Checkerboard: A Regular Structure and its Synthesis

Fan Mo and Robert K. Brayton
Department of Electrical Engineering and Computer Sciences, University of California at Berkeley
{fanmo, brayton}@eecs.berkeley.edu

Abstract — *A regular circuit structure called a Checkerboard (CB) is proposed. In some CB configurations, all mask layers except the via layers are pre-designed, which is attractive for high manufacturability and performance predictability and for lowering mask costs. The synthesis algorithms developed for the CB makes use of their structural regularity and flexibility. No technology mapping is needed; placement and routing are integrated. Experimental results are favorable for CB up to about 10k gates, compared to other structures such as standard cells. For example, the Vk-CB version of CB uses on average only about 4% more area but is 8% faster.*

1. Introduction

Regular circuit structures become more important with the shrinking geometries of DSM, when manufacturability is a key issue [1, 4]. A few Programmable-Logic-Array (PLA)-based regular structures, the River PLA (RPLA) and the Whirlpool PLA (WPLA), have been proposed [2, 3]. Their design methodologies are simple, involving mainly technology independent logic synthesis.

In this paper, a new regular structure called Checkerboard (CB) is proposed. The basic element of the CB, called a block, is a layered structure. The lower layers, poly-silicon (abbreviated as "poly") and metal1, form a k-input k-output NOR array, which implements logic functions. A static configuration (wired NOR) is used. The higher layers, metal2 and 3, form a switchbox, implemented by via masks, for the global wires. A CB is an array of these blocks. In its fixed form (fixed k), the CB masks of poly, metal1, metal2 and metal3 are pre-designed and analyzed before real circuit design takes place. To implement the logic, only the connections between the layers, the via layers (we regard the contacts between poly and metal1 as vias), need to be determined and implemented with via masks.

The synthesis algorithm for CB inherits some of the simplicity of PLA synthesis; it does not need technology mapping, which saves significant time and manpower to migrate the methodology to a new process.. The algorithm starts with a technology independent optimization of a Boolean netlist. The result is decomposed into a network of OR gates with up to k binate inputs. Then the problem is to map this to a CB with $N_X \times N_Y$ blocks (we need to fix the number of blocks) and design the wires. There are two unique structural features. One is that gates in the same block with common inputs can share input pins, thereby reducing the number of connections. The other is that in each block, the input (poly) and output (metal1) lines are orthogonal and these can be permuted arbitrarily and independently. The situation with the global wiring layers (metal2 and 3) is similar. Using these features, an integrated placement and routing algorithm is developed, adopting a simple spine net topology [5].

The rest of the paper is organized as follows. In Section 2, the structure of the CB is described. Section 3 gives the design flow for CB synthesis. Experimental results are given in Section 4 and Section 5 discusses future work.

2. The structure of the Checkerboard

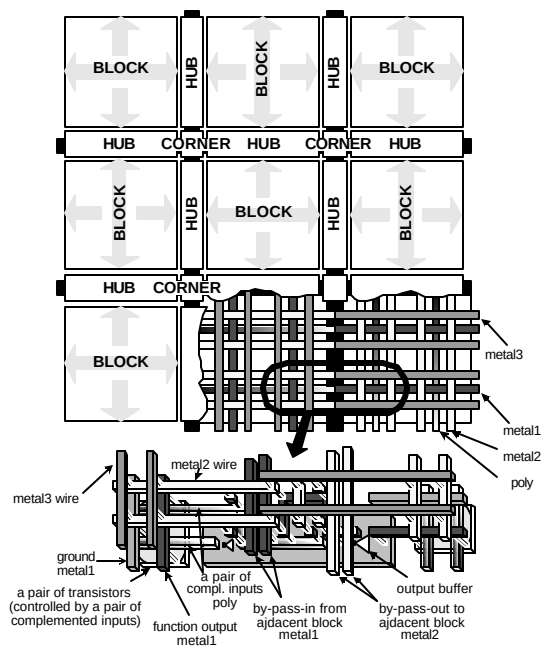


Figure 1. The Checkerboard structure.

The basic structure of the CB is illustrated in Figure 1. It is an array of blocks. The bottom layers of a block, composed of poly and metal1, form k NOR gates with up to k inputs. A NOR gate is a metal1 wire controlled by k input signals. We adopt a static structure; thus the output line has a pull-up resistor. The k gates in a block share the same k inputs, and both polarities of an input are available. Since the output of the NOR is buffered by an inverter, it is convenient to treat it as an OR gate with binate inputs. An input signal, on the poly layer and orthogonal to the metal1 wires, consists of a pair of complemented signals. Between each pair of meta1 signal wires, a metal1 wire is laid out and grounded (not shown in the upper portion of Figure 1 for simplicity); this is for the ground connections of the switching transistors. Above the OR gates are metal2 and 3, which are used for global interconnections. In each block, there are $2k$ metal2 wires and $2k$ metal3 wires, and they are orthogonal. Considering a pair of complemented poly lines as one signal, the signal densities of poly and metal1 are both k. Assume the CB is composed of $N_X \times N_Y$ blocks. The blocks at (x,y) that satisfies $(x+y)_{\mathrm{mod}2}=0$ are called "even blocks", while the others are "odd blocks". All the "odd" blocks are rotated by 90°. Adjacent blocks are connected by "hubs". A hub contains k repeated units;

one of which is detailed in Figure 1. Note that there is one input of the left block and one output of the right block here, while there are two metal2 and two metal3 wires. A hub unit has three major functionalities:

1. relaying global signals (metal2 and 3),
2. selecting input signals to the left block, and
3. selectively delivering the output of the right block.

The use of the by-pass wires is described in the next paragraph. All these are implemented by choosing a set of vias. The function of the corner is to relay the by-pass wires between adjacent hubs. By-pass wires use metal1 and 2, which only run across hubs and corners. The power supply and ground are spread in hubs and corners. For simplicity, they are not shown in Figure 1.

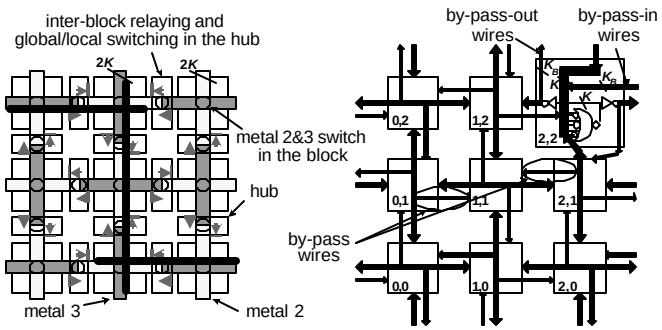


Figure 2. The abstract view of the Checkerboard PLA.

An abstract view of the CB structure is illustrated in Figure 2. Figure 2(a) shows the global interconnection network formed by metal2 and 3. If a long wire traverses blocks in either the X or Y directions, it alternates on metal2 and 3. The alternation between metal layers happens in the hub, which is called "relaying". The hub also provides for connecting between local and global signal levels. Since the metal2 and 3 segments in the blocks are fixed a priori, a wire on these layers must use the entire wire segment in the blocks. A wire can only break at a hub. However, a global wire can turn 90° to connect to another global wire inside the block though a via. An example is shown by the fat black lines in Figure 2(a). Figure 2(b) shows the local interconnection network. The blocks are labeled with their X,Y locations in the CB. The internal view of block (2,2) is detailed. Consider even block (1,1). Its k input signals can be chosen locally from among the

1. k outputs from its bottom neighbor (1,0),
2. k outputs from its top neighbors (1,2),
3. k_B by-passed outputs from the left block (0,1), and
4. k_B by-passed outputs from the right block (2,1).

To see the function of the by-pass wires (those narrow arrows in the figure), we examine four blocks in a cycle, (0,0), (1,0), (1,1) and (0,1), without the by-pass wires. The signals can flow in counterclockwise order through these four blocks. However if (0,1) wants an output from (0,0) as an input, a global interconnection would have to be accessed. To eliminate this, a few by-pass wires are laid out to facilitate signal flow in the reverse direction. By-pass wires are implemented in the hub, and cross the corners (four hubs share a corner) to enter adjacent hubs. Experimental results show that k_B=2 is sufficient in all test circuits.

If all the blocks in the CB are the same, except for the 90° rotation of the odd blocks, the structure is called a Constant-k CB (Ck-CB) and is parameterized by a single integer k. More generally, only the blocks in the same column/row need to have the same number of vertical/horizontal lines denoted by k_V/k_H. A more flexible configuration is that the columns are allowed to have different k_V and rows have different k_H. Such CB is called a Variable-k CB (Vk-CB).

The current version of the CB is combinational. To make possible a sequential CB structure, a new kind of block called latch-block can be created. The latch-block is similar to the OR block, except that the gates are replaced by latches. Since a latch occupies more layout area than a gate, the relationship between the maximum number of latches that can be placed in a block and the block size k_V/k_H needs to be set up. We have not developed an algorithm and done experiment on this. But the algorithm for pure combinational netlist, as described later, can be easily modified to do this job.

The area of a CB module can be derived, given the sizes of the blocks and hubs, which are both related to k (or k_H's and k_V's for Vk-CB), and N_X and N_Y, the numbers of blocks in X and Y directions. The delay formulation of the CB is straightforward. Any static timing analysis method can be applied. The circuit size we deal with, which is up to 10K gates, makes the wire delays negligible. However, as shown later, the algorithm is potentially suitable for incorporating wire delay computation. Note that the CB structure is static; therefore the gates in the same block operate independently. The sharing of input pins does not change the topology of the netlist. The delay computation starts by levelizing the netlist, and then propagating delays from the primary inputs to the primary outputs level by level. The delay of a gate is formulated as follows:

$$D(g) = d_C + n_I(g)d_{L1} + n_{FO}(g)d_{L2}$$

Here, d_C is a constant component, d_{L1} and d_{L2} are the load dependent components, $n_I(g)$ is the number of input pins of the gate, and $n_{FO}(g)$ is the number of gates this gate drives. The term with d_{L1} represents that a switching transistor must drive the drain-source capacitance of all the transistors (including itself) on the output line. d_{L2} represents the delay caused by the load to the output buffer. A fanout is an input line with k transistors plus an input inverter, hence d_{L2} is linear to k. This prevents using a very large k. Due to the sharing of input pins, n_{FO}(g) of a CB might be smaller than its original value in the input netlist. So the delay computed based on the initial netlist forms an upper bound. Also note that for Vk-CB, the k in the above discussion may be different from block to block.

3. Design flow

The design flow involves two stages, logic synthesis and physical design. The logic synthesis is simply a normal technology independent optimization followed by a decomposition into OR gates with up to k binate inputs. The task of physical design is to map the netlist of OR gates to a CB module with $N_X \times N_Y$ blocks. Because of 1) the structural regularity, 2) the free permutation of signal lines on poly and metal1 layers in the blocks and 3) the use of a spine net topology, placement and routing are merged. The design flow of Ck-CB and Vk-CB only differ in the cost functions.

3.1. Logic synthesis

Logic synthesis uses *SIS*, an existing synthesis package [6,7]. After technology independent logic optimization, the levels of the Boolean network are adjusted to make a trade-off between area and delay. Then the network is decomposed into a netlist of OR gates with the *SIS* command "tech_decomp –o k", where k is the size of the CB block, or the maximum number of outputs and the maximum number of inputs of a block. We call the input and output pins of a block "terminals". Input pins of gates placed in the same block that are on the same net can share and input terminal.

3.2. Physical design

Before starting the physical design, the number of blocks in the CB must be known. Since the number of gates is known, denoted by G, the number of blocks is calculated as:

$$N_X = N_Y = \left\lceil \sqrt{\frac{G}{uk}} \right\rceil,$$

in which, u is an utilization factor (u is normally 0.4~0.5). Normal values of k are around 10. Even for Vk-CB, we use the above equation with k=10 to determine N_X and N_Y. Recall that the decomposition in the logic synthesis step also needs an upper limit k' on the number of inputs to a gate.

Placement and routing are integrated in a single simulated-annealing framework. The key element is the net topology. It is unlikely that a simulated-annealing based placer can afford to use a Steiner Tree computation during every random move. Although approximate models can be used [9, 10], their estimation errors may cause non-convergence at the routing stage. We use a simple net topology, a spine. It was shown in [5] that the spine topology is acceptable in terms of wire length if the placement is done at the same time. In effect, the placement freedom can compensate for the limitations of the spine topology. In addition, by selecting between the vertical and the horizontal spines, detailed in sub-section 3.2.2, use of obviously bad spines can be avoided. The global routing problem becomes that of constructing spines for each net, where the segments of the spines are assigned to the columns and rows of blocks (abbreviated as "bands"). The manipulation of the spine net topology is fast. Despite the fact that the wire delay effect is ignored in this version of CB, it is very easy to compute wire delay on a spine topology. An important feature is that during the routing, the terminal locations (or the permutations of the output and input lines of the blocks) are not fixed. Because of the freedom of permuting input (poly) and output (metal1) lines in the blocks, the global routing can be finished first; then the routing results can be used to decide the permutation on these two layers. The I/O ports are placed external the CB module. They can be treated as terminals in the blocks surrounding the module, but do not really occupy those blocks; we only require the connections to reach them. The physical design flow is summarized in the following pseudo code:

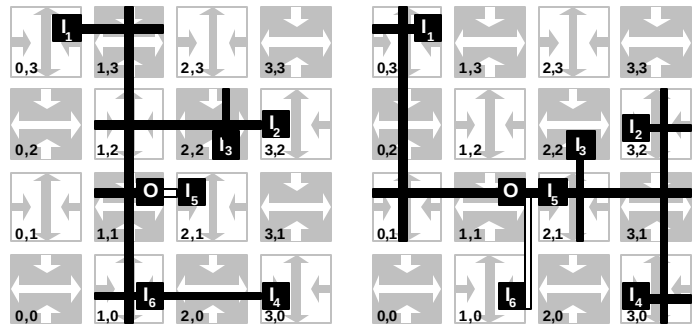
```
Simulated annealing framework {
    Randomly move a gate or swap a pair of I/Os.
    Update the terminals of the affected blocks (see 3.2.1).
    Construct spine topologies for the affected nets (see 3.2.2).
    Route the bands affected (see 3.2.3).
    Evaluate the cost function (see 3.2.4).
    If (rejected) restore the last placement.
}
```

3.2.1. Gate placement and terminal creation

When a gate is moved, or a pair of I/Os are swapped, only a subset of the terminals and nets are affected. The following routing steps only involve the nets that are affected.

3.2.2. Topological construction of nets

A fixed spine topology is used in the global interconnection of the CB. The output terminal of a net connects to a spine and all the input terminals reach the spine via ribs orthogonal to the spine. It is called a "vertical spine" if the spine is vertical and the ribs are horizontal; otherwise, it is called a "horizontal spine".



(a) vertical spine. (b) horizontal spine.

Figure 3. The spine net topology.

The construction of a spine takes linear time in the number of terminals on the net. We examine the wire lengths of the vertical and horizontal spines of a net, and choose the smaller one. The spines are built on a grid of the blocks, which can be regarded as a kind of global routing. Due to the rotation of the odd blocks, some terminals may not stay oriented correctly relative to the spine and/or ribs. In such cases, "turns" are necessary in the blocks carrying the terminals. The wire length evaluation takes these turns into account. Two special cases may further reduce the number of segments. One is where some input terminals are on the spine, and the other is that two or more input terminals are on the same rib. In addition, if input terminals are in the adjacent blocks of the output terminal, local connections can be used and can save global wiring resources. An example is illustrated in Figure 3. In the vertical spine, as shown in Figure 3(a), the output terminal O needs a turn, because the terminal direction is horizontal while the spine is vertical. Similarly, the input terminal I_3 also needs a turn. Input terminal I_5 does not need a rib, because its block (2,1) is adjacent to block (1,1), which contains the output terminal, and thus a local connection can be made. In the horizontal spine in Figure 3(b), input terminal I_6 can be connected via a local by-pass. In the vertical spine, I_6 can be connected in the same way. However since a rib is already available for the connection of I_4, I_6 joins that rib and saves a by-pass. The number of by-passes a block can access is limited, denoted by k_B. Using a by-pass is preferred. After the construction of a spine net, a set of wire segments are produced, which are assigned to bands. During the simulated-annealing, only one gate or a pair of I/Os are affected at each move, so only a few nets need to update their topologies. This involves deleting segments from and inserting segments into bands. Again only the routing of the affected bands need to be updated.

3.2.3. The routing of the bands

Since the permutations of metal1 (gate outputs) and poly (gate inputs) layers are independent, the terminal locations have a single degree of freedom within its block. For instance, in Figure 3, the Y location of the output terminal can be 1 to k within block (1,1). This gives flexibility in arranging the segments of the spine nets in their bands. A segment of the global wire can choose one of the 2k tracks from its band. Segments in the same band may have a compact arrangement, such that all fit in the band with no overlap. Thus the number of segments in a band can be much larger than 2k. The segment arrangements in different bands are independent. The arrangement of the segments determines the terminal locations.

The algorithm for the segment arrangement in a band is a greedy approach, which is similar to the interval packing or left edge algorithm [10]. There are 2k wiring tracks (alternating on

metal2 and 3) in a band, labeled 1 to $2k$, but only k local signal tracks (accessing inputs and outputs of the gates), labeled 1 to k. At most one of two global wiring tracks, $2j-1$ and $2j$ where $1 \le j \le k$, can access local track j of a block. If two global segments access the input/output of a block along the band they cannot be placed in an odd-even track pair. For each segment, a mask is created to represent which position(s) the segment accesses the terminal(s) of the block(s). The mask is simply a bit vector, with each bit indicating whether the terminal of a block is accessed. In Figure 3(a), for instance, the rib segment connecting input terminal I_4 has a mask of 0101. Of course, some segments can have zero mask, e.g. the vertical spine in Figure 3(a). Although they do not use any global wiring, local connections, both direct or through a by-pass, add terminal constraints. In such cases, pseudo segments are added in the band with zero lengths and non-zero masks. In Figure 3(a), the turn segment of the output terminal O, which is horizontal and of length one, has a mask of 0110. Its local connection to I_5 does not contribute to the segment length but it sets one bit in the mask. The interval packing algorithm is modified, by adding a check for mask violation. However, the optimality of the original algorithm is lost. The algorithm is given in the following pseudo code:

1. Order all wire segments in ascending order of their left edges. Label all segments as unassigned.
2. *CurrentTrack*=1. *CurrentMask*=0. *LastMask*=0.
3. *CurrentRightEdge*=0, *Updating*=false.
4. Pick up the next unassigned segment *m* in the ordered list.
5. If *LeftEdge*(*m*)≤*CurrentRightEdge*, go to 4.
6. If ((*CurrentTrack*=even) and (*Mask*(*m*) & *LastMask*≠0)), go to 4.
7. Assign *m* to the *CurrentTrack*. *CurrentRightEdge*=*RightEdge*(*m*). *CurrentMask* |= *Mask*(*m*). *Updating*=true.
8. If (*Updating*=false), *CurrentTrack*++, *CurrentMask*=*LastMask*, and go to 3. Else if (all segments assigned), exit, Else go to 4.

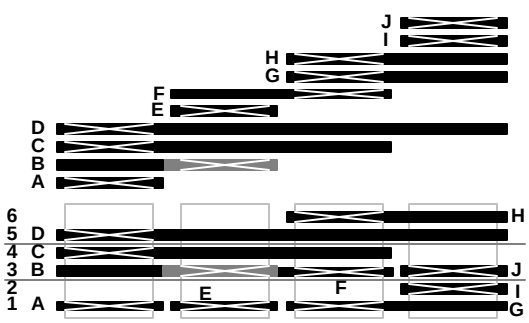


Figure 4. Band routing example.

An example is shown in Figure 4. A "1" bit in the mask shows in the figure as a white cross. Note that Segment B has a grey part with a cross. The left half of B is a real global wire segment. The right part is not, but it accesses the local wires (input or output of a gate). This happens when a horizontal rib accesses an input pin on the right. The reason why wire B, C, D, F or H cannot be placed in Track 2 is that they will incur mask violations. However, Wire D can be legally placed one track above Wire C, because tracks 4 and 5 belongs to different odd-even track pairs. The left half of F overlaps the grey part of B, which means in the second block, B accesses the input or output of a gate on a lower layer, while F uses global wire resources on a higher layer. It can be easily seen that, with the traditional interval packing algorithm, only five tracks are needed with the mask constraints dropped.

3.2.4. The cost function

The goal is to find a violation-free placement and routing for a given circuit on a given CB module with $N_X \times N_Y$ blocks with size of k for Ck-CB or variable sizes denoted by $k_V(x)$ and $k_H(y)$ for Vk-CB.[1] Although the two algorithms only differ in the cost function, the second algorithm produces a set of values for k that is most suitable for the logic being implemented.

(1) ***Ck-CB***: The cost function penalizes placement and routing violations defined below. If more than k-output terminals or input terminals appear in a block, the block is said to have a *placement violation*. Define the average and maximum placement violations as:

$$V_P = \frac{1}{N_X N_Y} \sum_{x=0}^{N_X-1} \sum_{y=0}^{N_Y-1} \left[\max\left(\left|T_O(x,y)\right| - k, \left|T_I(x,y)\right| - k, 0 \right) \right].$$

$$V_{PM} = \max_{x=0}^{N_X-1} \max_{y=0}^{N_Y-1} \left[\max\left(\left|T_O(x,y)\right| - k, \left|T_I(x,y)\right| - k, 0 \right) \right]$$

In the equations, $|T_O(x,y)|$ and $|T_I(x,y)|$ are the numbers of output and input terminals of block (x,y). Similarly, if a band needs more than $2k$ tracks to accommodate all the segments, then it incurs a *routing violation*. Define the average and maximum routing violations as:

$$V_R = \frac{1}{N_X + N_Y} \left[\sum_{x=0}^{N_X-1} \max\left(W(x) - 2k, 0\right) + \sum_{x=0}^{N_Y-1} \max\left(W(y) - 2k, 0\right) \right],$$

$$V_{RM} = \max\left[\max_{x=0}^{N_X-1} \left[\max\left(W(x) - 2k, 0\right) \right], \max_{y=0}^{N_Y-1} \left[\max\left(W(y) - 2k, 0\right) \right] \right]$$

in which W() is the number of tracks needed in the band. The cost function is

$$c = w \cdot \left(V_P + V_R\right) + \max\left(V_{PM}, \left\lceil \frac{V_{RM}}{2} \right\rceil \right),$$

where w is a small fraction. The 1/2 in the second term accounts for the difference of density $2k$ on global routing layers versus the density k of gate input/output layers. The goal is to reduce the second term to zero. Although, when this happens, the first term is also zero, without the first term, the annealing process may get stuck at high temperatures.

If after simulated-annealing the second term is non-zero,

$$k^* = \max\left(V_{PM}, \left\lceil \frac{V_{RM}}{2} \right\rceil \right) > 0,$$

then using a CB with $k+k^*$ and the same placement and routing results would give a violation free design, but this would not fit into an a priori fixed k configuration.

Note that doing only the physical design of a CB does not necessarily need k as an input. We can set k=0 and do the annealing, which outputs a k^* as the size of the CB blocks. This can be thought of as an indirect area minimization. An assumption is that we have a series of CB templates with different k's. However, the synthesis algorithm needs a k' to control the decomposition anyway. The modification of k is different from Vk-CB, because all the blocks in the CB are still the same in size (common k), although k is modified.

(2) ***Vk-CB***: Let $k_V(x)$ and $k_H(y)$ be the sizes of the blocks in column x and row y respectively, in which x=0,1,..,N_X–1, and y=0,1, …N_Y–1. As a placement of the gates and the routing are

[1] Variable-k can actually be applied in two different senses. The first is really a slight variation of constant-k. A *set* of k values is chosen a priori for both the rows and columns. This set is chosen independent of the logic to be implemented. We have only experimented with choosing all values of k equal. The second notion of variable-k is what is used in this paper. The set of values for k for the rows and columns is customized to the logic being implemented.

generated, the smallest k_V's and k_H's are chosen such that no violation occurs:

$$k_V(x) = \max\left\{ \max_{yy=0,ODD}^{N_Y-1} |T_O(x,yy)|, \max_{yy=0,EVEN}^{N_Y-1} |T_I(x,yy)|, \left\lceil \frac{W(x)}{2} \right\rceil \right\}.$$

$$k_H(y) = \max\left\{ \max_{xx=0,EVEN}^{N_X-1} |T_O(xx,y)|, \max_{xx=0,ODD}^{N_X-1} |T_I(xx,y)|, \left\lceil \frac{W(y)}{2} \right\rceil \right\}$$

Then the cost function is simply the area of the V*k*-CB:

$$A = \left[g_H(N_X - 1) + g_B \sum_{x=0}^{N_X-1} k_V(x) \right] \times \left[g_H(N_Y - 1) + g_B \sum_{y=0}^{N_Y-1} k_H(y) \right],$$

in which, g_B and g_H are the width of a unit of the block and the width of the hub, respectively.

4. Experimental results

We compared the CB with standard-cell (SC), River PLA (RPLA) and Whirlpool PLA (WPLA). A 0.35-µm technology was used since a fairly rich standard-cell library was available. Eighteen LGSynth91 benchmark examples were tested [8]. The first seven (s208.1~s820) are sequential circuits but with latches removed, and the last eleven examples (apex7~x3) are purely combinational. Each circuit was optimized using technology independent operations in *SIS* using "script.rugged". The levels of the resulting netlists were reduced gradually using command "reduce_depth -d". This allows a set of netlists with different area/delay tradeoffs to be generated; Smaller depth generally means faster but larger circuit. Each netlist was mapped to SC, RPLA, C*k*-CB and V*k*-CB. The mapping of CB starts with a decomposition to OR gates with up to *k'*=10 inputs, using the *SIS* command "tech_decomp –o". Then the integrated placement and routing is called. The X/Y numbers of blocks were determined by the gate number of the netlist, as described at the beginning of subsection 3.2. Thereafter, they are fixed. Both C*k*-CB and V*k*-CB were implemented. Only the level=4 netlist was mapped to WPLA, because WPLA is only a four-level structure. All programs were run on a DEC Alpha 8400 5/625 workstation.

The results are given in Table 1. The values in the parentheses after the circuit names are the levels used in the *SIS* "reduce_depth" command. The #gate column gives the equivalent gate numbers of the SC, which reflect the circuit size. The SC areas assume an 80% area utilization for routability concerns, which means the areas listed contain 20% white space. The areas of RPLA and WPLA already contain some white space and are fully routed. The delay computation does not take wire delays into account, because these testing examples are small so that gate or NOR-array delays dominate. The area and delay data of the WPLA, RPLA, C*k*-CB and V*k*-CB are normalized with respect to the SC values.

Table 1. Area/delay results

	#gate	area				delay			
name	SC	WPLA	RPLA	C*k*CB	V*k*CB	WPLA	RPLA	C*k*CB	V*k*CB
s208.1(16)	91		1.44	0.74	0.44		1.26	0.84	0.84
s208.1(8)	97		1.23	0.68	0.40		0.83	0.70	0.70
s208.1(6)	110		1.07	0.61	0.38		0.80	0.72	0.72
s208.1(4)	111	0.33	1.12	1.15	0.40	0.76	0.68	0.64	0.64
s298(10)	168		0.88	0.76	0.70		1.17	0.77	0.77
s298(6)	204		0.77	1.01	0.91		1.11	0.79	0.79
s298(4)	187	0.33	0.96	1.10	0.95	1.24	1.1	0.95	0.95
s382(16)	233		0.80	0.88	0.82		1.53	1.36	1.39
s382(8)	287		0.60	0.72	0.71		1.29	1.16	1.16
s382(6)	285		0.55	1.07	0.98		1.36	1.21	1.21
s382(4)	317	0.47	0.64	0.96	0.82	1.26	1.3	1.11	1.11
s400(16)	235		0.66	0.87	0.62		1.38	1.36	1.36
s400(8)	237		0.63	0.87	0.79		1.16	1.38	1.34
s400(6)	235		0.63	1.30	1.21		1.14	1.45	1.45
s400(4)	276	0.43	0.49	1.11	1.06	1.24	1.20	1.12	1.12
s444(16)	224		0.76	0.92	0.46		1.38	1.30	1.30
s444(8)	250		0.60	0.82	0.58		1.44	1.06	1.06
s444(6)	254		0.58	0.81	0.60		1.50	1.18	1.18
s444(4)	304	0.44	0.59	1.00	0.61	1.28	1.24	1.20	1.20
s526(14)	256		0.61	0.80	0.4		1.67	1.06	1.06
s526(8)	311		0.48	0.98	0.56		1.41	0.69	0.69
s526(6)	294		0.68	0.70	0.54		1.36	0.79	0.75
s526(4)	302	0.48	0.92	1.15	0.56	1.32	1.32	0.84	0.80
s820(14)	431		0.45	0.64	0.46		1.44	1.08	1.10
s820(8)	474		0.76	1.12	0.74		1.2	0.88	0.88
s820(6)	461		0.8	0.89	0.56		1.16	0.95	0.95
s820(4)	531	0.49	0.96	1.38	0.83	1.08	1.10	0.98	0.98
apex7(16)	430		0.2	0.33	0.33		1.10	0.98	1.00
apex7(10)	522		0.31	0.47	0.40		1.18	1.00	1.02
C1355(20)	900		1.02	0.99	0.89		1.32	0.81	0.79
C1355(14)	947		0.78	1.20	1.12		1.19	0.58	0.58
C1355(10)	1.5k		0.8	1.52	1.32		1.11	0.59	0.58
C2670(24)	1.5k		2.01	1.36	1.26		1.10	1.48	1.42
C2670(18)	2.2k		1.45	2.19	2.00		1.03	0.96	0.93
C3540(26)	5.3k		0.56	0.96	0.93		1.05	1.15	1.10
C3540(18)	6.4k		0.54	2.54	2.43		1.02	1.17	1.13
C5315(30)	3.1k		1.12	0.87	0.86		1.13	0.57	0.57
C5315(16)	4.0k		1.61	1.02	0.94		0.73	0.58	0.58
C5315(12)	5.2k		1.44	1.21	1.11		0.78	0.65	0.65
C6288(50)	7.9k		1.38	0.91	0.90		1.24	1.56	1.56
C6288(25)	8.8k		1.90	2.10	1.81		1.15	1.15	1.13
C6288(18)	16.4k		1.59	2.05	2.03		1.26	1.32	1.31
C7522(36)	5.1k		2.14	1.42	1.32		0.76	0.93	0.93
C7522(28)	6.1k		1.89	2.29	2.28		0.77	0.97	0.97
C7522(18)	9.2k		1.32	2.2	2.41		0.75	0.8	0.8
C7522(12)	10.9k		1.20	2.76	2.83		0.91	1.08	1.08
i8(14)	2.2k		1.47	0.94	0.86		0.70	0.56	0.56
i8(10)	2.0k		1.29	1.98	1.88		0.65	0.61	0.61
i10(44)	4.8k		1.44	1.20	1.18		0.70	0.77	0.74
i10(26)	9.2k		1.00	1.96	1.95		0.69	0.92	0.86
i10(18)	13.3k		0.99	1.79	1.76		0.94	1.01	0.98
k2(28)	2.3k		1.35	0.93	1.03		1.27	0.44	0.45
k2(16)	2.8k		1.35	0.99	0.97		1.21	0.42	0.41
k2(8)	5.7k		0.75	0.57	0.56		1.04	0.45	0.44
x3(16)	1.7k		1.25	1.03	1.00		1.00	0.46	0.45
x3(8)	1.8k		1.24	1.17	1.04		1.09	0.33	0.32
average		0.49	1.00	1.18	1.04	1.07	1.11	0.93	0.92

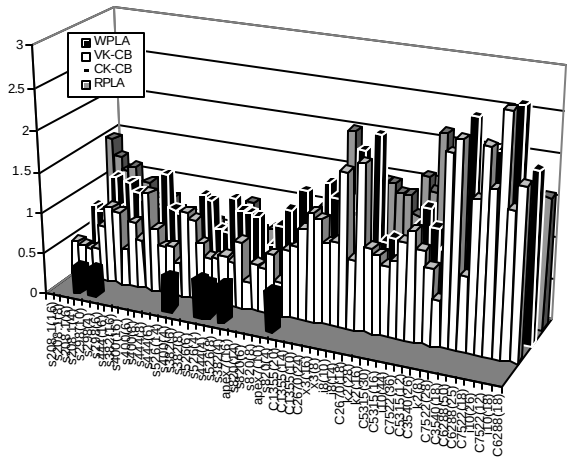


Figure 5. Area comparison.

Comparing the results, C*k*-CB averages 18% more area but 7% less delay than SC. In comparison with RPLA, C*k*-CB is 18% larger in area and has 18% less delay. Versus SC, V*k*-CB is 4% larger and 8% faster. Although WPLA is very small compared to other three, it is only appropriate for four-level circuits and has 7% more delay than SC. Figure 5 plots the area data in ascending order of circuit size. It indicates that usually the area of CB gets worse as circuit size increases. The run time of the CB algorithm is about 10 times that of SC, but SC run time does not include placement and routing.

5. Discussion

Checkerboard is a regular circuit structure. All the layers except the via layers are pre-designed, hence manufacturability issues can be analyzed and optimized well, independent of the circuit design. The mapping of a circuit to a CB module consists of a decomposition into OR-gates with up to *k* inputs and the integrated placement and routing of the gates. The spine net topology greatly simplifies the evaluation of wire length and routability. In a future extension to a timing driven version, this is a big advantage. Following are some current disadvantages and discussions of possible solutions.

1. Decomposition. The current CB structure has a low utilization of the gates in the blocks. This is partly because the decomposition of a Boolean network results in many gates with a small number of inputs. There are two possible solutions. One is to employ a folding technique that allows higher utilization of the block gates. Folding can be applied on the input and/or output lines. However, the routing may become harder, since permutability of the signals is partially lost. Another solution is to postpone the decomposition of wide gates until the physical design stage. Since the number of wide gates is usually very small, decomposing them "on-the-fly" can be an option. Thus, whenever a wide gate is moved during annealing, signals in the surrounding blocks are examined to find input signals of the wide gate. Then the decomposition is based on this information. In the technology mapping for standard-cell, a similar situation exists, that is, variable decompositions exist at certain nodes. However such nodes in technology mapping are too many.

2. Difference between CB, FPGA and Gate Array (GA). In a GA style design, an array of transistors is fabricated; only a few masks for the interconnection need to be designed. This reduces time to market. The main purpose of developing the CB structure is to reduce the number of layout patterns for easier manufacturability analysis and optimization, although pre-fabrication of lower layers (up to poly) is feasible. The FPGA is a circuit such that its functionality is determined after it is fabricated. The logic function and interconnection of the FPGA are field programmable. The CB is not programmable in this sense. With C*k*-CB only the masks of the vias need to be designed; more concretely, the mask output specifies if a via should be made at a pre-defined location.

3. Comparison of CB, WPLA and RPLA. In addition to the size of circuits that each type can effectively handle, there is a difference in chip-level integration of multiple modules. For RPLA and WPLA, global regularity is low if many are integrated on a chip. The CB, C*k*-CB in particular, is potentially suitable for whole chip implementation without loss of regularity. Block-level placement and routing and more metal layers are required [5]; all the CB modules would use the same *k*. In addition, the additional metal layers would use similar regular patterns, thus global regularity would be maintained.

4. The circuit size a CB can handle. Rent's rule [11] indicates that the single CB structure is not suitable for circuits larger than 10k gates. The CB structure only contains two global wiring layers, metal2 and 3. Consider a square region composed of $n \times n$ blocks. The maximum number of wires that can cross the boundary of this region is

$$P_{CPLA} = 4 \cdot n \cdot 2k,$$

in which, "4" comes from the four edges and "2*k*" is the wiring density of a block. In this region, the number of gates is:

$$G = u \cdot n^2 \cdot k,$$

in which, u is the utilization. Rent's rule gives an estimate of the number of external connections of this region [11],

$$P_{RENT} = r_1 \cdot G^{r_2},$$

where r_1 and r_2 are the Rent's coefficient and exponent, respectively.

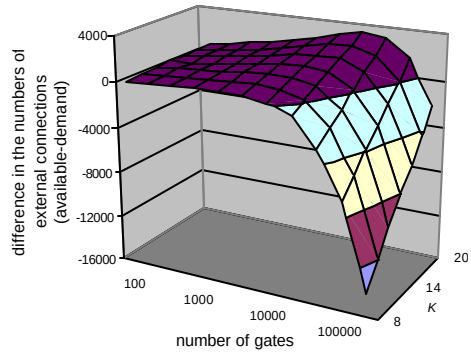


Figure 6. Estimating number of external connections.

Figure 6 illustrates the number of external connections the region can provide versus that predicted by Rent's rule. Based on a utilization u=0.5–0.01(k–8), the number of blocks is derived and the number of global wires that cross the boundary of the region is obtained. The computation with Rent's rule uses r_1=3 and r_2=0.75. A negative value in the difference of the numbers of external connections provided by the CB and predicted by Rent's rule means possible global wiring congestion. A direct result is the necessity of increasing *k* after simulated-annealing. The figure shows that for the same number of gates, larger *k* is better. However, as mentioned before, large *k* leads to more delay. This prevents the building of large circuits using large *k*. Also as *k* becomes large, the utilization

may drop, because in the decomposition, many gates are 2- or 3-input no matter how large k is. Therefore, the size of a circuit suitable for CB implementation should be limited to about 10K gates.

5. Power dissipation and variants of the CB. Static NOR-arrays consume static power, which is a disadvantage for modern IC design. Direct extension to a dynamic version may not be feasible, because the "hand-shake" signals which control the pre-charging/evaluation are hard to generate and propagate. One possible solution is to use a pipelining configuration. The odd-blocks operate under one phase, the even-blocks work under another phase. To make this possible, the gates should be placed in the blocks compatible with their phases. A second solution is to adopt NMOS pull-ups, instead of resistor pull-ups, which are controlled by the complemented signals of the inputs (in contrast, PMOS transistors use the original signals). One drawback is a threshold voltage loss, but the signal levels will be recovered by the subsequent buffers. Another problem is the delay caused by the serialized pull-up NMOS transistors. When number of inputs is large, or the pull-up NMOS chain is long, the output rise time is slow. Thus if this scheme is to be used, small k is preferred, possibly by decomposing wide gates "on-the-fly".

6. The metal2/3 wiring scheme. The current wiring scheme for metal2/3 may cause large number of segments and vias for long interconnections, because every time a wire crosses a block, it alternates the layers. The original motivation of using such a scheme is to maintain a fine granularity of the metal2/3 routing grid, such that whenever a wire turns, it consumes at most one metal2 segment and one metal3 segment inside that block. It might be better to adopt a scheme with both long segments and short segments, similar to the one in FPGA. Recall that each input line in the bottom logic block corresponds to two metal2 segments, and each output line corresponds to two metal3 segments. We may let half of the metal2 segments to be long segments that span several blocks, while the other half are still within the ranges of the blocks. The short segments may directly have connections to the input pins. Symmetrical assignment can be applied to metal3 segments. Then the band routing algorithm may need modification.

Acknowledgement

This work was supported by GSRC (grant from MARCO/DAPPA 98DT-660, MDA972-99-1-0001). We gratefully acknowledge support from the California Micro program and our industrial sponsors, Cadence and Synplicity.

References

[1] M. Lavin and L. Liebmann, "CAD Computation for Manufacturability: Can We Save VLSI Technology from Itself?", ICCAD 2002, pages 424-431.

[2] F. Mo and R.K. Brayton, "River PLA: A Regular Circuit Structure", DAC 2002, pages 201-206.

[3] F. Mo and R.K. Brayton, "Whirlpool PLAs: A Regular Logic Structure and Their Synthesis", ICCAD 2002, pages 543-550.

[4] "Silicon VLSI Technology", Chapter 5, Lithography, edited by J.D. Plummer, M.D. Deal and P.B. Griffin, Prentice Hall, 2000

[5] F. Mo and R.K. Brayton, "Fishbone: A Block-Level Placement and Routing Scheme", ISPD 2003, pages 204-209.

[6] 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

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

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

[9] J.L.Ganley, "Accuracy and Fidelity of Fast Net Length Estimates", ACM VLSI Integration, the VLSI Journal, vol.23, no.2, Nov, 1997, pages 151-155.

[10] N.A.Sherwani, "Algorithms for Physical Design Automation", kluwer Academic, 1993.

[11] B.S. Landman and R.L. Rosso, "On a Pin Versus Block Relationship for Partitions of Logic Graphs", IEEE Trans. Comp, C-20, 1971, pages 1469-1479.