plane are:

$$W(n) = |I(n)|u_I(n) + |B(n-1)|u_B(n-1) + |T(n-1)|u_T(n-1)$$

$$H(n) = |T(n)|u_T(n) + |B(n)|u_B(n) + |O(n)| + S_{BUFI}$$

where |.| is the number of the signals in the type, S_{BUFI} is the size of the input buffer, and $u(.)$ is a variable denoting the ratio of unate signals of a type. If only one polarity of a signal is used in the plane, then this signal is classified as unate. Otherwise it is binate. A unate signal occupies one line in the plane, while a binate signal occupies two. So the binate coefficient of $B(.)$ type is defined as:

$$u_B(.) = \frac{|B_{una}(.)| + 2\times[|B(.)| - |B_{una}(.)|]}{|B(.)|}$$

where $B_{una}(.)$ is the set of unate signals in $B(.)$. Similar definitions can be derived for $u_T(.)$ and $u_I(.)$. The size of the WPLA is:

$$W = \max[W(0), H(3)] + \max[H(1), W(2)] + S_{BUFM}$$
$$H = \max[W(3), H(0)] + \max[H(2), W(1)] + S_{LATCH}$$
$$Area = W \times H$$

where S_{BUFM} and S_{LATCH} are the size of the intermediate buffer and the latch, respectively. Hence the area of the WPLA is completely determined by the embedded logic.

The total delay is a summation of the delays of the four planes, assuming the last switching buffer determines the delay of its driving plane. The delay formulation of a static PLA plane follows [15], thus:

$$D = \sum_{i=0\sim3} e_N K_T \left(|I(i)|u_I(i) + |B(i-1)|u_B(i-1) + |T(i-1)|u_T(i-1)\right) + K_P\left(|O(i)| + |B(i)|u_B(i) + |T(i)|u_T(i)\right) + \max\left[\left(D_{BUFI} + d_{BUFI}L_{BUFI}(i)\right), \left(D_{BUFM} + d_{BUFM}L_{BUFM}(i)\right)\right],$$

in which, K_T, K_P are coefficients determined by the technology, D_{BUFI} and D_{BUFM} are the intrinsic (load independent) delays of the input and intermediate buffers, d_{BUFI} and d_{BUFM} are the load dependent delays of the two kinds of buffers, $L_{BUFI}(.)$ and $L_{BUFM}(.)$ are the loads of the corresponding buffers, and e_N is the density of the plane that can be derived from the bit map of the plane. The formula does not include the set-up and hold times of the latches. Although more precise delay formulations can be used, the essential point is that there are no extra elements to predict in the delay computation, given the logic implemented in the WPLA.

The delay formulation shows that reducing size can usually reduce delay, if e_N does not grow fast at the same time. This characteristic makes the design flow straightforward; synthesis algorithms only need to focus on minimizing the area since usually delay is minimized as well. This can be explained by two factors; 1) the number of logic levels[1] is fixed, and 2) uniform buffering is used in WPLAs. In a standard-cell design, collapsing nodes on a critical path can reduce the logic levels in the hope of reducing delay, essentially introducing more parallelism. However, real gates have limited drive capacities. Increasing parallelism means larger loading; hence buffers are inserted, or driving gates themselves are duplicated. Inserting buffers may introduce additional delays; duplicating gates actually shifts the load burden backwards. In addition, such timing optimization may trigger an unexpected blow-up in area. Placement and routing factors may further complicate the problem. When a standard-cell implementation does not meet delay requirements, it is difficult to decide whether further collapsing should be done and if so, how. The WPLA synthesis approach has no such scenario.

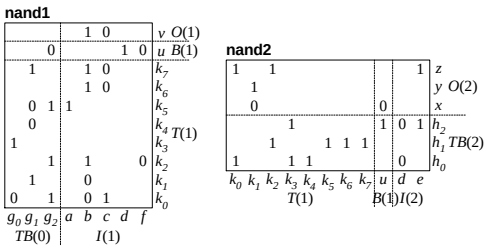
3. *Doppio-ESPRESSO*, a four-level minimization algorithm

3.1. Overview

The basic idea of WPLA synthesis is to minimize a pair of NANDs, and iterate for different pairs until no further improvement. The possible pairs in the WPLA are 0-1, 1-2 and 2-3. The minimization of a pair of NANDs differs from conventional SOP minimization since the WPLA structure allows negated products. To employ SOP minimization, we transform form the NAND-NAND to SOP form, apply a SOP minimizer and transform the result back. The *Doppio-ESPRESSO* is summarized in the following pseudo code:

```
do
        {nA,nB}=SOP2NN(ESPRESSO(NN2SOP(n0,n1)))
        if {nA,nB} better than {n0,n1}
                {n0,n1}={nA,nB}
        end if
        {nA,nB}= SOP2NN(ESPRESSO(NN2SOP(n1,n2)))
        if {nA,nB} better than {n1,n2}
                {n1,n2}={nA,nB}
        end if
        {nA,nB}= SOP2NN(ESPRESSO(NN2SOP(n2,n3)))
        if {nA,nB} better than {n2,n3}
                {n2,n3}={nA,nB}
        end if
until no improvement
```

Here "improvement" and "better" mean smaller total area. We use an example throughout the discussion of the transformation and optimization algorithms. Following are the initial NAND-NAND matrices,



called nand1 and nand2. The inputs to nand1 include $TB(0)$ and $I(1)$, and the outputs of nand1 include $T(1)$, $B(1)$ and $O(1)$. $T(1)$, $B(1)$ and $I(2)$ form the inputs to nand2, and nand2 outputs $TB(2)$ and $O(2)$. For better visualization, only the care bits are shown. The SOP form has a product matrix and a sum matrix. The vacant space in the product matrix means '-', while the vacant space in the sum matrices means '0', or "don't output". The conventions of the SOP form include:

(1) The use of negative products is forbidden.
(2) The products output to the sums only.
(3) The sums only take the products as inputs.

The transformation algorithm should generate and accept the SOP form with these restrictions. In addition, the polarities of the primary inputs and outputs can be obtained by using the appropriate input and output buffers. So if the original function outputs signal Z, it is possible to end up with an optimized function of the complement $\overline{Z}$.

3.2. *NN2SOP*, the NAND-NAND to SOP transformation

We start with the simplest case, that is, $I(2)=\Phi$, $BO(1)=\Phi$ and the nand2 matrix is positive unate. Then the transformation is simply copying the nand1 matrix to the product matrix and copying the transpose[2] of the nand2 matrix to the sum matrix. Suppose $BO(1)\neq\Phi$. This is one of the major structural differences between WPLAs and conventional PLAs. Since the SOP form can only output from the sums, the $BO(1)$ signals have to be raised to $O(2)$. Suppose Y is a $BO(1)$ signal. We can create a new $O(2)$ signal, $\overline{Y}$, which is simply the complement of Y, but now corresponds to an output of the sum. Another case is $I(2)\neq\Phi$, which means some external signals enter nand2 directly. Since only product terms can enter the sums in SOP form, the $I(2)$ signals need need to be pushed back to $I(1)$:

[1] level: A PLA is a two-level circuit, or a SOP. To prevent confusion in the description of the so-called multi-level logic minimization where PLAs stay in different levels, we use the terminology "depth" instead. So the depth of a circuit is in fact half of the number of the levels.

[2] transpose: here transpose means a 90 degree clockwise rotation followed by a mirroring of the X-axis.

$$Z = \overline{\prod_i Y_i \prod_j y_j} = \overline{\prod_i Y_i \prod_j \overline{T_j}}$$

where Z is an $O(2)$ signal, y_j's belong to $I(2)$, and $T_j = \bar{y}_j$, for all j, are the new $T(1)$ signals. They replace $I(2)$ and can be used in nand2. Finally, nand2 is not positively unate (it is always so in a SOP), which means nand2 can use both polarities of its input signals. Suppose an $O(2)$ signal Z is the NAND of a set of positive $T(1)$ signals Y_i and a set of negative $T(1)$ signals Y_j. Each Y_j can be expressed by a NAND of a set of $TB(0) \cup I(1)$ signals x_{jk}. Then replace Y_j by x_{jk}'s:

$$Z = \overline{\prod_i Y_i \prod_j \overline{Y_j}} = \overline{\prod_i Y_i \prod_j \overline{\prod x_{jk}}} = \overline{\prod_i Y_i \prod_{j,k} \prod x_{jk}} = \overline{\prod_i Y_i \prod_{j,k} x_{jk}} = \overline{\prod_i Y_i \prod_{j,k} \overline{x}_{jk}} \quad (1)$$

However as mentioned above, x_{jk}'s have to be relayed to $T(1)$; thus their polarities should be adjusted. Now the steps of the transformation to SOP form are described.

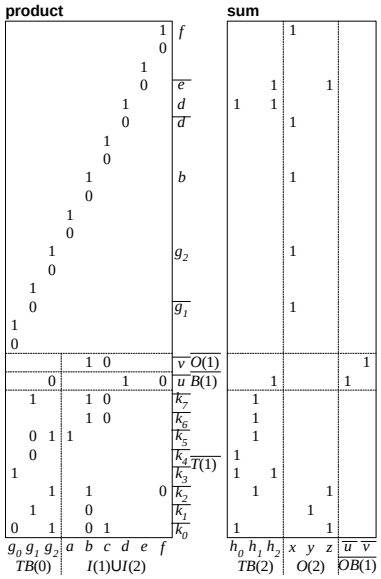
Step 1. The inputs of the product matrix include $TB(0)$ and $I(1) \cup I(2)$. The outputs of the sum matrix include $TB(2)$, $O(2)$ and $\overline{BO(1)}$ [3].

Step 2. Copy nand1 to the product matrix.

Step 3. For each input signal in the product matrix, build two rows, one with a '0' in the column of that signal, and one with a '1' in the column. By doing so, all the inputs of the product matrix, including $TB(0)$ and $I(1) \cup I(2)$, are relayed. The relayed signals appear as if they are new $T(1)$ signals that can be used by nand2. We call them pseudo-$T(1)$ signals.

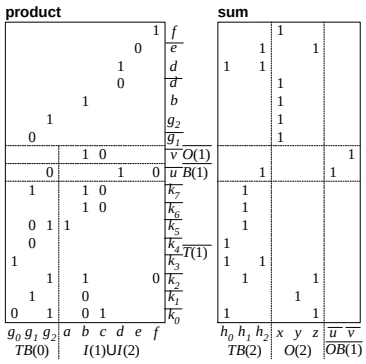
Step 4. Copy the transpose of nand2 to the sum matrix. Use the pseudo-$T(1)$ signal when $I(2)$ is required. For example, $h_0 = \overline{k_0 k_3 k_4 \overline{d}}$ where d is an $I(2)$ signal. Then the pseudo $T(1)$ signal d is used instead. Thus the 0's in the sum matrix come only from the $TB(1)$ part of nand2, $x = \overline{\overline{k_1} \overline{u}}$, for instance. Equation (1) is used to break the negative $TB(1)$ literals into a set of $TB(0) \cup I(1)$ signals, which have their pseudo-$T(1)$ versions available.

Step 5. Use a column singleton to relay a $BO(1)$ signal, such as $\overline{u}$ and $\overline{v}$. Notice that if the net is $\overline{O(1)}$, then it is also a row singleton in the sum matrix, because no $TBO(2)$ signal uses it. But $\overline{B(1)}$ must not be row singleton, because by definition some of the $TBO(2)$ use it.



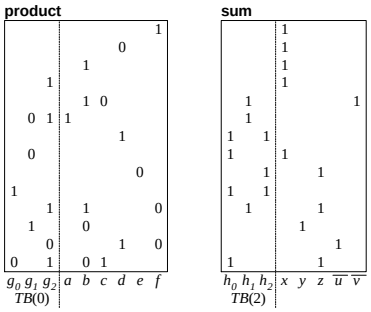
[3] $\overline{BO(1)}$ means the set of complemented BO(1) signals. Similar for other abbreviations.

It is obvious that all the pseudo-$T(1)$ signals are not always utilized. But keeping them does not affect the SOP minimization. A more succinct SOP representation, with all unused pseudo-$T(1)$ rows removed, looks like:



3.3. *ESPRESSO*, the SOP minimization algorithm

ESPRESSO is employed to perform the SOP minimization [2,5]. It makes no change to the input net list and the output net list. However, the content of the product and sum matrices may change. In the example, *ESPRESSO* gives the following optimized SOP form.

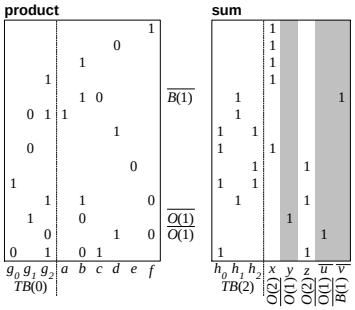


3.4. *SOP2NN*, the SOP to NAND-NAND transformation and optimization

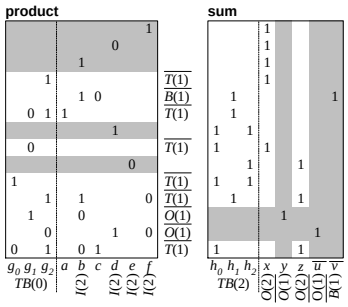
During the transformation from the minimized SOP back to a nand1-nand2 form, the original $BO(1)$ and $O(2)$ may have a new distribution between the two NANDs. The external inputs $I(1)$ and $I(2)$ may have new distributions as well. We will show that the re-distribution provides additional opportunities to optimize the logic functions. However, $TB(0)$ and $TB(2)$ are unchanged, because these signals are fixed, due to the structural restriction of the WPLA.

The algorithm consists of two parts. The first part, Steps 1 to 4, produces the nand1 and nand2 matrices from the SOP. These steps are mostly done by definition. Then the nand1-nand2 is further optimized using Steps 5 to 7.

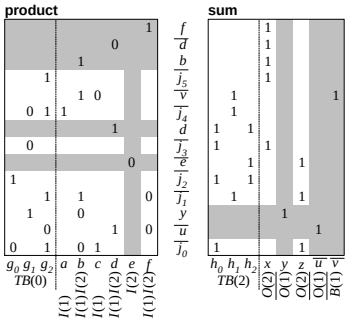
Step 1. This step distinguishes $\overline{B(1)}$, $\overline{O(1)}$ and $O(2)$. In the sum matrix, the $TB(2)$ columns are left alone. In the remaining columns, the ones with a single 1 become $\overline{BO(1)}$, because these columns correspond to relays. The others are $O(2)$. Shade the $\overline{BO(1)}$ columns. Then check the rows associated with the $\overline{BO(1)}$ column singletons. If a '1' is also a row singleton, then the associated $\overline{BO(1)}$ signal should be $\overline{O(1)}$. Otherwise it is $\overline{B(1)}$. Label the signal types of the recognized $\overline{B(1)}$ and $\overline{O(1)}$, and obtain the following SOP matrices.



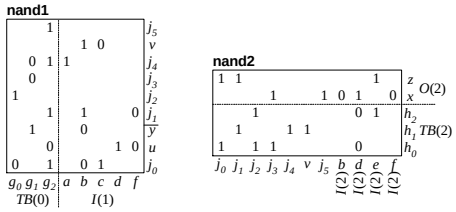
Step 2. This step recognizes $I(2)$ and $\overline{T(1)}$. In the product matrix, leave alone the $\overline{BO(1)}$ rows. Check the remaining rows. If a row has care bit(s) in $TB(0)$ columns, or it has two or more care bits in the row, then the row is associated with a $\overline{T(1)}$ signal. The remaining rows, with single '0' or '1' in the non-$TB(0)$ columns are $I(2)$. Shade these rows, and label the corresponding columns with $I(2)$.



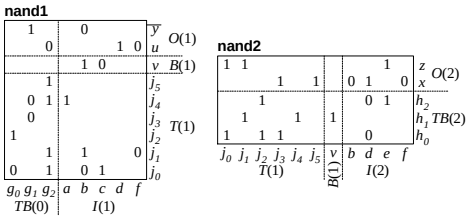
Step 3. This step identifies $I(2)$-only signals, because some $I(2)$ can also be $I(1)$, if they are used by both the nand1 and nand2. Check each column that has been identified as $I(2)$. An $I(2)$-only signal requires that each care bit appearing in the column should be the row singleton. Otherwise it is also $I(1)$. In the example, signal *e* is identified as an $I(2)$-only signal. Shade all the $I(2)$-only columns in the product matrix.



Step 4. The non-shaded region in the product matrix is copied to nand1, and the transpose of the non-shaded region in the sum matrix is copied to nand2, but the $I(2)$ care bits should be inverted. After the operation, the $TB(1)$ columns in the nand2 matrix should contain no 0's.



Re-arranging rows and columns, we get:



So far, the optimization comes only from the SOP minimization. Further optimization is possible. Suppose:

$$Z = \overline{\prod_i Y_i \prod_j T_j}$$

where, $T_j = \overline{x}_j$, is a single-literal function in the nand1 and the literal x_j is a $TB(0)$. Then all T_j's can be combined in nand1 using a new $T(1)$ signal N:

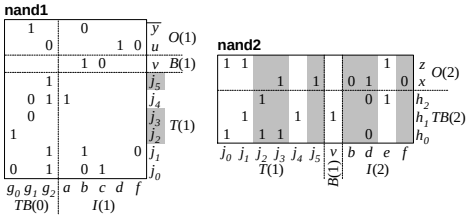
$$N = \overline{\prod_j x_j}$$

which leads to:

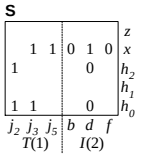
$$Z = \overline{\prod_i Y_i \overline{N}}$$

Now we have the option of removing some care bits in nand2 and replacing them with a new signal. If enough care bits are removed from the nand2 matrix by applying this transformation, then some columns might become empty and thus can be deleted. The cost is to introduce new columns for signals like N in the above formulas. We want to maximize the total reduction. The following steps implement this idea.

Step 5. First, search for qualified $TB(1)$ signals in the nand1 matrix, i.e., row singleton with the single care bit in a $TB(0)$ columns. Shade the columns in the nand2 matrix associated with the selected $TB(1)$ signals. Also in the nand2 matrix, shade the $I(2)$ columns except the $I(2)$-only columns.



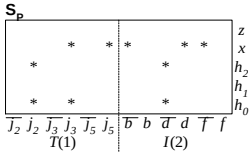
All the shaded columns in nand2 form a sub-matrix *S*.



Step 6. To maximize the reduction, we identify common patterns in the *S* matrix. To eliminate a column, all the care bits in the column should be covered by some selected pattern(s). Denote the number of $T(1)$ columns eliminated by C_T, and the number of $I(2)$ columns eliminated by C_I. Eliminating these columns will save C_T+C_I columns in nand2 and C_T rows in nand1, at the expense of creating R_P new rows in nand1 and R_P new columns in nand2. Here R_P is the number of patterns used. Define *gain* as the total reduction in size of the two matrices:

$$gain = (C_T + C_I - R_P)H_2 + (C_T - R_P)W_1$$

where H_2 is the height of the nand2 matrix, and W_1 is the width of the nand1 matrix. To simply the pattern recognition when different polarities may exist in the same signal, the *S* matrix is expressed in a pattern matrix S_P as shown below.



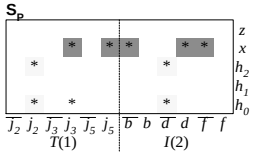
Each column in S is split into two, one for the positive literal, and one for the negative literal. Then the care bits are replaced by *'s. The algorithm has two parts. The first collects a set candidate patterns for the covering, and the second selects a subset that maximizes the gain. The first part is summarized below.

```
all the *'s are labeled "uncovered"
PATTERN=Φ
for each column c
        temp pattern p=*'s in c
        hasCommon=false
        for each column cc except c
                if p⊆cc
                        the *'s of p∩cc are labeled "covered"
                        hasCommon=true
                end if
        end for
        if hasCommon=true
                the *'s in c are labeled "covered"
                PATTERN∪=p
        end if
end for
```

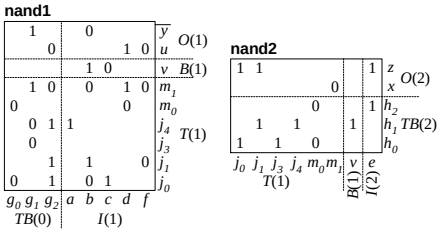
After the first step, the set *PATTERN* contains the candidate patterns. Define the size of a pattern as the product of the number of *'s in the column and the number of columns in matrix S_P that are covered by this pattern. Then a seed pattern, p_0, is chosen from the candidate set, which gives the highest gain. Notice that the highest gain provided by p_0 alone might not be positive, because $R_P = 1$, while C_T and C_I might both be 0 at this moment. If a tie occurs, choose the larger pattern. Further ties can be broken by choosing the one with the larger number of *'s in the column. The second part of the algorithm is as follows.

```
select the seed pattern p0
list[0]= p0
pattern-= p0
get gain[0]
i=1
while PATTERN≠Φ
        choose p from PATTERN that increases gain the most,
        or, if none exists, choose the one decreases gain the least.
        list[i]=p
        PATTERN–=p
        get gain[i]
        i++
end while
find the maximum gain[n]. If tie, use the first one.
if gain[n]≤0
        choose nothing.
else
        choose the first n patterns in list.
end if
```

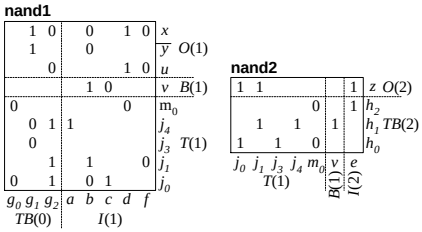
In this example, two patterns are chosen as shown below.



Remove the columns in nand2 covered by the chosen patterns, and replace their functions with new $T(1)$ signals. In the example, column j_2, j_5, d, b and f in nand2 are removed, and m_0 and m_1 are created.



Step 7. Check if an $O(2)$ signal now becomes a row singleton in the nand2 matrix. For instance, $O(2)$ signal x now has only a single care bit in the row, which means that it can be pulled back to nand1 and become a $O(1)$. When the row is saved, it might generate empty columns in the nand2 matrix, m_1 in this case. Then the m_1 column can be saved. This operation concludes the *SOP2NN* transformation and optimization algorithm; the final nand1-nand2 matrices are shown below.



Finally we give the original logic functions:

$$k_0 = \overline{\bar{g}_0 g_2 \bar{b} c}, \quad k_1 = \overline{g_1 \bar{\bar{b}}}, \quad k_2 = \overline{g_2 bf}, \quad k_3 = \bar{g}_0,$$
$$k_4 = \bar{\bar{g}}_1, \quad k_5 = \overline{\bar{g}_1 g_2 a}, \quad k_6 = \overline{b\bar{c}}, \quad k_7 = \overline{g_1 b\bar{c}},$$
$$u = \overline{\bar{g}_2 d\bar{f}}, \qquad v = \overline{b\bar{c}},$$
$$h_0 = \overline{k_0 k_3 k_4 \bar{d}} = g_2\bar{b}c + g_0 + \bar{g}_1 + d, \quad h_1 = \overline{k_2 k_5 k_6 k_7} = g_2 bf + \bar{g}_1 g_2 a + b\bar{c},$$
$$h_2 = \overline{k_3 u d \bar{e}} = g_0 + d + \bar{e}, \qquad x = \overline{\bar{k}_1 \bar{u}} = \bar{g}_1 + b + g_2 + \bar{d} + f,$$
$$y = \overline{\bar{k}_1} = g_1\bar{b}, \qquad z = \overline{k_0 k_2 e} = \bar{g}_0 g_2 \bar{b} c + g_2 b\bar{f} + \bar{e}$$

and the final logic functions:

$$j_0 = \overline{\bar{g}_0 g_2 \bar{b} c}, \quad j_1 = \overline{g_2 bf}, \quad j_3 = \bar{\bar{g}}_1,$$
$$j_4 = \overline{\bar{g}_1 g_2 a}, \quad m_0 = \overline{\bar{g}_0 \bar{\bar{d}}},$$
$$u = \overline{\bar{g}_2 d\bar{f}}, \qquad v = \overline{b\bar{c}},$$
$$h_0 = \overline{j_0 j_3 \overline{m_0}} = g_2\bar{b}c + \bar{g}_1 + g_0 + d, \quad h_1 = \overline{j_1 j_4 v} = g_2 bf + \bar{g}_1 g_2 a + b\bar{c},$$
$$h_2 = \overline{\overline{m_0} e} = g_0 + d + \bar{e}, \qquad x = \overline{g_1 \bar{g}_2 \bar{b} d \bar{f}} = \bar{g}_1 + b + g_2 + \bar{d} + f,$$
$$\bar{y} = \overline{g_1 \bar{\bar{b}}}_1, \qquad z = \overline{j_0 j_1 e} = \bar{g}_0 g_2 \bar{b} c + g_2 b\bar{f} + \bar{e}$$

Comparing u, v, h_0, h_1, h_2, x, y and z, the optimized functions are logically equivalent to the original ones, but the nand-nand size reduces from 8×10+11×6 to 8×9+7×4. *ESPRESSO* gives a nand2-nand2 with the size of 8×9+11×5. The additional improvement is due to the *SOP2NN* algorithm.

4. Experimental results

We compare the following methods of implementation: standard-cells (SCs), network of PLAs (NPLAs) [7], River PLAs (RPLAs) [13] and Whirlpool PLAs (WPLAs). An NPLA can be regarded as an intermediate representation between technology independent and technology dependent logic optimizations [4]. The RPLA is a regular structure composed of a stack of PLAs; the adjacent PLAs are connected via river routing. Logically it represents a multi-level Boolean network. In fact, a *depth*=2 NPLA or RPLA is logically similar to the WPLA, except that 1) WPLAs can have primary outputs directly from the product terms and 2) the product terms can appear in both polarities. The *depth*=1 NPLA and RPLA degrade to a single PLA. A 0.35-micron technology was used for the comparisons, since a standard-cell gate library was available for this, with over 100 gates, and each logic gate has at least two choices of drive strength. Typical parameters of the gate library are given in Table 1.

Parameter	ND2	ND2X4
logic function	2-input nand	2-input nand
area (um^2)	54	126
load limit	12	22
input pin load	1.0	1.2
intrinsic delay (ps)	170	540
load dependent delay (ps/load)	60	30

Table 1. Typical parameters of the gate library

Standard-cell implementations use over-the-cell-routing. Since the gates use metal-1 for internal connections, metal-2 and -3 are needed

for inter-gate connections. NPLAs use metal-1 and -2 for internal connections, so the NPLA needs metal-3 and -4 for inter-PLA connections. The RPLAs only need metal-1 and -2 for routing. A WPLA uses only metal-1 and -2. Some typical parameters in the PLA designs are given in Table 2.

Parameter	value
size of a programmable bit (um)	0.8x1.0
width of input/output buffer (um)	4.0
width of intermediate buffer (um)	10.0
intrinsic delay of a transistor (ps)	30
load dependent delay of a transistor (ps/loading transistor)	40
intrinsic delay of in/out buffer (ps)	250
load dependent delay of in/out buffer (ps/loading transistor)	8
intrinsic delay of intermediate buffer (ps)	150
load dependent delay of intermediate buffer (ps/loading transistor)	5

Table 2. Typical parameters in the PLA designs

Fifteen FSM examples from the LGSynth 91 benchmark set [3] were tested. After the latches are removed from each example, (we do not deal with state minimization and encoding), the combinational part is optimized with *SIS* [1] (using *script.rugged*) to achieve an initial Boolean network with depth d_0. Then for SC, NPLA and RPLA synthesis, we generate area/delay trade-off curves, by decreasing the depth gradually from d_0 using the *SIS* command "reduce_depth –d *d*". At each depth *d* [4], for SC we use *SIS* "map –n1 –AFG" command (minimum delay circuit that respects load limit) for technology mapping; for NPLAs, we cluster all single-output nodes at the same level, and call *ESPRESSO* with its default settings to minimize the clustered multiple-output PLAs. The RPLAs are synthesized with its own algorithm [13]. WPLAs have a fixed depth of 2, so it only has one solution per example (no area-delay trand-off).

In Table 3, the number of programmable bits of NPLAs, RPLAs (both *depth*=2) and WPLAs are compared. In this case, both the NPLA and RPLA are four-level structures. However due to the different algorithms used to synthesize them, their results are slightly different. The differences in the bit numbers show the additional improvement achieved by the *Doppio-ESPRESSO* algorithm; *Doppio-ESPRESSO* achieves on average 20% more optimization than *ESPRESSO*. However, fewer programmable bits do not necessarily imply smaller areas, because PLA structures also contain components such as buffers etc. For WPLAs, there can be "white space" along the boundary and in the center, as illustrated in Figure 1.

example	NPLA	RPLA	WPLA	example	NPLA	RPLA	WPLA
s208.1	1623	1609	1038	s420.1	4691	4728	4439
s298	4053	4133	3800	s444	7152	7288	7046
s344	7559	7559	6234	s526	8208	8208	7490
s349	7902	7819	6582	s641	21175	21175	16583
s382	7428	7508	7198	s820	17862	18840	13675
s386	10200	10200	8228	s838.1	37353	35759	28032
s400	6575	6616	5714	s1488	53958	54884	44211
				s1494	56481	56070	47533
				Average	120%	120%	100%

Table 3. Bit counts of NPLA, RPLA (*depth*=2) and WPLA

Area/delay and synthesis times are given in Table 4. The "tech-indep. depth" refers to the depth during the technology-independent logic minimization. The values in the column are exact for the NPLA, RPLA and WPLA, since their logic levels will not change. However for the SC, there is a technology-mapping step after that, and the gate levels (including buffers when calculating levels) are shown in the "SC gate level" column. No placement or routing has been done for SCs and NPLAs, so these areas are just the raw areas of the logic components. Although we can assume that routing is done on higher metal layers, in reality, SCs may need cap cells on both sides of the rows and feed-thru cells. NPLAs require block-level placement, which may generate white space. The RPLAs have their finalized layouts, which contain white space, so they give fair comparisons. In addition, the delays of the SCs and NPLAs may change after routing, due to parasitics on wires. In contrast, WPLAs consume no additional area nor have additional delay uncertainties.

For a better view of the experimental results, the area/delay data of the SCs, NPLAs and RPLAs are normalized with respect to the WPLA results and plotted in Figure 3. The (1,1) point represents the WPLA single point for all examples. Connected points for SC, NPLA or RPLA represent area/delay trade-off curves for a single example. Figure 3 shows that SCs generally have larger areas than WPLAs, but can provide smaller delays if more area is allowed. NPLAs are just the opposite; they can provide smaller (raw) areas, but usually are slower. Comparing the *depth* = 2 cases, on average, WPLAs are 37% and 0% smaller than SCs and NPLAs respectively, but only 5% and 3% slower than SCs and NPLAs. However, recall that the areas of SCs and NPLAs only account for raw logic components and use more metal layers. After placement, the areas of both are expected to grow, especially NPLAs. So in reality, these area/delay curves would shift to the right relative to the WPLA point. The WPLA is on average 56% smaller than the *depth*=2 RPLA and 13% faster than it. The RPLAs are not expected to be as useful in implementing small circuits such as those in this experiment [13], because when the circuit is small or the depth is small, the river routing region may occupy a large portion of the entire RPLA area. A rough estimate of the river routing area can be obtained by the difference between the area of the RPLA and the NPLA. Comparing the area of SC, NPLA, RPLA and WPLA with similar delays (may have different depths), we find that WPLA is on average 19% larger than NPLA (raw area), 26% smaller than SC and 32% smaller than RPLA.

Note that some SC, NPLA and RPLA curves are not monotone decreasing with area; thus reducing the depth may not necessarily lead to faster circuits. Other curves are unpredictable in shape, so timing closure becomes even more difficult. Thus, in addition to uncertainty caused by physical design, area/delay relations of SCs, NPLAs and RPLAs are also unpredictable, while WPLAs do not suffer from such problems.

We also found that the number of gate levels after technology mapping is non-linear to the depth of the technology-independent optimized circuit, and the relationship is not even monotonic. An interesting phenomenon is that in some circuits like "s838.1", when the depth is reduced, the actual number of gate levels increases. This can be explained by two factors. One is from the covering in the technology mapping. Suppose the classical tree covering is used, where the technology-independent optimized netlist is first transformed into a generic netlist with only nand2's and inverters. If the depth is not small, the level of the generic netlist follows the depth quite well. But when the depth is very small, the nodes in the netlist are large, and many levels of nand2's and inverters have to be used to represent them. This makes the levels of the generic netlist and thus the mapped gate netlist almost unpredictable. The other factor is the loading problem. As the depth goes down, it is conceivable that the loads (on nets between nodes, and the SOP connections within nodes) tend to increase. To obey the load limit and improve speed, appropriate buffering should be done during technology mapping, which also increases the levels of gates. This shows that even within logic synthesis, the technology-independent step has difficulty predicting the behavior of the technology-dependent step. The relationship between depth, gate levels, area and delay is complicated.

Logic synthesis times for NPLAs and WPLAs are usually smaller than SCs, because SCs need a technology mapping stage, which becomes notably slower as the circuit size increases. The RPLA synthesis times are the slowest, due to its iterative node-placement algorithm [13].

[4] The *SIS* "reduce_depth –d x" command may not always reduce the depth to the designated value *x*, but to some value no greater than *x*.

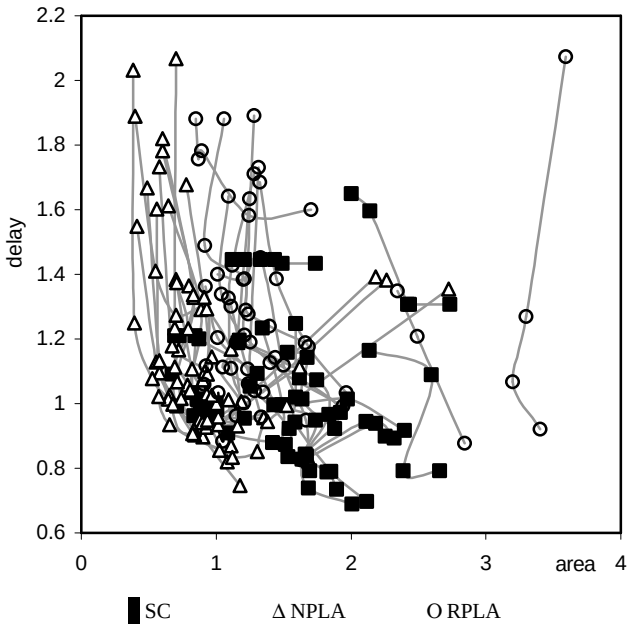


Figure 3. Normalized area/delay curves

5. Conclusions

Whirlpool PLAs (WPLAs) are logically four-level NOR networks. Their cyclic structure makes them compact. The design methodology for WPLAs involves only logic synthesis; no prediction is needed because area and delay are totally determined by the logic embedded in the WPLA. Doppio-ESPRESSO, a new four-level logic minimization algorithm for WPLA synthesis exploits additional structural flexibility. Experimental results show that WPLAs are quite competitive, in terms of area and delay, with standard cell implementations and network of PLA implementations, but are much more regular and predictable. It also is superior to another regular structure, the River PLA, in both area and delay, for the examples tested. A comparison between WPLAs and *depth* = 2 NPLAs and RPLAs also shows the advantage of the Doppio-ESPRESSO algorithm in terms of the total number of programmable bits needed to build a circuit. However, some remaining problems require more discussion:

(1) The regularity of a chip involves both local and global regularity. The WPLA provides a structure with local regularity. However to integrate multiple WPLAs on a chip and achieve global regularity is not easy. The problem includes, partitioning of the circuit into many WPLAs, placing and routing them in a regular way.

(2) The pin positions of the WPLA are fixed after the synthesis. This seems worse than for SC implementations. However consider implementing the same logic functions (a part of a large circuit) with SC. The gates are usually placed closer, although not necessarily in a rectangular region. The pins connecting to the external circuit are actually on some of the gates. It is unlikely that these pins can be moved arbitrarily, because the gates need to maintain some spatial relations indicated by the gate-level placement. The pins can move within a small range by moving the gates carrying them. To move them farther, the only way is to flip the entire "SC block". However changing orientation of a "SC block" is not as flexible as a WPLA, because the "SC block" cannot do things like "rotate 90^o". Therefore, the fixed pin position is not a serious drawback of the WPLA compared to the SC, because the WPLA can be thought of as "placed and routed".

(3) The PLA structures experimented with in this paper are static. These consume quiescent DC power because they use pull-up/down devices. In fact, dynamic PLA structures are more power efficient and are faster than static PLAs [17]. The WPLA structure can have a dynamic version, which is faster than its static counterpart.

(4) PLAs can also be re-sized to get different area/performance characteristics. Also the characterization of a set of PLA parameters is much faster than that of a library of hundreds of gates.

(5) Engineer Change Orders (ECOs) for SC implementations involve both synthesis and physical design modifications. But for the WPLA, it is mainly a synthesis problem.

(6) A programmable version of the WPLA is anticipated, and experiments need to be done to show if it is a good alternative to the LUT-based structures.

6. Acknowledgement

This work was supported by GSRC (grant from MARCO/DARPA 98DT-660, MDA972-99-1-0001).

7. References

[1] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis", Tech. Rep., UCB/ERL M92/41, Electronics Research Lab, University of California, Berkeley, May 1992

[2] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization", IEEE Transactions on Computer Aided-Design, vol. 6, Sep 1987, pages 727-750

[3] http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth91/

[4] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "Multi-level logic synthesis', Proc. of IEEE, vol. 78, Feb. 1990

[5] R. Brayton, G. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli, "Logic minimization algorithms for VLSI synthesis", Kluwer Academic Publishers, 1984

[6] R. Bryant, K-T. Cheng, A. Kahng, K. Keutzer, W. Maly, R. Newton, L. Pileggi, J. Rabaey, A. Sangiovanni-Vincentelli, "Limitations and challenges of computer-aided design technology for CMOS VLSI", Proceedings of the IEEE, vol. 89, issue 3, Mar 2001, pages 341-365

[7] S. Khatri, R. Brayton and A. Sangiovanni-Vincentelli, "Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric", Proceedings of International Conference on Computer aided Design, Nov 2000, pages 412-418

[8] Y. Iguchi, T.Sasao and M. Matsuura, "Realization of multiple-output functions by reconfigurable cascades", Proceedings of International Conference on Computer Design, 2001, pages 388-393

[9] T. Sasao, M. Matsuura and Y. Iguchi, "A Cascade Realization of Multiple-Output Function for Reconfigurable Hardware", International Workshop on Logic Synthesis, 2001

[10] F. Mo and R. Brayton, "River PLA: Structure and Design Methodology", Tech. Rep., University of California, Berkeley, 2001

[11] J. Cong, H. Huang and X. Yuan, "Technology Mapping for k/m-macrocell Based FPGAs", Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Feb 2000, pages 51-59

[12] A. Singh, G. Parthasarthy and M. Marek-Sadowska, "Interconnect Resource-Aware Placement for Hierarchical FPGAs", International Conference on Computer-aided Design, Nov 2001, pages 132-136

[13] F. Mo and R.K. Brayton, "River PLA: A Regular Circuit Structure", Design Automation Conference, Jun 2002, pages 201-206

[14] F. Mo and R.K. Brayton, "Regular Fabrics In Deep Sub-Micron Integrated-Circuit Design", International Workshop on Logic and Synthesis, Jun 2002, pages 7-12

[15] C.A. Papachristou, A.L. Pandya, "A Design Scheme for PLA-Based Control Tables with Reduced Area and Time-Delay Cost", IEEE Transactios on Computer Aided Design, vol. 9, no. 5, May 1990, pages 453-472

[16] F. Mo, A. Tabbara and R.K. Brayton, "A Force-Directed Macro-Cell Placer", International Conference on Computer Aided Design", Nov 2000

[17] Y.B. Dhong and C.P. Tsang, "High Speed CMOS POS PLA using Predischarged OR Array and Charge Sharing AND Aray", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 39, no. 8, Aug 1992, pages 557-564

example	tech-indep depth	SC gate level	area (1000um^2)				delay (ns)				synthesis time (min)			
			SC	NPLA	RPLA	WPLA	SC	NPLA	RPLA	WPLA	SC	NPLA	RPLA	WPLA
s208.1	8	9	4.91	1.72	8.82		3.12	3.91	3.92		0.1	0.1	1.2	
	4	9	5.26	1.70	8.10		3.02	2.33	2.49		0.1	0.1	1.3	
	3	9	5.94	1.82	7.86		2.47	1.92	2.02		0.2	0.1	1.2	
	2	9	5.98	2.22	8.36	2.46	2.47	1.70	1.74	1.89	0.2	0.1	1.2	0.1
	1	9	6.71	2.90	—		2.47	1.41	—		0.3	0.1		
s298	5	9	9.05	3.39	9.94		3.03	3.55	3.51		0.1	0.1	1.1	
	3	9	11.0	4.72	10.6		2.84	3.04	3.14		0.1	0.1	1.0	
	2	6	10.1	4.92	12.1	4.24	2.06	2.42	2.28	2.60	0.2	0.1	1.4	0.1
	1	6	11.3	3.73	—		2.06	2.48	—		0.2	0.1		
s344	9	13	9.83	5.08	8.81		3.94	5.92	5.76		0.1	0.1	1.2	
	5	11	14.8	7.06	8.46		3.56	4.41	4.33		0.1	0.1	1.2	
	4	11	15.5	5.84	10.1		3.21	3.69	3.65		0.1	0.1	1.3	
	3	10	14.1	7.62	10.6		2.78	3.56	3.50		0.2	0.1	1.2	
	2	11	18.4	8.57	10.1	8.44	3.12	3.10	3.09	3.32	0.2	0.1	1.2	0.1
	1	8	19.6	19.1	—		2.96	4.59	—		0.3	0.2	1.2	
s349	9	13	10.3	5.26	9.25		3.94	6.01	6.21		0.1	0.1	1.5	
	5	11	14.2	7.28	8.01		3.56	4.41	4.92		0.1	0.1	1.5	
	4	11	16.8	6.31	10.6		3.21	3.84	4.57		0.1	0.1	1.5	
	3	10	14.5	8.04	10.6		2.78	3.62	4.00		0.2	0.1	1.4	
	2	11	18.5	8.88	12.6	8.8	3.12	3.16	3.77	3.30	0.3	0.1	1.5	0.1
	1	8	19.7	19.1	—		2.96	4.59	—		0.4	0.2		
s382	8	13	12.6	4.14	12.6		3.56	5.23	5.52		0.2	0.1	1.5	
	4	11	15.5	5.60	11.6		3.12	3.81	4.02		0.2	0.1	1.5	
	3	10	15.4	6.61	10.6		2.82	3.47	3.76		0.3	0.1	1.5	
	2	9	17.1	8.43	13.6	10.1	2.68	3.10	3.50	3.38	0.4	0.1	1.5	0.1
s386	7	11	10.6	9.11	14.2		4.32	4.34	4.43		0.1	0.1	1.4	
	5	10	11.2	13.2	14.4		3.94	3.58	4.20		0.2	0.1	1.4	
	3	11	13.2	17.6	16.3		3.82	3.44	3.50		0.3	0.1	1.4	
	2	11	19.1	17.2	16.0	15.8	3.79	4.02	4.10	3.97	0.4	0.1	1.5	0.1
s400	8	12	12.7	4.64	10.5		3.88	5.38	5.38		0.1	0.1	1.4	
	4	10	12.8	5.42	10.1		3.17	3.66	3.70		0.2	0.1	1.4	
	3	8	12.7	6.34	9.96		2.93	3.28	3.31		0.3	0.1	1.4	
	2	10	14.9	7.44	9.20	8.0	2.46	2.88	2.99	3.11	0.3	0.1	1.4	0.1
	1	6	17.0	12.2	—		2.16	3.09	—		0.5	0.3		
s420.1	12	17	10.1	3.47	11.5		4.55	6.39	5.95		0.1	0.1	1.6	
	6	17	10.9	3.58	11.0		4.55	3.93	4.06		0.2	0.1	1.6	
	4	17	12.0	4.74	11.2		4.55	3.39	4.22		0.2	0.1	1.7	
	3	17	12.9	5.83	12.6		4.55	3.19	3.90		0.3	0.1	1.7	
	2	17	14.5	7.45	15.0	9.0	4.51	2.86	3.74	3.15	0.4	0.1	1.7	0.1
	1	17	15.7	14.6	—		4.51	3.50	—		0.5	0.2		
s444	8	13	12.1	4.41	11.5		3.98	5.38	5.53		0.1	0.1	1.6	
	4	9	13.5	5.23	10.1		3.22	3.66	4.61		0.2	0.1	1.7	
	3	8	13.7	6.13	10.0		2.82	3.22	4.20		0.3	0.1	1.6	
	2	9	16.4	8.10	12.1	9.0	2.54	3.04	3.09	3.23	0.4	0.2	1.7	0.2
	1	7	17.1	12.2	—		2.37	3.09	—		0.4	0.2		
s526	7	10	13.8	6.24	10.5		3.32	5.37	5.47		0.1	0.1	1.4	
	4	9	16.8	8.62	10.0		3.16	4.30	4.46		0.2	0.1	1.5	
	3	9	15.9	7.84	13.5		2.76	3.47	3.75		0.3	0.1	1.4	
	2	8	16.3	9.08	18.8	9.7	2.46	3.16	3.31	3.33	0.3	0.2	1.5	0.2
	1	9	19.5	10.5	—		2.29	2.73	—		0.5	0.2		
s641	14	18	13.1	5.94	12.7		5.29	8.33	8.30		0.1	0.1	1.3	
	7	16	19.5	8.19	13.0		4.82	6.22	7.75		0.2	0.2	1.3	
	5	15	22.2	10.5	13.4		4.39	5.62	7.86		0.3	0.2	1.4	
	3	13	29.5	13.9	18.6		4.47	5.70	6.98		0.5	0.3	1.3	
	2	12	35.8	14.0	25.4	14.9	4.04	4.81	7.06	4.41	0.6	0.3	1.3	0.4
s820	7	11	23.3	11.9	20.3		4.96	7.18	7.21		0.3	0.2	1.4	
	4	10	25.6	13.9	22.1		4.89	5.69	5.93		0.3	0.3	1.5	
	3	10	24.9	14.8	25.7		4.33	4.89	5.03		0.5	0.4	1.5	
	2	11	28.7	18.1	29.9	15.3	3.95	4.30	4.42	4.28	0.5	0.5	1.5	0.4
s838.1	20	21	23.0	18.6	33.5		8.67	11.5	10.0		0.3	0.2	1.5	
	10	21	24.1	19.1	39.9		8.67	7.32	9.91		0.3	0.2	1.5	
	7	21	28.1	21.8	44.1		8.67	6.70	10.4		0.3	0.2	1.5	
	5	21	27.2	34.0	49.8		7.41	6.13	8.02		0.3	0.2	1.5	
	4	21	30.7	37.1	55.2		6.92	5.98	6.79		0.3	0.3	1.6	
	3	33	33.4	43.3	55.8		7.02	6.10	5.81		0.5	0.4	1.5	
	2	33	36.1	45.8	54.1	33.2	6.51	6.77	5.92	7.16	0.7	0.6	1.5	0.6
	1	33	47.1	90.5	—		6.30	9.70	—		0.9	0.6		
s1488	5	11	46.5	38.2	59.2		7.04	9.62	9.21		0.5	0.5	2.7	
	3	11	49.9	44.5	60.3		6.96	7.68	7.70		1.6	0.7	2.8	
	2	11	54.2	50.5	65.3	54.2	6.87	7.15	6.96	6.94	2.0	0.8	2.8	0.9
s1494	5	11	47.0	38.4	50.2		7.11	9.67	9.58		0.8	0.5	2.9	
	4	11	50.5	43.1	55.1		7.02	8.66	8.47		1.0	0.6	3.2	
	3	11	48.2	38.8	67.2		6.92	7.51	77.9		2.4	0.8	3.3	
	2	10	54.6	52.6	70.1	54.5	7.05	7.21	7.31	7.03	3.2	0.9	3.3	0.9

Table 4. The comparison of area/delay and synthesis time