On Designing Via-Configurable Cell Blocks for Regular Fabrics

Yajun Ran
ranyj@ece.ucsb.edu

Malgorzata Marek-Sadowska
mms@ece.ucsb.edu

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106 USA

ABSTRACT

In this paper we describe the design process of a via-configurable block for regular fabrics. The block consists of via-configurable functional cells, via-decomposable flip-flops, and via-configured sizable repeaters. The fabric has fixed layers up to M2. An M1-M2 via mask is used to define the block's functionality. The upper-level metals are customized. Compared to other structures based on LUTs or PLAs, and fixed flip-flops, our block has much smaller area, higher performance and lower power consumption.

Categories and Subject Descriptors: B.7.1 [Types and design styles]: Gate Arrays

General Terms: Design

Keywords: Regular fabric, via configurable, layout

1. INTRODUCTION

There is a big gap in performance, area and power between standard cell-based ASICs and FPGAs. Structured ASICs can potentially fill this gap [14]. Such fabrics have most of their parts prefabricated and designers need only to customize a few masks to complete the whole design. Designs produced in structured ASIC style should have shorter development and manufacturing times, as well as lower cost, compared to standard cell-based ASICs. They should have higher performance, gate density, and lower power consumption compared to FPGAs.

Structured ASICs are built as arrays of basic blocks composed of combinational and memory elements. A good basic block (essential for a successful fabric) should provide powerful functional expression, high performance and integration density, and flexibility to meet different application needs. A common block structure is based on look-up tables (LUTs). An n-LUT can implement any n-input function, which simplifies the synthesis process [10]. In [11], the authors propose a hybrid basic block consisting of a LUT, a D-FF and several NAND3 gates. In both cases the intra-cell connections are fixed, and vias program inter-cell connections. PLAs are an attractive alternative as components of basic blocks, since any function can be written as a sum-of-product (SOP) form and directly mapped into a PLA. Due to their regularity and predictability, PLAs were proposed as building blocks in a cell-based flow for designing large circuits, but not for a fabric [5, 8, 9].

In this work, we propose a novel block for regular fabric and explore the opportunities brought by the via programmability. Our block is composed of via-configurable functional cells, via-decomposable flip-flops, and a via-configured sizable-repeater-array. A via-decomposable flip-flop can serve as a memory unit or can be decomposed into several parts by configurable vias and then used for combinational purposes. It nicely resolves the concern that a fabric with too few or too many flip-flops is area-wasteful. A via-configured sizable repeater array provides repeaters with several driving strengths.

The present version of our fabric has prefabricated transistors, contacts, and the M1 (metal 1) layer. The M2 (metal 2) mask is also fixed. All the poly, M1 and M2 metal layers (the most challenging parts for manufacturing) have very regular geometries and uniform density. The M1-M2 vias are customized to provide the required functionality. The higher metallization layers (less challenging to manufacture) are designed in a standard-cell-like fashion.

Experimental results show that our fabric achieves much better area, performance and power consumption compared to LUT-, PLA- and hybrid-block-based structures. It comes close to standard cell-based design in terms of area and performance, but has much better manufacturability, shorter time-to-market, and lower cost.

The paper is organized as follows. In Section 2 we describe the via-configurable functional cell. In Sections 3 and 4 we discuss the via-decomposable flip-flops and via-configured sizable repeater array. In Section 5 we explain the evaluation flow used for comparisons. In Section 6 we present the experimental results. Section 7 concludes the paper and discusses future work.

2. VIA-CONFIGURABLE FUNCTIONAL CELL (VCC) DESIGN

2.1 Basic structure of the VCC

We assume that the via-configurable functional cell (VCC) will be manufactured in static CMOS technology, though a dynamic VCC cell can be similarly designed. We also assume that P- and N-networks of the gates configured

DAC 2004, June 7–11, 2004, San Diego, California, USA.

from the VCC consist of series-parallel connected transistors which are geometric duals. This is a general assumption; most CMOS gates fall into this category. Our VCC has a layout style similar to that of [13] with emphasis on via configurability. The main characteristics of the VCC layout are:

1) Each cell has a single n- and p-diffusion strip.
2) Complementary p- and n-MOS transistor-pairs are vertically aligned.
3) M1-metal (vertical) segments extend from the ground and Vdd to the center of the VCC for possible connections from transistor terminals to the M2 (horizontal) segments.
4) Potential via sites are at the intersects of M1 and M2.
5) Vdd and ground use M2 in parallel to the diffusion strips.

Our VCC cell is parameterized by the number of transistor pairs n it contains. Given n, we determine the number of M2 rows needed to make all the connections, and we generate automatically the layout of the n-VCC.

Figure 1 shows an example of a 4-VCC, which realizes the function $f = \overline{(x_1 + x_2)(x_3 + x_4)}$. The M2 segment W3 is not used by the function and can be used for inter-cell routing.

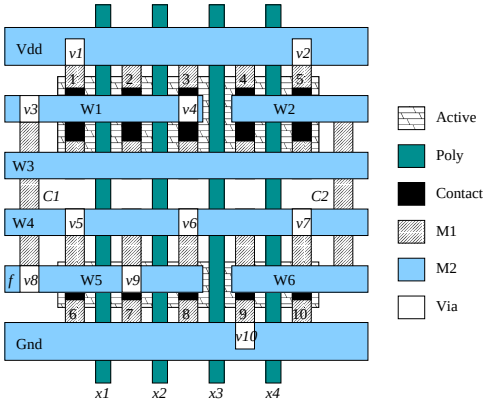


Figure 1: VCC layout model ($f = \overline{(x_1 + x_2)(x_3 + x_4)}$)

2.2 Properties of the VCC

2.2.1 Cell layout basics

In [13], Uehara and van Cleemput proposed a graph model for transistor circuits in which a labeled edge represents a transistor and nodes represent connected transistor source and drain terminals. A circuit consists of two two-terminal series-parallel multigraphs (TTSPMs), M for an N-network and M^d for a P-network. The M and M^d are the geometric duals. Figure 2 gives an example, where (a) shows a transistor circuit and (b) shows the corresponding graph. The symbols N (north pole) and S (south pole) represent the two terminals of each TTSPM.

We follow the terminology of [7]. A *trail* in a graph is an alternating sequence of nodes and edges beginning and ending with nodes, with no edge repeated. A *dual trail* is a pair of trails, one in M and the other in M^d,

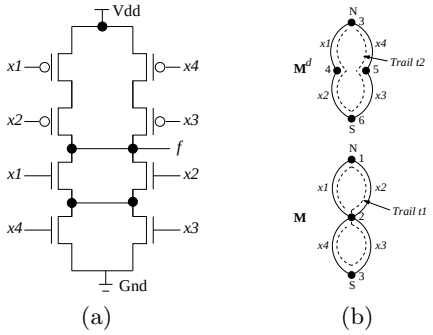


Figure 2: Trans. circuit (a) and its graph model (b)

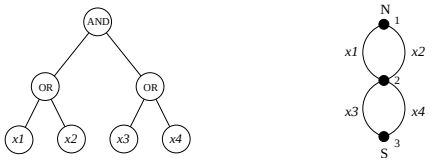
which consist of the same sequence of dual edges. A *dual Euler-trail* is a dual trail which includes all the edges of a graph. For example, $t_1 = \{2, x_1, 1, x_2, 2, x_3, 3, x_4, 2\}$ and $t_2 = \{6, x_1, 4, x_2, 3, x_3, 5, x_4, 6\}$ in Figure 2 are two trails and also a dual Euler-trail.

The transistors in a dual Euler-trail can be placed in one diffusion strip following their order in the trail, because two adjacent edges in the graph share transistor terminals and can be abutted. For example, Figure 1 shows the layout of the circuit in Figure 2(a). The traditional cell-layout optimization problem seeks the minimum number of dual Euler-trails (the trail cover) for a circuit. Our problem is different, because we have laid out one-diffusion-strip transistor-sequence and then determine what functions can be implemented by introducing the M1-M2 vias.

2.2.2 Functional coverage

A function realized by a single stage CMOS gate can be represented as $\bar{f} = \bar{f}(x_1, ..., x_n)$ where only AND and OR operations are allowed among input variables. Such a function is formed by a series-parallel connected transistor network and can also be represented by a *composition tree* T [7]. Leaf-nodes in T correspond to the literals in the function (transistor inputs in the circuit). The interior nodes are labeled with operation types (AND (*) or OR (+)). Each function with more than one literal has the composition-tree depth larger than 1. A function and its TTSPM can be constructed by visiting the composition tree in a pre-order and applying the corresponding operations. When several branches meet in TTSPM, AND implies a serial connection and OR a parallel. Figure 3(a) shows a composition tree for the function $\bar{f} = (x_1 + x_2)(x_3 + x_4)$ and 3(b) shows the corresponding TTSPM (N-network only). A composition tree is ordered such that the left-side children of an AND node correspond to the branches in the TTSPM closer to a specified terminal, N or S. Two functions are *composition tree equivalent* if their composition trees differ only in the children-orders of the corresponding nodes (commutative law in Boolean algebra).

f is an n-function if it has n literals and can be realized by one CMOS stage with n transistor pairs. A negative literal in f means that the corresponding input signal has been inverted by the previous stage. We treat it in the same way as positive literals. For example, $\bar{f} = (x_1 + \bar{x}_1 x_2)$ is a 3-function with transistor inputs x_1, $\bar{x}_1$ and x_2. Let f_1 and f_2 be n-functions with $X = \{x_1, \cdots, x_n\}$. f_1 and f_2 are *P-equivalent* if there exists a permutation β such that $f_1(\beta(X)) = f_2(X)$. Two P-equivalent functions can be mapped to the same tran-



(a) composition tree (b) TTSPM

Figure 3: Function $\bar{f} = (x_1 + x_2)(x_3 + x_4)$

sistor circuit with different transistor input assignments. In the sequel, we call two functions equivalent if they are either composition-tree-equivalent or P-equivalent.

The functional coverage of a n-VCC is defined as

$$FC(n) = \frac{RF(n)}{TF(n)} \tag{1}$$

where $RF(n)$ is the number of non-equivalent functions realizable by an n-VCC. $TF(n)$ is the number of all possible non-equivalent n-functions.

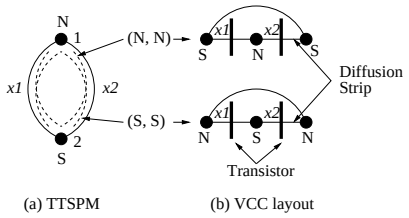
An n-function corresponds to a TTSPM with n edges. Therefore $TF(n)$ is equal to the number of essentially different TTSPMs with n edges, which fulfill the following properties [6]

$$nTF(n) \quad = \quad 1 + \sum_{i=1}^{n-1} TF(i)(1 + e_{n-i}) + e'_n \tag{2}$$

where $e_i = \sum_{j|i} jTF(j)$ and $e'_i = e_i - iTF(i)$.

Our VCC allows us to connect any two transistor terminal points. If an n-function has a single dual-Euler trail, it can be implemented by an n-VCC. We enumerate all the qualified n-functions by finding all the dual Euler-trails with n edges and extracting non-equivalent n-functions from them.

Their endpoints determine if two trails can concatenate under AND or OR operation. There are three types of a trail endpoint: N, S, I (internal node) [7]. A trail in a TTSPM is characterized by the types of its endpoints. There are 6 trail types represented by unordered pairs: (N, N), (S, S), (N, S), (N, I), (S, I), (I, I). Consider Figure 4. When an (N, N) trail and an (S, S) trail are mapped to a VCC, they have the same structure except that the labels N and S are exchanged. Therefore, we can reduce (N, N) and (S, S) to a single trail type (E, E). Our notation differs from [7], modified for clearer explanation. Figure 5 shows examples.



(a) TTSPM (b) VCC layout

Figure 4: Trail type (N, N) and (S, S)

Consider two TTSPMs T_1 and T_2 with trails t_1 and t_2 which are to be ANDed (connected in series) or ORed (connected in parallel). t_1 and t_2 can be concatenated to form a longer trail if one endpoint of each trail is a terminal of T_1 and T_2 (N or S). Figure 6 shows an example. By checking the types of the two endpoints of the newly formed trail, we determine its type. Table 1 lists the trails concatenated from two trails for AND and OR operations; "-" stands for

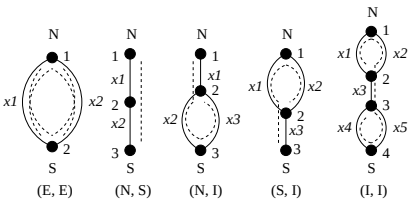


Figure 5: Trail types

infeasible concatenation.

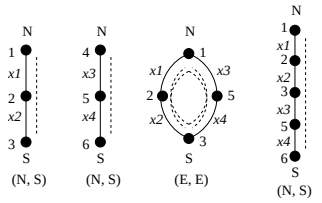


Figure 6: Trail concatenation

	AND					OR				
	(E, E)	(N, S)	(N, I)	(S, I)	(I, I)	(E, E)	(N, S)	(N, I)	(S, I)	(I, I)
(E, E)	(I, I)	(S, I)	(I, I)	-	-	(E, E)	(N, S)	(N, I)	-	-
(N, S)	(N, I)	(N, S)	(N, I)	-	-	(N, S)	(E, E)	(N, I)	-	-
(N, I)	-	-	-	-	-	-	-	-	-	-
(S, I)	(I, I)	(S, I)	(I, I)	-	-	(S, I)	(S, I)	(I, I)	-	-
(I, I)	-	-	-	-	-	-	-	-	-	-

Table 1: All concatenation cases for two trails

We represent each dual Euler-trail by an ordered pair [TN, TP], where TN (TP) is the type of its N(P)-network Euler-trail. For example, the dual Euler-trail in Figure 2 can be represented as [(I,I), (E, E)]. Two dual Euler-trails dt_1 = [TN_1, TP_1] and dt_2 = [TN_2, TP_2] can be concatenated for the operation $\odot$ (AND or OR) only if both Euler-trails of the N- and P-network can be concatenated. The resulting dual Euler-trail dt = [TN, TP] can be represented as

$$\begin{aligned} TN &= TN_1 \odot TN_2 \\ TP &= TP_1 \odot^d TP_2 \end{aligned} \tag{3}$$

Where AND^d=OR, OR^d=AND.

THEOREM 1. *Any dual Euler-trail with n ($n > 1$) edges can be split into two dual trails at any internal point-pair, of whose both points connect to the corresponding edges in each trail.*

Proof: Omitted due to page limit.

COROLLARY 1. *Any dual Euler-trail with n ($n > 1$) edges can be obtained by an AND or OR operation on two dual trails.*

Proof: Omitted due to page limit.

The inverse of the Corollary 1 does not hold, i.e. not every two dual trails can form a dual Euler-trail because they may not be concatenatable. Corollary 1 tells us that if a dual Euler-trail dt with n edges exists, there must exist a number i such that dt can be formed by ANDing or ORing a dual trail with i edges and a dual trail with $(n - i)$ edges. If we enumerate all the possible concatenations of dual trails with i edges and dual trails with $(n-i)$ edges for $1 < i < n$ under AND or OR operations, we must find dt. This forms the basis of our dual-n-edge trails enumeration (corresponding to all n-VCC implementable functions). It can be expressed

as

$$\Phi_n = \bigcup_{i=1}^{\lfloor n/2 \rfloor} \left[(\Phi_i \text{ AND } \Phi_{n-i}) \cup (\Phi_i \text{ OR } \Phi_{n-i}) \right] \quad (4)$$

where Φ_i is the set of dual trails with i edges.

The idea of the enumeration procedure is as follows. When we obtain a feasible dual Euler-trail from two shorter dual trails, we add it into the dual Euler-trail set Φ_n only if it is irredundant. A dual Euler-trail with n edges is redundant if its related function is equivalent to and has the same dual Euler-trail type as some already existing dual Euler-trail in Φ_n. During the enumeration, we keep all the functionally equivalent but irredundant dual Euler-trails to allow possible concatenation when constructing longer dual Euler-trails later.

Two functions are non-equivalent only if their composition trees have different root node types, or their composition trees are not isomorphic after erasing all the node labels.

Tree isomorphism problem can be solved in linear time [1]. After obtaining all the non-equivalent functions, we count them to compute the functional coverage of an n-VCC. We also build our VCC cell library based on these functions for technology mapping.

Table 2 lists the functional coverage up to 8-VCC.

n	2	3	4	5	6	7	8
$TF(n)$	2	4	10	24	66	180	522
$RF(n)$	2	4	10	22	50	112	280
$FC(n)(\%)$	100	100	100	91.7	75.8	62.2	53.6

Table 2: Functional coverage of n-VCC

2.2.3 VCC M2 wire segmentation

To decide how many M2 rows are needed in an n-VCC, we have the following theorem.

THEOREM 2. *At most $2(\lfloor (n-1)/2 \rfloor + 2)$ M2 rows are needed in an n-VCC to realize any circuit which contains n transistor pairs and has a single dual Euler-trail.*

Proof: Omitted due to page limit.

M2 wires can be segmented to achieve better performance. A good segmentation scheme should not lose the functional implementation capability. Based on Theorem 2, when the transistor count increases by two, the upper bound of M2 rows of N(P)-network increases by one. We therefore suggest a wire segmentation scheme depicted in Figure 7 (only half of the network is shown since N and P parts are symmetric). Vertical lines are M1 and horizontal lines are M2 segment.

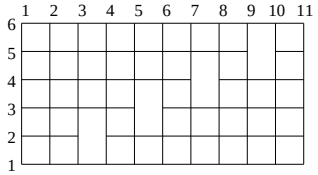


Figure 7: Wire segmentation scheme (for a 10-VCC)

THEOREM 3. *Any n-VCC ($n > 1$) implementable function can be realized by an n-VCC with segmentation scheme shown in Figure 7 when the number of M2 rows for N(P)-network equals $\lfloor (n-1)/2 \rfloor + 2$.*

Proof: Omitted due to page limit.

2.2.4 Multiple gate implementation

The n-VCC is the only combinational cell in our fabric. It can be split into two parts in different ways and used as two gates. We separate the transistors by dividing them into two halves and tapping the diffusion abutting points at the common Vdd (ground). The wire segmentations shown in Section 2.2.3 guarantee that the functions of the two parts can be realized independently.

COROLLARY 2. *Any i-VCC implementable function ($1 < i < n$) and $(n-i)$-VCC implementable function can be realized by an n-VCC with the segmentation scheme in Figure 7 if they can share the common ground and Vdd at the middle abutting point.*

Proof: Omitted due to page limit.

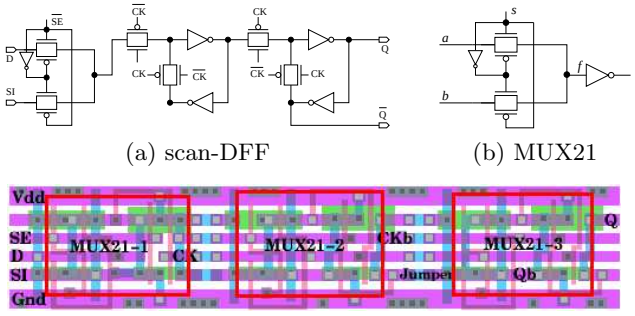
We can decompose a VCC only into two parts. Decomposition into more than two parts requires that the middle parts share Vdd and ground on both sides, which usually makes the function of the middle gate unrealizable.

One extra M1 wire is required to connect the N- and P-network outputs together if VCC is used as one gate, and two are needed when it is used as two gates. For this reason, in our fabric we provide two extra M1 wires.

3. VIA-DECOMPOSABLE D-FFS

In our via-configurable fabric, we propose a decomposable memory element for more flexible use. Latches and flip-flops are generally constructed from simple gates. By making the connections between those internal gates configurable (through vias), they can be used interchangeably as combinational gates or as memory elements.

Figure 8 shows an example. A scan-DFF is decomposed into three configurable 2-1 multiplexors. We also show the layout of a scan-DFF assembled from three 2-1 multiplexers using jumpers and vias.



(a) scan-DFF (b) MUX21

(c) Layout of scan-DFF by three MUX21s

Figure 8: Via-decomposable scan-DFF

The general function of a 2-1 multiplexor (Figure 8(b)) can be written as

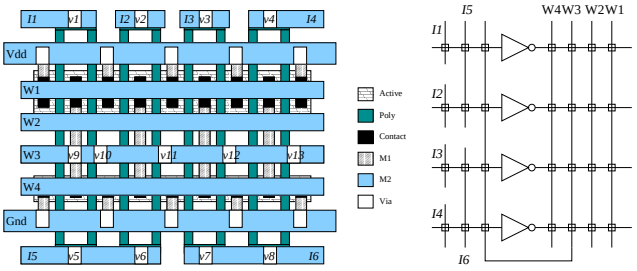
$$f(s,a,b) = sa + \bar{s}b$$

For example, by selecting via configurations and input assignments, we can use it to implement the following functions:

$$\begin{aligned}
f(x,y) &= xy + \bar{x}\bar{y} && (s \to x, a \to y, b \to \bar{y}) \\
f(x,y) &= x\bar{y} + \bar{x}y && (s \to x, a \to \bar{y}, b \to y) \\
f(x,y) &= xy && (s \to x, a \to y, b \to 0) \\
f(x,y) &= x\bar{y} && (s \to x, a \to \bar{y}, b \to 0) \\
f(x,y) &= \bar{x}\bar{y} = \overline{x+y} && (s \to \bar{x}, a \to \bar{y}, b \to 0)
\end{aligned}$$

4. VIA-CONFIGURED SIZABLE INVERTERS

In each fabric block we include a via-configurable inverter array containing a set of inverters which can be used separately or can be combined to form a larger one. Figure 9 shows an array with four inverters. The poly loop structure facilitates the Vdd and ground sharing between inverters and enables all inverters to be laid out using one diffusion strip. By placing vias v_1 – v_4, we have four individual inverters; segments W1 – W4 provide separate outputs. By placing vias v_5 and v_6, or v_7 and v_8, we obtain a size 2X inverter. By placing vias v_{10} – v_{13}, 2∼4 inverters can be connected to form a 2∼4X larger inverter. Moreover, two inverters can be cascaded to form a buffer, by placing via v_9 and one of v_{10} – v_{13}.



(a) layout model (b) schematic

Figure 9: Via-configurable repeater array

The number of inverters required in each block can be estimated based on wiring complexity metrics, such as Rent's exponent. We suggest a family of fabrics with different block compositions for different application needs.

5. DESIGN FLOW OF REGULAR FABRICS

Currently, we don't have any specific synthesis tool for our via-configurable fabrics. We use standard-cell tools to perform synthesis. We first enumerate all the functions an n-VCC can realize and build a standard-cell library. After technology mapping the circuits, we pack them into n-VCCs and then into blocks. A block-level placement and routing are then performed.

6. EXPERIMENTS AND DISCUSSIONS

We performed experiments on 20 largest MCNC benchmarks ranging from 1.5k to 17k nodes. We compare our VCC-based fabric with LUT- and PLA-based fabrics. For a fair comparison, we first conduct experiments to find the best block composition for each fabric type.

6.1 Experimented fabric types

PLA-based fabric. We used dynamic PLAs [4]. The area and delay of a PLA can be calculated based on the number of inputs, outputs and terms [9]. A PLA-based fabric is composed of equal-sized cells. The common PLA optimization techniques, such as folding and sharing, cannot be used in a partially prefabricated fabric.

Since parameters of the basic PLA (number of inputs, outputs, terms) have significant effect on the circuit area and timing, we try different depth-numbers and PLA parameters, selecting for comparisons the configurations which achieve the best area-delay product.

For conservative comparison, we used the best PLA-cells for each circuit, allowing D-FFs to be placed and routed anywhere and used only as required. The results for PLAs are reported after place and route.

LUT-based fabric. In a LUT-based fabric, each combinational core cell is a n-input LUT (n-LUT). We adopted a via-programmable version of a LUT from [11]. The entire LUT is designed with M1 only except for M2 wires providing via configurability.

We experimented with a 3-LUT-based and 4-LUT-based fabric with various block compositions. We found that the block which contains five 4-LUTs and one D-FF gives the best area-delay product. We use that block for comparisons.

3-LUT/NAND3 hybrid structure based fabric. Pileggi *et al* [11] have proposed a hybrid structure with two 3-input NAND gates, one 3-LUT and one D-FF as a basic block of a fabric. We refer to it as CMU.

VCC-based fabric. We experimented with 4-VCCs, 5-VCCs and 6-VCCs. We found that the block with four 5-VCCs and one D-FF gives the best results. We use it in later comparisons. The configurable D-FF is about 25% larger than D-FFs used by other fabrics.

6.2 Fabric comparison

The fabrics consist of duplicated same-type blocks. Table 3 lists their compositions. The metal patterns inside each block are fixed and those between blocks are determined by a standard-cell router.

Fabric	Block composition	Mapping
PLA-based	1 basic PLA	espresso[12]
LUT-based	5 4-LUT/1 DFF	FlowMap[3]
CMU	1 3-LUT/2 NAND3/1 DFF/7 INV	SIS map[12]
VCC-based	4 5-VCC/1 DFF/4 INV	SIS map[12]

Table 3: Configurations for different fabric types

For each fabric type we invoked the same flow (Section 5) (with different mapping tools (Table 3)) and evaluate the area, timing and power consumption after global routing. For reference, we also mapped the circuits to standard cells. We use `script.delay` in SIS [12] for delay optimization and a commercial 0.18μm standard-cell library of 280 cells with rich sizes for each type of cell. We placed the circuits using Capo [2] and routed with a maze router. The experiments were conducted on a PC with Intel P4 2.0GHz running Linux.

Table 4 lists the area/timing/power comparisons for four fabric types. The data have been normalized to standard-cell results. "#Nodes" is the circuit size after standard-cell mapping. We run logic simulation to obtain transition density of each node, reporting only the dynamic power. We also list the transistor utilization which shows packing efficiency for each fabric.

PLA-based fabric is more than 10 times larger, and consumes 3 times more power compared to standard-cell design. Although PLAs are faster if the wire delays between them are not considered, this advantage is lost after layout.

LUT-based fabric is much more effective in packing circuits than the PLA-based fabric. But it still needs three times more area than standard-cell design. The LUT cell is relatively large, and many D-FFs are wasted due to the fixed structure (only 35% of D-FF utilization on average). The performance is in fact, more than twice as bad than that of standard cell style.

Circuit	# Nodes	# FFs	Area				Delay				Power consumption				Transistor utilization (%)			
			PLA	LUT	CMU	VCC	PLA	LUT	CMU	VCC	PLA	LUT	CMU	VCC	PLA	LUT	CMU	VCC
alu4	2951	0	6.79	2.48	4.12	1.65	1.28	1.65	1.67	1.17	0.90	1.35	1.34	1.10	13.8	80.3	23.8	97.6
apex2	3874	0	7.72	2.37	3.68	1.53	1.90	2.45	1.92	1.31	1.36	1.07	1.19	1.01	14.9	80.3	23.8	97.8
apex4	1970	0	8.42	3.13	3.76	1.59	2.31	1.99	1.95	1.21	2.37	1.81	1.04	0.94	18.7	80.4	23.8	95.1
bigkey	3912	224	5.72	1.98	2.86	1.27	3.43	4.87	1.56	1.19	3.47	1.78	1.04	0.83	17.8	86.7	56.7	97.3
clma	17062	33	22.71	3.66	4.60	1.76	4.48	3.36	2.45	1.25	2.85	4.19	1.48	1.47	8.2	80.6	24.9	96.3
des	2716	0	9.00	3.13	4.45	1.66	1.66	3.68	1.77	1.08	2.75	1.65	1.15	0.93	17.2	80.4	23.8	94.2
diffeq	2567	377	10.72	2.78	2.76	2.12	2.35	1.82	2.30	1.52	3.61	5.87	0.93	1.67	10.2	76.2	72.4	72.7
dsip	2601	224	4.97	1.63	2.58	1.27	5.82	5.18	1.53	1.16	2.18	1.51	0.99	0.74	23.4	96.5	72.3	91.6
elliptic	6047	1122	8.10	3.06	2.58	2.48	1.95	2.47	1.96	1.38	2.71	3.82	1.16	0.48	11.1	72.0	73.6	65.2
ex1010	4405	0	11.81	3.87	4.17	1.71	2.36	3.76	2.00	1.32	2.46	1.85	1.30	0.98	10.0	79.9	24.3	90.2
ex5p	1489	0	10.74	3.29	3.88	1.62	1.55	1.97	1.79	1.30	3.36	2.22	1.15	0.97	18.4	80.1	24.1	97.9
frisc	5359	886	14.72	2.82	3.04	2.34	1.97	2.16	2.00	1.21	4.81	4.39	2.24	1.76	7.9	74.2	72.9	72.5
misex3	2950	0	8.83	2.17	3.50	1.53	1.21	1.23	1.26	0.91	1.16	1.07	1.13	1.01	13.0	80.2	24.1	95.0
pdc	3597	0	16.73	3.99	4.32	1.75	4.03	3.01	2.24	1.34	3.14	1.69	1.37	1.09	11.7	80.3	24.1	96.2
s298	1461	8	26.16	4.42	4.33	1.75	1.89	2.76	1.99	1.26	13.52	10.25	1.10	0.91	7.3	81.0	25.6	92.3
s38417	14362	1463	11.10	2.48	3.19	2.00	3.85	4.32	2.44	1.72	4.07	4.30	1.29	1.23	11.4	79.4	62.2	84.6
s38584.1	9387	1260	8.88	2.77	2.96	2.11	3.17	2.21	2.28	1.34	5.17	4.91	1.36	1.29	13.1	81.6	69.6	76.9
seq	3593	0	6.73	2.29	3.61	1.52	1.29	2.15	1.84	1.27	1.83	1.21	1.26	1.06	17.4	80.3	23.8	97.9
spla	2336	0	15.01	4.16	4.36	1.72	3.46	2.84	2.21	1.22	2.57	1.86	1.22	0.99	15.5	80.3	24.7	91.5
tseng	2186	382	10.00	3.06	2.54	2.40	2.23	1.91	1.63	1.08	9.44	9.26	4.26	1.02	9.8	64.0	73.8	66.6
AVG			11.24	2.98	3.56	1.79	2.61	2.79	1.94	1.26	3.69	3.30	1.40	1.07	13.5	79.7	42.2	88.5

Table 4: Circuit performance/area/power and transistor utilization with different fabrics

The CMU-hybrid structure has timing and power advantages over the LUT-based design. It is so because the 3-input NAND gates are extensively used. Because LUTs are inefficiently used by a mapper due to their inferior performance and large area, and because D-FFs are not utilized efficiently, designs mapped to this fabric tend to have large area.

VCC-based fabric performs well in all three metrics. Compared to standard-cell design, the area increases by 79%. This is caused by the larger core cell, which is designed to provide as much configurability as possible. Due to the strong functionality implementation capability and decomposability of D-FFs, the packing efficiency is very high. In most cases, more than 95% of VCCs and more than 85% of D-FFs are utilized. On average, the VCC structure shows only 26% performance degradation and consumes 7% more power than the standard-cell design. A better synthesis and packing algorithm could provide some improvement.

7. CONCLUSIONS AND FUTURE WORK

We have developed a design methodology of via-configurable functional cells. In our VCC, the contacts, transistors, M1 and M2, have very regular patterns, and are manufacturability-friendly. We also propose a via-configurable repeater array and D-FF for complete via-configurable regular fabric. Our fabric has performance and area comparable to standard-cell implementations.

In the future, we will develop a flow to directly synthesize circuits into VCCs and use more complex functions to improve circuit performance. We will also explore the inter-cell routing structures and investigate the probability of fixing more metal layers to relieve design efforts and to lower the design costs.

8. ACKNOWLEDGMENTS

This work was supported in part by NSF through CCR grant 0098069 and in part by MICRO through IBM. The authors also acknowledge the Intel equipment grant.

9. REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection produce routable placements? In *Proceedings of DAC*, pages 260–263, 2000.

[3] J. Cong and Y. Ding. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. on CAD*, 13(1):1–12, January 1994.

[4] Y. B. Dhong and C. P. Tsang. High speed CMOS POS PLA using predischarged OR array and charge sharing AND array. *IEEE Trans. on Circuits and Systems II*, 39(8):557–564, August 1992.

[5] S. P. Khatri et al. Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric. In *Proceedings of ICCAD*, pages 412–418, 2000.

[6] Z. A. Lomnicki. Two-terminal series-parallel networks. *Adv. in Appl. Prob.*, 4(1):109–150, April 1972.

[7] R. L. Maziasz and J. P. Hayes. *Layout Minimization of CMOS Cells*. Kluwer Academic Publishers, 1992.

[8] F. Mo and R. K. Brayton. River PLAs: A regular circuit structure. In *Proceedings of DAC*, pages 201–206, 2002.

[9] F. Mo and R. K. Brayton. Whirlpool PLAs: A regular logic structure and their synthesis. In *Proceedings of ICCAD*, pages 543–550, 2002.

[10] C. Patel, A. Cozzie, H. Schmit, and L. Pileggi. An architectural exploration of via patterned gate arrays. In *Proceedings of ISPD*, pages 184–189, 2003.

[11] L. Pileggi et al. Exploring regular fabrics to optimize the performance-cost trade-off. In *Proceedings of DAC*, pages 782–787, 2003.

[12] E. Sentovich et al. SIS: A system for sequential circuit analysis. Technical Report UCB/ERL M92/41, University of California, Berkeley, 1992.

[13] T. Uehara and W. M. van Cleemput. Optimal layout of CMOS functional arrays. *IEEE Trans. on Computers*, 30(5):305–312, May 1981.

[14] B. Zahiri. Structured ASICs: Opportunities and challenges. In *Proceedings of ICCD*, pages 404–409, 2003.