Research Article

Impact of Dual Placement and Routing on WDDL Netlist Security in FPGA

Emna Amouri,¹ Habib Mehrez,¹ and Zied Marrakchi²

¹ LIP6, Université Pierre et Marie Curie, 4 Place Jussieu, 75252 Paris, France

² Flexras Technologies, 153 Boulevard Anatole France, 93200 Saint-Denis, France

Correspondence should be addressed to Emna Amouri; amouriemna@yahoo.fr

Received 28 June 2013; Revised 11 October 2013; Accepted 16 October 2013

Academic Editor: Nadia Nedjah

Copyright © 2013 Emna Amouri et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The wave dynamic differential logic (WDDL) has been identified as a promising countermeasure to increase the robustness of cryptographic devices against differential power attacks (DPA). However, to guarantee the effectiveness of WDDL technique, the routing in both the direct and complementary paths must be balanced. This paper tackles the problem of unbalance of dual-rail signals in WDDL design. We describe placement techniques suitable for tree-based and mesh-based FPGAs and quantify the gain they confer. Then, we introduce a timing-balance-driven routing algorithm which is architecture independent. Our placement and routing techniques proved to be very promising. In fact, they achieve a gain of 95%, 93%, and 85% in delay balance in tree-based, simple mesh, and cluster-based mesh architectures, respectively. To reduce further the switch and delay unbalance in Mesh architecture, we propose a differential pair routing algorithm that is specific to cluster-based mesh architecture. It achieves perfectly balanced routed signals in terms of wire length and switch number.

1. Introduction

FPGAs are an attractive platform for cryptographic applications due to their low cost compared to full custom ASIC design and their short time to market period. In addition, their reprogrammability allows upgrading easily the cryptographic algorithm. However, unprotected hardware implementations are vulnerable to side channel attacks (SCA). It has been shown that differential power analysis (DPA) attack [1] is very powerful. DPA is capable of revealing the secret key by measuring power consumption leaked by a cryptographic device.

During the last years, many countermeasures have been proposed to protect cryptographic devices against SCA. They fall into two main categories: the masking logic and the hiding logic.

The principle of masking logic is to randomize the power consumption by using a random mask and thus decorrelate the intermediate data from the circuit power consumption. This technique was introduced first at algorithmic level [2] and then at gate level [3]. It has been shown that this technique can be broken by attacks based on probability

density function (PDF) [4] or glitches [5]. To overcome glitch problem, masked dual rail precharge logic (MDPL) [6] has been proposed. It merges masking with dual rail dynamic logic. However, MDPL shows a high area overhead [7].

On the other side, the principle of hiding logic consists in consuming the same amount of power consumption regardless of data inputs. This is achieved by using differential logic (signals are encoded as two complementary wires) and precharging the differential signals in every clock cycle. It is also called dual rail precharge logic (DPL). Several implementations of secure dual rail cells have been proposed, specifically for ASICs, such as SABL [8], WDDL [9], STTL [10], and MDPL [6].

The wave dynamic dual rail logic (WDDL) technique, developed by Tiri and Verbauwhede [9], is the most popular DPL countermeasure. It is based on a standard cell flow, and it is the most suited for FPGA implementation. However, this technique has been proved to prevent DPA, provided the routing of differential signals is balanced [11]. This task is hard to achieve in FPGA because routing resources are limited and have thus to be properly shared between differential components of the design.

To address the routing problem, Yu and Schaumont [12] suggest to implement a second complementary WDDL module on the FPGA with identical routing to the direct WDDL circuit. However, this technique causes a fourfold area increase and presents weakness against the Hamming distance (HD) model. McEvoy et al. [13] proposes Isolated WDDL to solve this weakness. However, the area increase resulting from this method is comparable to Schaumont technique. To decrease the area increase of Schaumont method, Baddam and Zwolinski [14] presented a design technique that separates the true part from the false part of the design by implementing an inverter using XOR gate. This technique may be sensitive to attacks based on glitches. In [15], authors propose a constrained placement technique in Xilinx and Altera FPGAs to improve dual signals timing balance, without increasing the design area. But current commercial routing software does not support balanced routing of differential nets. In [16] authors propose to achieve a reduction of fanout and number of gates in timing path as a solution to counter routing imbalance. However, this technique causes an area overhead.

There are some other proposed logic-level styles derived from WDDL technique, such as IMDPL [17], DRSL [18], STTL [10], SecLib [19], WDDL with or without early evaluation [20], and BCDL [21]. All these DPL logic styles can be implemented in FPGA but are also concerned with the problem of dual rail balancing.

In this paper, we deal with the problem of delay unbalance of dual signals, in WDDL design, without adding logic to balance dual networks to avoid area increase. We study the impact of different placement techniques on differential timing, and we propose a balanced-timing routing algorithm to balance propagation delays of dual signals. FPGAs we are targeting are a tree-based FPGA (MFPGA) [22], a simple mesh-based FPGA [23], and a cluster-based mesh FPGA. In addition, we propose a differential pair routing algorithm suitable for the cluster-based mesh architecture to improve the delay balance.

The rest of the paper is organized as follows. In Section 2 we present the architecture overview of FPGAs we are targeting. In Section 3 we present the WDDL features, and we explain the method used to analyze the delay differences in dual rail logic design. Then in Section 4, we evaluate the dual rail routing unbalance for different placement techniques in mesh- and tree-based FPGAs. In Section 5, we propose an architecture independent router which is based on the PathFinder router and whose objective is to optimize delay balance of dual nets, and we study its impact on differential propagation delays in tree-based FPGA, simple mesh-based FPGA, and cluster-based mesh FPGA. Then, to reduce delay unbalance and switch unbalance in cluster-based mesh FPGA, we propose in Section 6 a differential pair routing method. Finally, Section 7 concludes this paper.

2. FPGA Architectures

2.1. Mesh-Based FPGA. We consider an island style FPGA [23]. It contains configurable logic blocks (CLB) placed into

a regular 2-dimensional grid. Each CLB consists of a 4-input look-up-table (LUT) and a flip-flop (FF). A CLB is surrounded by unidirectional routing network [24]. The channel width is varied according to netlist requirement but remains in multiples of 2 [24].

A unidirectional disjoint switch box connects different tracks (or wires) of vertical and horizontal channels together. Each input pin of a CLB is connected to the adjacent routing channel, and the output pin connects with the routing channel on its top and right through the diagonal connections of the switch box (highlighted in the bottom-left switch box shown in Figure 1). The fraction of tracks that an input of a CLB is connected to and that is connected to the output, referred to as $Fc(in)$ and $Fc(out)$, is set to be maximum at 1.0.

In this work, we assume wires are of logical length $L = 1$, meaning they span 1 logic block (LB). As the LBs are all assumed to be identical, a single LB and its adjacent routing channels can be combined to form a tile that can be replicated to create the full FPGA. A tile is shown in Figure 1.

2.2. Multilevel Hierarchical FPGA

2.2.1. MFPGA Architecture Overview. A first multilevel hierarchical FPGA architecture (MFPGA) was designed and evaluated in [22]. Experimental results show that MFPGA implements circuits with an average gain of 40% in total area compared to mesh architecture [23]. MFPGA architecture shown in Figure 2 has a tree-based topology whose leaves are logic blocks (LBs). It has linear populated switch boxes and unidirectional wires. This architecture unifies two unidirectional connected networks.

- (i) The downward network, based on the butterfly fat tree (BFT) topology and involving a logarithmic population growth of unidirectional downward switch boxes (DS), connects these switch boxes to LBs inputs.
- (ii) The upward network comprises upward switch boxes (US). These USs allow LBs outputs to reach all DSs at different levels of the architecture.

Figure 2 shows MFPGA architecture with 2 levels and *Arity* 4 (each cluster contains 4 clusters).

2.2.2. MFPGA Layout. This section presents the method used for creating the MFPGA layout. In Figure 3, we show the floorplan of a $level_2$ cluster of the MFPGA, which is topologically equivalent to the MFPGA view presented in Figure 2. The floorplan shows a regular structure based on tiles, with each one including logic block, a set of $level_1$ switches, and a set of $level_2$ switches. The tile is replicated on column to form a cluster of $level_1$; then this cluster is replicated on rows to form cluster of $level_2$. By replicating tiles and $level_1$ cluster, we can increase the arity of the MFPGA in $level_1$ and $level_2$, respectively. In the same way we can build the next levels of hierarchy.

Figure 4 shows a representative chip of MFPGA architecture with 4 levels of hierarchy and arity $8 \times 8 \times 8 \times 4$ (2048 LBs).

FIGURE 1: Mesh FPGA.

FIGURE 2: MFPGA architecture with 2 levels of hierarchy (4×4).

3. Differential Timing Analysis in WDDL Design

3.1. WDDL Technique. In this paper, we are concerned with the WDDL strategy as a countermeasure against DPA. WDDL is characterized by the following features.

- (i) The netlist is duplicated into a true and a false part. For every logic gate, a complementary gate exists. This latter expresses the complementary (false) output of the direct (true) gate using the complementary inputs of the direct gate. Figure 5 shows the basic WDDL gates.
- (ii) The calculation is composed of two phases: when the clock is high, the precharge operation is performed,

in which all signals are reset. When the clock becomes low, the evaluation phase is achieved, and exactly one of the two dual outputs is evaluated to "1."

- (iii) If any direct gate switches to high, the dual gate does not and vice versa. Therefore, the activity of the circuit is constant regardless of the values of input data.
- (iv) The components used are limited to positive logic (AND and OR gates), and inverters are implemented by cross coupling complementary outputs. This allows the precharge wave propagation throughout the combinatorial gates.

The constant activity is a necessary condition to achieve constant power consumption, but it is not sufficient. To

FIGURE 3: Floorplan of MFPGA architecture (4×4).

FIGURE 4: MFPGA Chip (2048 LBs).

guarantee the robustness of WDDL technique, dual signals must be balanced which means the following:

- (i) match source-sink delay: the delay from A to I_i must be equal to the delay from A' to I_i' for each $i = 1, \dots, k$ (cf. Figure 6);
- (ii) match load capacitances: $C_A = C_A'$ at the differential outputs of the logic gates (cf. Figure 6). This load is dominated by the capacitance associated with

FIGURE 5: WDDL components.

the routing between cells. Hence, matching the interconnect capacitances of the differential signal wires is crucial for the countermeasure to succeed.

So, special constraints on placement and routing tools must be applied in order to balance the direct and complementary networks.

3.2. Delay Model. To estimate the complementary networks balance, we use the following metrics.

FIGURE 6: Differential load capacitance.

- (i) We compute the number of mismatched (unbalanced) signals (or connections) in terms of number of switches used for routing, called here *Sw.Mis.Signals*.
- (ii) We compute for all dual *source_sink* pairs the absolute difference $\Delta delay = |delay(true) - delay(false)|$, where $delay(true)$ and $delay(false)$ are interconnect latencies of true and false signals, respectively.

Propagation delay is computed using the Elmore delay model [25] for the ST 0.12 μ process. Even if the Elmore delay is a simplified model, it is quite accurate, can be computed in linear time, and allows us to compare various algorithms to choose the most efficient approach.

Figure 7 shows the RC-model for FPGA routing elements. The buffer is modelled by a constant delay, a resistor, and an input capacitance [26]. The constant delay is the intrinsic delay of the buffer. The pass transistor is modelled by a linear resistor and a capacitance [26]. These parameters are obtained by running many simulations with Eldo. The wire is modelled by a resistance and a capacitance that depend on the wire length and technological parameters.

The Elmore delay of a (*source* $\rightarrow$ *sink*) path is then as follows:

$$Delay_{Elmore} = \sum_{i \in source\_sink\_path} (R_i \cdot C(Sub\_tree_i) + T_{d,i}), \quad (1)$$

where $T_{d,i}$ is the intrinsic delay of a buffer if element i is a buffer (0 otherwise). R_i is the equivalent resistance of element i . It can be the wire resistance, the buffer resistance, or the pass transistor resistance. In (1), $C(Sub\_Tree_i)$ is the lumped capacitance of element i . It is the total capacitance

FIGURE 7: RC model for FPGA routing elements.

FIGURE 8: RC tree network.

of the subtree rooted at element i , that is, the downstream capacitance which is not isolated by buffers.

After routing is completed, our router builds an equivalent RC tree for each net. The RC tree is a set of interconnected resistors with capacitors from each node in the network to ground. In Figure 8, we show the RC tree corresponding to the highlighted path in the crossbar. R_w , C_w , R_{sw} , C_{sw} , and C_{in} represent the wire resistance, wire capacitance, switch resistance, switch capacitance, and the capacitance of the sink buffer, respectively. T_d and R_d are the intrinsic delay and the resistance of the driver.

To determine the resistance and capacitance of routing wires, we need to know how long these wires are. Before starting the routing process, our router builds the routing graph of the FPGA architecture and calculates lengths of all wires based on layout regularity. For example, in the multilevel FPGA, the wire length depends on its level in the architecture, its direction (downward or upward), its source, its destination, and the arity of the architecture. After routing and building the RC tree of each net, the router computes the propagation delay between the source node and each destination node, using the Elmore delay.

In the rest of the paper, we present a case study on the DES [27] cryptographic algorithm, which is a popular symmetric cryptographic algorithm that is widely used, and some of the Altera QUIP netlists implemented in WDDL [28]. Table 1 presents the characteristics of the WDDL netlists we use.

TABLE 1: Characteristics of WDDL designs.

WDDL Benchs	4-LUTs	IN	OUT	Functionality
DES	2152	10	19	Cryptographic design
BarreI16	586	42	34	Shift register
BarreI32	1406	76	66	Shift register
BarreI64	3112	143	130	Shift register
Mux8	2966	22	130	Bus of muxes
Mux32	2888	74	34	Bus of muxes
xbar ₁₆ × 16	546	160	34	Crossbar

4. Impact of Dual Placement in FPGA

4.1. Dual Placement in Mesh FPGA. Our placement tool uses the simulated annealing algorithm [23] to place the CLBs/IOs instances of the netlist on the CLBs/IOs blocks of FPGA. The goal of the placer is to minimize the sum of half-perimeter of the bounding boxes (BBX) of all the nets. The BBX of a net is a minimum rectangle that contains the driver instance and all the receiving instances of a net. The placer moves an instance randomly from one block position to another. After each move operation the BBX cost is updated incrementally. Depending on the cost value and the annealing temperature, the move operation is accepted or rejected. After placement, the router tool uses the pathfinder algorithm [29] to route all the netlist nets on the FPGA routing resources.

To improve dual rail unbalance in WDDL design, we investigate two placement techniques.

4.1.1. Symmetrical Placement. The mesh architecture presents a homogeneous interconnect. So, we can divide the architecture into two separate and symmetrical domains which present identical routing resources. We wanted to exploit this symmetry to balance the routing of the differential netlist by performing a symmetrical placement.

For this purpose, we started by placing the direct network on the half of the mesh architecture. Then we placed symmetrically the complementary network on the second half of the architecture, as shown in Figure 9.

Table 2 shows the statistics obtained for the WDDL cryptographic DES netlist and for some WDDL QUIP netlists. Results correspond to the case where netlists are placed without constraint and the case where they are placed symmetrically. We can notice from these results that the dual placement statistics are improved. In fact, the average $\Delta delay$ is improved by 56%. In addition, the number of mismatched (unbalanced) connections in terms of number of switches used for routing, called here *Switch Mismatch*, is decreased from 2185 to 1168. However, the unbalance is still important.

First, as already explained in Section 3.1, in WDDL netlists, inverters are removed and replaced by cross coupling complementary outputs. So, the dual subnetlists are not independent. Thus, connections between direct and dual gates, which are placed in separate domains, cannot be routed symmetrically. In addition, we have constrained the placer to place the direct network in half of the architecture, and thus

FIGURE 9: Symmetrical placement in mesh FPGA.

reduce the search space. So, the placement quality decreases and may cause long nets and more congestion in the routing.

4.1.2. Adjacent Placement. Keeping the direct and dual gates in adjacent places in mesh architecture can be favorable to obtain nets as symmetrical as possible. Besides, current techniques to control routing in ASICs choose to keep dual gates close to each other.

We choose to place each true instance above the complementary instance. When an instance is moved to a new block position, the dual instance must be moved to the adjacent block position. This is achieved for true and false instances. Figure 10 shows a result of the adjacent placement.

With this constraint, the balance between dual nets greatly improves. The last four columns of Table 2 show a gain of 80% of the average $\Delta delay$ in WDDL DES netlist and 78% in WDDL QUIP netlists. In the WDDL DES design, the average gate to gate skew between dual gates is about 121 ps, whereas it is about 607 ps without constrained placement. In addition, the number of switch-unbalanced connections is decreased by 50%. So, it can be seen from these results that the adjacent placement is more efficient than the symmetrical placement in the Mesh FPGA.

4.2. Dual Placement in MFPGA. The MFPGA configuration flow begins by a top-down recursive multilevel partitioning phase. The multilevel partitioning algorithm [30] consists of 3 phases. First, during the coarsening phase, the size of the hypergraph is successively decreased. Next, during the initial partitioning phase, a k-way partitioning of the smaller hypergraph is computed. Finally, during the uncoarsening phase, the partitioning is successively uncoarsened and refined using the FM [31] refinement algorithm with an objective of reducing external communication.

In the MFPGA partitioning, first, the top level clusters of the tree-based architecture are constructed; then each

TABLE 2: Placement results ($\Delta delay$) of WDDL netlists in mesh FPGA.

WDDL design	Nets	Unconstrained placement				Symmetrical placement				Adjacent placement			
		Max (ps)	Mean (ps)	Std. Dev. (ps)	Switch mismatch	Max (ps)	Mean (ps)	Std. Dev. (ps)	Switch mismatch	Max (ps)	Mean (ps)	Std. Dev. (ps)	Switch mismatch
DES	2162	5317	607	746	2375	4623	289	551	1431	1989	121	176	1476
Barrel16	628	2323	458	466	815	1480	164	229	532	2448	165	299	479
Barrel32	1482	6825	829	908	2072	5504	333	648	1266	5140	222	440	1247
Barrel64	3254	8351	1114	1229	4760	10330	674	1269	3158	7090	252	545	2841
Mux8.64 bit	2988	7023	579	939	2242	6263	230	645	676	2960	76	255	553
Mux32.16 bit	3072	3683	447	536	2301	4271	108	353	630	1702	58	147	620
xbar.16 × 16	706	5745	634	850	732	3706	258	375	485	3013	92	195	384
Average	2041	5752	666	810	2185	5168	293	581	1168	3477	140	293	1085

FIGURE 10: Adjacent placement in mesh FPGA.

cluster is partitioned into subclusters until the lowest level of the architecture is reached. For example, we run a recursive netlist partitioning for a $4 \times 2 \times 4$ MFPGA architecture. This architecture has 3 hierarchical levels, has an arity of 4 in levels 1 and 3, and an arity of 2 in level 2. The netlist is first partitioned into 4 parts. Then instances inside each part are partitioned into 2 fractions. The maximal number of instances in each fraction is equal to 4 since the arity in level 0 is equal to 4. In each partitioning phase we apply a multilevel coarsening followed by a multilevel refinement. Finally, we obtain how instances are distributed between clusters of the tree-based topology.

After the partitioning, we run the placement phase. Each cluster is assigned to a random position inside its owner cluster since clusters positions are equivalent inside the same cluster. After placement, the router assigns nets that connect placed instances to routing resources in the interconnect structure, using the PathFinder routing algorithm [29].

As for the mesh-based FPGA, we investigate symmetrical and adjacent placement techniques in MFPGA.

4.2.1. Symmetrical Placement. The MFPGA architecture presents an interesting symmetry. Indeed, all the clusters of a same level are identical, such as $level_1$ clusters $Cluster(1, 0)$ and $Cluster(1, 4)$ shown in Figure 11. Thus, symmetrical wires have the same length. This is the case of green and red wires belonging, respectively, to symmetrical clusters $Cluster(1, 0)$ and $Cluster(1, 4)$. So, we perform a symmetrical placement, as shown in Figure 11, so that direct and dual subnetlists can have exactly the same routing resources.

Figure 12 shows the different steps of symmetrical recursive netlist partitioning based on a multilevel approach. The WDDL netlist is first partitioned into 2 parts: a partition that contains true gates and a partition that contains dual gates. After clustering according to this partitioning, we obtain 2 clusters containing real and dual networks, respectively, and

called *TopClusterTrue* and *TopClusterFalse*, respectively. In the example shown in Figure 12, the MFPGA architecture has $4 \times 2 \times 4$ arity.

To obtain a symmetrical placement in MFPGA architecture, we have to perform identical partitioning for the two complementary networks. For this purpose, we start by partitioning the real network. Only the hypergraph representing real gates is considered. We perform a recursive partitioning based on a multilevel approach to this hypergraph. In each partitioning phase we apply a multilevel coarsening followed by a multilevel refinement.

In this partitioning phase, only the half of the MFPGA architecture is considered. In this example, the half of MFPGA corresponds to a $2 \times 2 \times 4$ architecture. The second half of the architecture is reserved to dual gates.

Once real gates contained in the *TopClusterTrue* cluster are partitioned, we partition the dual network but without using the multilevel partitioning approach. In this phase, the dual hypergraph is considered. This hypergraph is partitioned recursively at all hierarchical levels beyond the *TopClusterFalseLevel* level, starting by the level *TopClusterFalseLevel* - 1 to the level 1. To partition the hypergraph at level ℓ , we attribute to each dual cluster of level $\ell - 1$ the same partition index of the real cluster. Then, dual clusters are clustered according to the partitioning result, and thus we obtain new clusters at level ℓ . In this way, real and dual gates are partitioned in the same way at all hierarchical levels by restriction with clusters of the last level: *TopClusterTrue* and *TopClusterFalse*. Finally, we disaggregate the hypergraph to remove the level of *TopClusterTrue* and *TopClusterFalse*.

After partitioning, we place the direct network and then we place the dual network. We attribute randomly to every real cluster a position inside its owner. Then, we assign to each dual cluster the same position as the real cluster. This process is performed in all levels ℓ of the hypergraph; $1 \leq \ell < TopLevel - 2$.

FIGURE 11: Symmetrical placement in MFPGA architecture ($2 \times 4 \times 4$).

At level *TopLevel* – 2 of the hypergraph, we attribute to real clusters a random position c , $0 \leq c < k/2$, with k representing the last level arity of the MFPGA architecture we target. Then, we assign to each dual cluster a position c' , where $c' = c + (k/2)$ and c is the real cluster position. Thus, we obtain a symmetrical placement.

Table 3 summarizes the placement statistics obtained for the WDDL DES and QUIP netlists. The obtained results show that the dual balance statistics are improved. In fact, the number of unbalanced connections in terms of number of switches used for routing is decreased by 90%. In addition, the average $\Delta delay$ is improved by 50% in WDDL designs. However, we can see that the delay unbalance is still important (666 ps). This is due first to unbalanced connections in terms of switch number. Second, direct and dual subnetlists are not independent. Since, in the symmetrical placement, direct and dual gates are in two separate parts of the architecture, there will be a great number of connections between direct and dual gates which are not in the same cluster, and thus they must be routed via the upper level of the interconnect. So, these connections cannot be routed symmetrically. Moreover, in MFPGA architecture, the difference between lengths of wires becomes very important in the upper level. So, symmetrical placement is not the most suitable placement for the tree-based FPGA.

4.2.2. Adjacent Placement. Instead of separating direct and dual networks in two separate domains of the architecture,

we choose to keep real and dual gates in adjacent places in the architecture. In other words, we place 2 complementary gates in the same cluster of the MFPGA, as shown in Figure 13. At the end of the placement, each gate must have a vector of positions which describes the position of the gate and the positions of owner clusters in each level ℓ of the architecture. For example, in Figure 13, the gate B_T has a vector of positions $\{0, 1, 2\}$. This means that B_T is placed in position 2 in its owner cluster of level 1. Then, this latter cluster has position 1 inside its owner cluster of level 2. Finally, this latter cluster is placed at position 0. To have adjacent placement, dual gates must be placed in the same cluster at each level of the architecture. So, at the end of the placement, they must have vectors of positions in which the positions of clusters starting by level 1 to the upper level of the architecture are the same.

Figure 14 shows the different steps of adjacent recursive netlist partitioning based on a multilevel approach in order to perform an adjacent placement in MFPGA. In this example, we consider a $4 \times 2 \times 4$ MFPGA architecture. Algorithm 1 shows the pseudoalgorithm to perform the adjacent recursive partitioning. In the adjacent partitioning, the hypergraph of all the WDDL circuit is considered. To guarantee that dual gates will be placed in the same cluster in each level of the architecture, we start by grouping each pair of real and dual gates. To do this, we assign to each pair of complementary gates the same partition index, then we run a clustering phase according to this partitioning solution. So, we obtain a new hypergraph which contains new clusters at level 1.

FIGURE 12: Symmetrical partitioning steps in multilevel architecture $4 \times 2 \times 4$.

TABLE 3: Placement results ($\Delta delay$) of WDDL netlists in MFPGA.

WDDL design	Unconstrained placement				Symmetrical placement				Adjacent placement			
	Max (ps)	Mean (ps)	Std. Dev. (ps)	Switch mismatch	Max (ps)	Mean (ps)	Std. Dev. (ps)	Switch mismatch	Max (ps)	Mean (ps)	Std. Dev. (ps)	Switch mismatch
DES	7006	1420	1438	1131	6840	732	870	110	4106	365	429	34
Barrel16	3828	648	660	242	2505	375	312	21	1946	231	235	11
Barrel32	5317	1165	1094	961	3631	420	468	104	3054	330	426	90
Barrel64	9157	1664	1752	1368	7000	1128	933	145	6902	542	636	96
Mux8.64 bit	8940	2004	2205	1088	6303	813	843	79	5385	386	539	25
Mux32.16 bit	8834	1882	2056	956	6742	849	869	82	6458	401	560	34
xbar_16 × 16	3281	724	672	272	2258	347	249	8	1344	137	139	2
Average	6622	1358	1411	859	5039	666	649	78	4170	341	423	41

FIGURE 13: Adjacent placement in MFPGA architecture ($2 \times 4 \times 4$). FIGURE 14: Adjacent partitioning steps in multilevel architecture $4 \times 2 \times 4$.

```

Let: hypergraph = netlist;
    arch_levels = MFPGA levels number;
    Arity[lev] = Arity in MFPGA at level lev;
Phase 1: grouping pairs of dual gates;
Let: sub_hypergraph = hypergraph (level = 1);

phase 2:
for (int lev = arch_levels-1; lev >= 0; lev--) {
  if (lev = 0) {
    partitioning (sub_hypergraph, TopLevelCluster, Arity[lev]);
    // partition instances inside TopLevelCluster into k parts
    // K = Arity[lev]
  } else {
    for (each Cluster in level 1 of the sub_hypergraph) {
      partitioning (sub_hypergraph, Cluster, Arity[lev]);
      // partition instances inside Cluster into k parts
      // K = Arity[lev]
    }
  }
}

phase 3:
uncoarsening sub_hypergraph;

```

ALGORITHM 1: Pseudoalgorithm of adjacent partitioning.

These clusters have a size equal to 2, since each cluster contains 2 dual gates. After that, we perform a recursive partitioning for the tree-based architecture, without any constraint. The hypergraph considered in this phase corresponds to the hypergraph having leafs with a size equal to 2. These leaf clusters are highlighted in Figure 14. We call the hypergraph considered in this phase *sub_hypergraph*. In the recursive partitioning, first instances inside the *TopLevelCluster* are partitioned into *k* parts. *k* is the number of clusters at the upper level in the tree-based architecture (*k* equal to 4 for MFPGA $4 \times 2 \times 4$). Thus, we obtain *k* clusters at level 1 in the subhypergraph. Then, each cluster is partitioned into subclusters, until the lowest level of the architecture is reached. At the end of the recursive partitioning performed on the subhypergraph, leaf clusters grouping dual gates are uncoarsened to obtain the initial hypergraph partitioned. In this way, each real and dual gates are placed in the same cluster of the MFPGA architecture.

The last four columns of Table 3 show the statistics on dual rail unbalance for the adjacent placement. It can be seen from these results that the balance between dual nets greatly improves. In fact, we can see a gain of 74% of the average $\Delta delay$ in WDDL designs. The average gate to gate skew between dual gates is about 341 ps, whereas it is about 1358 ps without constrained placement, and the switch mismatch is decreased by 95%. Therefore, the adjacent placement is more efficient to improve the delay balance in MFPGA than the symmetrical placement, as for the mesh architecture.

5. Timing-Balance-Driven Routing

5.1. Timing-Balance-Driven Routing Algorithm. In this part, we present the enhancements we made to the PathFinder

router to improve the differential timing balance. We called the new router a timing-balance-driven router. Routing was performed on adjacent placed netlists.

The new router is based on congestion and delay-balance negotiation. Interconnect resources are presented by a routing graph with nodes corresponding to wires and CLBs pins and edges presenting switches.

Consider the connection to sink *j* of net *i*. Inspired from timing driven extension of PathFinder [23], we define the cost of using a routing resource *n* as

$$\begin{aligned}
 Cost(n) = & (1 - Crit(i, j)) \cdot Cong\_cost(n) \\
 & + Crit(i, j) \cdot Resource\_diff\_delay(i, n, j) \quad (2) \\
 & \cdot f(Diff\_sw\_nb(i, n, j)),
 \end{aligned}$$

where

$$f(Diff\_sw\_nb(i, n, j)) = \max(0.1, |Diff\_sw\_nb(i, n, j)|). \quad (3)$$

The first term in (2) is the congestion sensitive term, and the second term is the delay balance sensitive term. The congestion-delay tradeoff of each net connection is controlled by how critical it is. Before describing our routing algorithm, we define the different terms of (2).

- (1) *Cong_cost(n)* is the congestion cost of a routing resource *n*. It takes into account the number of nets sharing this resource, and the history of congestion on it [23].

(2) To explain $Crit(i, j)$, we define the following notations.

- (a) $Connection\_diff\_delay(i, j)$ is the timing unbalance between the routed connection to sink j of net i and the complementary routed connection to sink j' of dual net.
- (b) The $Max\_connection\_diff\_delay$ is the maximum $Connection\_diff\_delay(i, j)$ among all routed design connections.
- (c) The unbalance criticality of a sink j of net i , denoted as $Crit(i, j)$, can be formulated as

$$Crit(i, j) = \min \left(0.9, \frac{Connection\_diff\_delay(i, j)}{Max\_connection\_diff\_delay} \right). \quad (4)$$

The connection unbalance criticality is a fractional number between 0 and 1. High connection unbalance criticality means that the real and dual routed connections have an important delay unbalance. The connection unbalance criticality is related to the routing results of the previous iteration. In the first iteration, the $Max\_connection\_diff\_delay$ is set to 1 and all connections are critical ($Crit(i, j) = 0.9$).

(3) The $Diff\_sw\_nb(i, n, j)$ is the difference between the number of used switches in the routing of dual signals. We note that this difference has an important impact on delay unbalance. We define the following notations.

- (i) To determine the differential number of routing switches used, at a resource node n , we must estimate the minimal remaining switches number from the current node n to the target sink node j . We call that the expected switches number at a node n denoted as $Expected\_sw\_nb(n, j)$. Its calculation depends on the architecture: in mesh FPGA, we know the coordinates (x, y) of both sink j and the current node n , so we can compute the minimum number of switches needed to reach sink j . In tree FPGA, we can compute the level of the lowest common owner cluster of both sink j and node n , and thus we determine the number of levels to cross to route the connection to sink j .
- (ii) The estimated total switches number that will be used to reach the sink j of net i from the net driver and using the node n , denoted as $Estimated\_Total\_sw\_nb(i, n, j)$, is the sum of switches already used to reach the node n and $Expected\_sw\_nb(n, j)$.
- (iii) We note that $Diff\_sw\_nb(i, n, j)$ is the difference between $Estimated\_Total\_sw\_nb(i, n, j)$ and the number of routing switches used in the

dual connection j' already routed denoted as $Total\_sw\_nb(i', j')$:

$$Diff\_sw\_nb(i, n, j) = Estimated\_Total\_sw\_nb(i, n, j) - Total\_sw\_nb(i', j'). \quad (5)$$

If $Diff\_sw\_nb(i, n, j) = 0$, the second term in (2) should not be equal to 0, because balancing the number of switches does not mean balancing delays of wires. We have experimentally determined that setting the minimal $f(Diff\_sw\_nb(i, n, j))$ to 0.1 is the best value for WDDL designs mentioned in this paper.

- (4) The $Resource\_diff\_delay(i, n, j)$ is the difference between the delay of a resource n used to route a connection to sink j of net i and the delay of the equivalent resource n' used by the complementary connection. We note that the delay of a resource n is a function of both the resource n and the path used to reach it from the source. When the node n is reached, we obtain a subrouting tree (set of wires and switches). We need to compare its delay to the delay of the corresponding subrouting tree of the dual already routed connection. The corresponding subrouting tree is a set of routing resources starting from the dual net driver until the equivalent node n' of node n . To find the equivalent node n' , we must determine its index in the dual routing tree. We call the $index(n)$ of a node n its position in the routing tree. Algorithm 2 shows the pseudocode to calculate $index(n')$.

Figure 15 shows an example of a routing result of the dual connection j' already routed and the routing graph available to route the connection j currently being routed. Weights on edges correspond to the delays of routing resources (in ps).

In this expansion, we can distinguish 3 scenarios.

- (i) *Scenario 1.* Expanding to the node f_1 leads to a routing path to sink j with $Estimated\_Total\_sw\_nb(i, n, j)$ less than $Total\_sw\_nb(i', j')$ (case a). So, the equivalent node of f_1 is the node with the same $index$ as node f_1 and which is the node t_1 ($index = 1$). Then, the equivalent node of sink j , reached from node f_1 , is the sink j' (case b). Thus, $Resource\_diff\_delay(i, j, j) = |Delay(j') - Delay(j)| = |1700 - 1200| = 500$.
- (ii) *Scenario 2.* Expanding to nodes f_2 and f_3 leads to a routing path to sink j with $Estimated\_Total\_sw\_nb(i, n, j)$ equal to $Total\_sw\_nb(i', j')$ (case a). So, equivalent nodes of f_2 and f_3 are nodes with the same $index$ and which are t_1 and t_2 , respectively. Then, the equivalent node of sink j , reached from node f_3 , is the sink j' (case b), and in this case, $Resource\_diff\_delay(i, j, j) = |1700 - 1600| = 100$.
- (iii) *Scenario 3.* Expanding to nodes f_4 and f_5 leads to a routing path to sink j longer than the dual routing path. In this case, we must compute

```

Let:  $n$  the node currently being routed
pred( $n$ ) the predecessor node of node  $n$ .
count( $n$ ) an incremental value reserved to node  $n$ 
index(src) = 0;
count(src) = 0;
index( $n$ ) = index (pred( $n$ )) + 1;
count( $n$ ) = count (pred( $n$ ));
if (Diff_sw_nb (i, n, j)) <= 0 { /* the current routing path is the shorter or
                                equal to the dual routing path */

    if (n != j) { /* case a */
        index( $n'$ ) = index( $n$ );
        count( $n$ ) ++
    }
    else { /* case b */
        index( $n'$ ) = index( $j'$ );
    }
}
else { /* the current routing path is longer than the dual routing path */
    equivalent_index = Total_sw_nb( $i'$ ,  $j'$ ) - Expected_sw_nb( $n$ ,  $j$ );
    if (equivalent_index < count( $n$ ) + 1) { /* case c */
        index( $n'$ ) = count( $n$ );
    }
    else { /* case d */
        index( $n'$ ) = equivalent_index;
        count( $n$ ) ++;
    }
}
}

```

ALGORITHM 2: Determining the index of the equivalent node n' .

the *equivalent_index* of the current node to determine the *index* of the equivalent node in the dual tree. The node f_4 is considered as an overhead node. Its equivalent node is the node t_1 (case c). We notice that in this way, based on (2), the node f_4 will be penalized in comparison with node f_3 in terms of cost. In fact, $Resource\_diff\_delay(i, f_4, j) = 600$, whereas $Resource\_diff\_delay(i, f_3, j) = 100$.

In the case of the node f_5 (case d), the equivalent node is the node which has *index* = *equivalent_index* = $Total\_sw\_nb(i', j') - Expected\_sw\_nb(f_5, j) = 3 - 1 = 2$ and which is t_2 . Then, the delay of sink j reached from node f_5 will be compared to the delay of sink j' (case d).

Based on the aforementioned notations, we can illustrate the flow of our routing algorithm.

In the first iteration, one of two complementary nets is routed, based on the congestion cost. The routing does not take into consideration the differential delay, since the dual net has not yet been routed. Once the original net is routed, the router stores information about the routing: the used resources, the delay of each resource, and the number of used switches.

Then, during the routing of the dual net, the router computes the delay of each used resource n , the $Resource\_Diff\_delay(i, n, j)$, and the $Diff\_sw\_nb(i, n, j)$. It tries to find the path with the minimum differential delay, taking into account the resource congestion.

FIGURE 15: Routing true and dual connections.

During the next iteration, the original net will be ripped up and rerouted taking into consideration the last routing results of the dual net (in the previous iteration) and vice versa. We note that dual connections are routed in the same order.

It can be seen from (2) that a critical connection will be routed by a minimum differential delay path even if it is congested, while a noncritical connection pays less attention to delay balancing and more attention to congestion.

The PathFinder routing algorithm performs many iterations until all resource conflicts are resolved. We note that this routing algorithm is not architecture specific and it can be applied to all target platforms.

5.2. Routing Results in MFPGA. From Table 4, we can see that the new routing algorithm produces interesting results in the multilevel FPGA. Indeed, the delay balance is improved by 76% in WDDL DES design, and by 81% in all WDDL designs compared to results obtained with adjacent placement and shortest path PathFinder routing algorithm. In addition, the router succeeded to route all WDDL designs with zero switch-unbalanced signals.

The two last columns of Table 4 show that this delay balance improvement is achieved without bad effect on critical path delay compared to results obtained with routability-driven router.

5.3. Routing Results in Simple Mesh FPGA. Routing results obtained with simple mesh FPGA are summarized in Table 5. We deduce that the timing-balance-driven router is efficient in the mesh architecture too. In fact, the average delay balance is improved by 69% in all WDDL designs and the *Total_Diff_sw_nb* is reduced by 71% compared to results obtained with adjacent placement and shortest path PathFinder routing algorithm. The *Total_Diff_sw_nb* is the sum of differential switch number for all unbalanced connections. It allows to better evaluate the switch unbalance. This delay balance improvement is achieved without increasing the critical path delay compared to results obtained with routability-driven router.

5.4. Routing Results in Cluster-Based Mesh FPGA. Table 6 presents results of routability-driven (congestion-based) routing and timing-balance-driven routing applied to adjacent placed WDDL netlists in cluster-based mesh FPGA, where each cluster contains 2 LBs. Experimental results show that the new routing algorithm achieves a gain of 75% in timing balance and a gain of 76% in the total differential switches number compared to results obtained with adjacent placement and shortest path PathFinder routing algorithm.

To evaluate the overhead of the new router, we compared the critical path delay obtained with timing-balance-driven and routability-driven routing algorithms. The last column of Table 6 shows that there is no increase in the critical path delay.

When we compare the three FPGA architectures explored in this paper, we can see that the multilevel architecture is the best one in terms of switch balance. In fact, in this architecture, the number of switches used to route a connection is limited. So, it is easier to balance routed dual connections in terms of switch number. The remaining delay unbalance comes from unbalance between architecture wires length in all levels. For the mesh architecture, we take advantage of the homogenous interconnect, but on the other side, the number of switches to route a connection is far from being limited. The simple mesh architecture presents better delay balance than the multilevel architecture and than the cluster-based

architecture for cluster size 2. But in terms of area, Table 7 shows that the cluster-based mesh with cluster size 2 presents the smallest architecture. To improve the delay balance in this architecture, we propose a specific differential pair routing for this architecture.

6. Differential Pair Routing Algorithm in Cluster-Based Mesh FPGA

6.1. Differential Pair Routing Algorithm. From previous results, we show that the cluster-based mesh architecture has the smallest area. However, this architecture presents an important switches unbalance between dual connections, which causes a delay unbalance and should cause an unbalance in power consumption. The optimal balance is obtained when we have balanced wire lengths and balanced number of routing switches. For this purpose, we propose to apply a specific differential pair routing algorithm to cluster-based mesh FPGA. The goal is to always route two differential signals through identical tracks and the same number of switches. Thus, the two routes have the same capacitances and the same delays, if we do not consider parasitics and crosstalk effects.

The routing technique we propose is inspired from the differential routing in ASIC proposed in [32]. However, the approach proposed in ASIC cannot be applied to FPGA because we are limited with the available routing resources in FPGA.

To achieve differential routing, we propose first to build a double-wire routing graph. We group equivalent architecture wires into pairs to form “double” wires.

Two wires w and w' are denoted equivalent if

- (i) they belong to the same routing channel;
- (ii) they have the same logical length L ;
- (iii) they have the same direction (North, South, East, and West);
- (iv) wires w and w' together are connected to more than one input of the adjacent CLB. For example, in Figure 16, wires 0 and 2 are not equivalent since they are connected to the same CLB input. But, wires 0 and 4 can be considered as equivalent since they are connected to 2 distinct CLB inputs; thus, they can be grouped in a “double” wire;
- (v) wires w and w' together are connected to more than one output of the adjacent CLB.

Concerning wires which are inputs (or outputs) of a CLB, they are considered as equivalent if they are inputs (or outputs) at the same CLB side (top, bottom, left, and right).

To obtain a double-wire routing graph, each pair of successive equivalent wires is transformed to a single double wire. Figure 16 shows the transformed routing graph with double wires. Each double wire has a size of 2 single wires. If the architecture contains wire which does not have an equivalent one, this wire remains a single wire and it is not used to route differential signals. After that, we abstract each pair of the dual signals as 1 “double” signal. It is possible to

TABLE 4: Timing-balance-driven routing results ($\Delta delay$) of WDDL designs in MFPGA.

WDDL designs	Max (ps)	Mean (ps)	Std. Dev. (ps)	Sw. Mis. Signals	Total Diff.sw_nb	Routability_driven critical path delay (ns)	Timing_balanced_driven critical path delay (ns)
DES	571	85	92	0	0	56.62	51.55
BarreI16	395	50	55	0	0	24.04	21.14
BarreI32	921	70	75	0	0	21.23	18.52
BarreI64	1333	133	148	0	0	61.72	59.9
Mux8_64 bit	881	44	71	0	0	28.75	25.84
Mux32_16 bit	801	61	88	0	0	40.11	37.82
xbar_16 × 16	366	32	55	0	0	13.39	13.84
Average	752	67	69	0	0	35.12	32.65

TABLE 5: Timing-balance-driven routing results ($\Delta delay$) of WDDL designs in simple mesh FPGA.

WDDL designs	Max (ps)	Mean (ps)	Std. Dev. (ps)	Sw. Mis. signals	Total Diff.sw_nb	Routability_driven critical path delay (ns)	Timing_balanced_driven critical path delay (ns)
DES	739	64	75	1419	2191	19.84	19.55
BarreI16	540	45	60	371	508	9.4	8.7
BarreI32	975	56	83	875	1432	8.32	8.59
BarreI64	1620	71	107	2198	3831	25.29	20.77
Mux8_64 bit	798	25	56	371	568	12.09	10.62
Mux32_16 bit	494	20	39	268	383	17.47	16.53
xbar_16 × 16	539	30	49	232	330	8.59	6.49
Average	815	44	67	819	1320	14.42	13.03

have a WDDL cryptographic design which contains dual-rail signals and single-rail signals. In fact, cryptographic coprocessors can be separated into control and datapath. As the secret key is used only in the datapath part, leakage from the control part is not crucial. Thus to ensure security of the design it is sufficient to implement only the datapath in WDDL. This will save area as WDDL takes more area on the FPGA than a single-rail design. In this case, single-rail signals are not grouped.

Once we obtain a double-wire routing graph and double-signal netlist, we route each pair of differential signals as a one double signal using double wires of the routing graph. Single-rail signals can be routed using double wires or single wires which do not have equivalent wires. We use the shortest path PathFinder algorithm [23] to run routing.

At the end of the routing, double wires of the routing graph are split into 2 single wires, and each double signal is decomposed into 2 differential signals. Then, each wire is affected to the appropriate signal. Thus, we ensure that the 2 differential signals are routed through wires of the same length and with the same number of switches.

We note that this methodology can be applied to mesh architecture with different cluster sizes, different wire lengths, and interconnect flexibilities. But, it is obvious that an architecture containing more equivalent wires is more adapted to differential pair routing of WDDL design.

6.2. Differential Pair Routing Results. Adjacent placement and differential routing techniques are applied in cluster-based mesh architecture with 2 LBs in each cluster, wires

of logical length L equal to 1, and interconnect flexibilities equal to 1. Table 8 shows results obtained with unconstrained placement and PathFinder router and those obtained with the new placement and routing techniques. We can see that the balance is greatly improved. All dual signals of WDDL designs are routed with the same number of switches ($Sw.Mis.Signals = 0$). Besides, the remaining differential delay, which is due to the unbalanced layout of clusters local interconnect, is insignificant.

Figure 17 shows the distribution of the C_ratio of all the nets of WDDL DES design obtained with the unconstrained placement and routing, with adjacent placement and timing-balance-driven routing, and with adjacent placement and differential pair routing. C_ratio is equal to C_{true}/C_{false} , where C_{true} and C_{false} are the internal capacitances of true and false net, respectively. The capacitance per net includes wires, switches, and buffers capacitances. The variation between the capacitances at differential nets exceeds a factor of 3 for the unconstrained placement and routing. The adjacent placement and timing-balance-driven routing improves the capacitance balance. In fact, the C_ratio varies between 0.7 and 1.5, and the most differential nets have a capacitance ratio equal to 1. On the other hand, the differential pair routing presents better results. It succeeded to match capacitances of almost all nets.

The first columns of Table 9 show the architecture characteristics of the cluster-based mesh with a cluster size 2, obtained with the new differential routing. Compared to results obtained with the timing-balance-driven routing shown in Table 7, we can see that there is a little increase

TABLE 6: Timing-balance-driven routing results ($\Delta delay$) of WDDL designs in cluster-based mesh FPGA (2 LBs).

WDDL designs	Max (ps)	Adjacent placement and routability-driven routing					Adjacent placement and timing-balance-driven routing					
		Mean (ps)	Std. Dev. (ps)	Sw. Mis. Signals	Diff_sw_nb	Critical path delay (ns)	Max (ps)	Mean (ps)	Std. Dev. (ps)	Sw. Mis. Signals	Diff_sw_nb	Critical path delay (ns)
DES	2383	107	218	1099	3209	16.23	813	34	70	631	990	15.66
Barrel16	2400	260	364	604	2757	9.43	620	91	102	536	977	9.2
Barrel32	4810	463	770	1455	10654	18.32	1616	117	158	1351	2824	16.2
Barrel64	8926	646	1093	3391	32299	26.64	2131	149	197	3190	7759	23.96
Mux8.64 bit	5751	464	781	1706	14421	11.58	1899	61	132	894	1940	9.94
Mux32.16 bit	4505	159	451	852	4936	14.02	1376	46	95	846	1445	14.07
Xbar_16 × 16	3245	182	375	423	1863	8.25	621	58	82	355	581	6.32
Average	4574	325	578	136	10023	14.92	1296	79	119	1114	2359	13.62

TABLE 7: Characteristics of simple mesh, cluster-based mesh and MFPGA architectures with timing-balance-driven routing.

WDDL benches	Architecture levels	MFPGA		$N \times N$	Simple mesh		Area $\times 10^6 \lambda^2$	$N \times N$	Cluster-based mesh ($N = 2$ LBs)		Area $\times 10^6 \lambda^2$
		SW $\times 10^3$	Area $\times 10^6 \lambda^2$		Channel width	SW $\times 10^3$			Area $\times 10^6 \lambda^2$	Channel width	
DES	$6 \times 8 \times 8 \times 8$	506	1499	47×47	20	625	1797	33×33	16	474	1354
Barrel16	$6 \times 6 \times 6 \times 6$	177	526	25×25	18	160	465	18×18	16	141	405
Barrel32	$4 \times 8 \times 8 \times 8$	380	1089	38×38	24	492	1387	27×27	22	415	1156
Barrel64	$8 \times 8 \times 8 \times 8$	786	2268	56×56	32	1418	3914	40×40	24	980	2705
Mux8	$8 \times 8 \times 8 \times 8$	729	2132	55×55	32	1368	3776	39×39	12	526	1559
Mux32	$6 \times 8 \times 8 \times 8$	729	2139	54×54	24	989	2794	38×38	14	564	1638
xbar	$6 \times 6 \times 6 \times 6$	203	587	24×24	12	102	304	17×17	12	102	301
Average		501	1462			736	2062		16	457	1302

TABLE 8: Differential pair routing results ($\Delta delay$) of WDDL designs with cluster size 2.

WDDL designs	Nets	Unconstrained placement and routing			Adjacent placement and differential routing		
		Max (ps)	Mean (ps)	Sw. Mis. Signals	Max (ps)	Mean (ps)	Sw. Mis. Signals
DES	2162	5450	319	2206	13	1.2	0
BarreI16	586	2400	407	752	11.3	1.6	0
BarreI32	1406	4810	623	1816	12	2.4	0
BarreI64	3112	11318	1131	4275	16	3.2	0
Mux8	2966	7468	624	1707	14.2	1.09	0
Mux32	2988	4642	423	1773	13.9	0.9	0
xbar_16 × 16	546	1790	336	673	14	1.14	0

FIGURE 16: Double-wire routing graph.

FIGURE 17: Ratio between interconnect capacitances at true and false nets in WDDL DES design.

in area. In fact, with the same architecture size, the average channel width increases from 16 to 18.

6.3. Impact of Cluster Size. To evaluate the impact of the cluster size on the differential pair routing results, we apply the adjacent placement and the differential-pair routing for cluster sizes 2, 4, and 8. Tables 9 and 10 show the cluster size

impact on the area of the architecture and on the critical path delay. We can see that when we increase the cluster size, the area of the architecture increases. On the other hand, we can see that the number of switches on the critical path decreases. Nevertheless, the critical path delay increases, since when the cluster size increases the wire length grows up. Compared to critical path delays obtained with the routability driven routing, Table 10 shows that the differential pair routing algorithm does not increase the critical delay for cluster sizes 2 and 8 and it increases it by only 5% for cluster size 4.

Concerning the delay unbalance between differential nets, it should increase when the cluster size increases. In fact, we suppose that mesh routing wires have the same length, and that intracluster local interconnect is unbalanced. So, when the cluster size increases, the layout of the cluster becomes bigger and the unbalance between local routing wires becomes more important. To have exact values of delay unbalance, we should make the layout of the different clusters.

7. Conclusion

The wave dynamic differential logic (WDDL) is a promising countermeasure to improve the robustness of secure devices. Nevertheless, to be effective, dual signals must be balanced to have equal propagation delays. To improve the delay balance in the design, we first performed an adjacent placement technique. This placement consists in placing real and dual

TABLE 9: Cluster-based mesh architecture characteristics for different cluster sizes.

WDDL benchs	$N \times N$	Differential pair routing						$N = 8 \text{ LBs}$					
		$N = 2 \text{ LBs}$			$N = 4 \text{ LBs}$			$N = 8 \text{ LBs}$			$N = 16 \text{ LBs}$		
		Channel width	SW $\times 10^3$	Area $\times 10^6 \lambda^2$	$N \times N$	Channel width	SW $\times 10^3$	Area $\times 10^6 \lambda^2$	$N \times N$	Channel width	SW $\times 10^3$	Area $\times 10^6 \lambda^2$	$N \times N$
DES	33×33	18	522	1486	24×24	20	625	1742	17×17	28	938	2460	17×17
Barrel16	18×18	16	141	405	13×13	28	236	638	10×10	28	325	851	10×10
Barrel32	27×27	24	447	1236	19×19	26	475	1296	14×14	40	802	2076	14×14
Barrel64	40×40	26	1051	2879	28×28	30	1150	3104	20×20	44	1749	4505	20×20
Mux8	39×39	12	526	1559	28×28	16	730	2058	20×20	28	1299	3405	20×20
Mux32	38×38	14	564	1638	27×27	20	791	2204	19×19	32	1274	3318	19×19
xbar_16x16	17×17	12	102	301	12×12	18	145	412	9×9	32	286	744	9×9
Average		18	479	1357		22	593	1636		34	961	2479	

TABLE 10: Impact of cluster size on critical path delay.

WDDL benchs	Differential pair routing			N = 8 LBs			N = 2 LBs			N = 4 LBs			Routability-driven routing			N = 8 LBs		
	N = 2 LBs			N = 4 LBs			N = 8 LBs			N = 2 LBs			N = 4 LBs			N = 8 LBs		
	Critical Sw. Nb.	Critical delay (ns)	Critical Sw. Nb.	Critical delay (ns)	Critical Sw. Nb.	Critical delay (ns)	Critical Sw. Nb.	Critical delay (ns)	Critical Sw. Nb.	Critical Sw. Nb.	Critical delay (ns)	Critical Sw. Nb.	Critical Sw. Nb.	Critical delay (ns)	Critical Sw. Nb.	Critical Sw. Nb.	Critical delay (ns)	Critical delay (ns)
DES	147	16.52	154	16.78	106	18.18	149	16.23	139	14.58	92	16.86						
Barrel16	102	10.8	98	13.67	82	14.06	85	9.43	118	14	72	12.42						
Barrel32	175	20.32	167	22.37	103	20.47	176	18.32	176	21.52	106	20.7						
Barrel64	211	23.7	233	32.0	121	24.8	232	26.64	228	27.8	153	29.39						
Mux8	96	9.07	79	9.45	74	12.54	194	18	107	11.18	90	15.64						
Mux32	133	13.11	116	14.0	94	16.17	150	14	131	14.15	112	18.62						
xbar_16 × 16	73	7.04	52	6.79	51	9.56	84	8.25	53	6.53	58	9.95						
Average	133	14.36	128	16.43	90	16.53	152	14.41	136	15.71	97	17.52						

gates close to each other. Then routing results were improved by an adaptation of the PathFinder routing algorithm. The proposed router is architecture independent. It is based on the congestion-delay negotiation and takes into consideration the differential delay, the differential number of used routing resources, and the congestion cost. Placement and routing techniques are performed on tree-based, simple mesh, and cluster-based mesh architectures. The results obtained are very interesting. Indeed, compared to unconstrained placement and routing, the new placement and routing techniques achieve 95%, 93%, and 85% of average timing improvement in tree-based, simple mesh, and cluster-based mesh architectures, respectively. In addition, the router succeeded to route all pairs of dual connections of WDDL designs with the same number of resources in Tree FPGA. The remaining delay unbalance is dominated by wire delays (wires belonging to the same level have not the same length). On the other hand, in the mesh architecture, the remaining delay unbalance is dominated by switch delays.

However, the cluster-based mesh architecture with cluster size 2 presents the smallest architecture. So, to reduce further the switch unbalance and to take advantage of the homogeneous interconnect of the mesh architecture, we proposed a specific differential pair routing technique to cluster-based mesh FPGA. This routing algorithm presents the best tradeoffs. It succeeded to route differential signals through balanced routing paths in terms of wire length and switch number. However, this routing technique can be applied only for cluster-based mesh architecture.

References

- [1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '99)*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397, 1999.
- [2] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the AES S-box," in *Proceedings of the 12th International Workshop on Fast Software Encryption (FSE '05)*, vol. 3557 of *Lecture Notes in Computer Science*, pp. 413–423, Springer, Paris, France, February 2005.
- [3] D. Suzuki, M. Saeki, and T. Ichikawa, "Random switching logic: a countermeasure against DPA based on transition probability," 2004, <http://eprint.iacr.org/2004/346>.
- [4] K. Tiri and P. Schaumont, "Changing the odds against masked logic," in *Proceedings of the 13th Annual Workshop on Selected Areas in Cryptography*, vol. 4356 of *Lecture Notes in Computer Science*, pp. 134–146, Montreal, Canada, 2006.
- [5] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked AES hardware implementations," in *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '05)*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 157–171, Springer, Edinburgh, UK, September 2005.
- [6] T. Popp and S. Mangard, "Masked dual-rail pre-charge logic: DPA-resistance without routing constraints," in *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '05)*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 172–186, Springer, September 2005.
- [7] T. Popp and S. Mangard, "Implementation aspects of the DPA-resistant logic style MDPL," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '06)*, pp. 2913–2916, IEEE Computer Society, Island of Kos, Greece, May 2006.
- [8] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Proceedings of the IEEE 28th European Solid State Circuit Conference (ESSCIRC '02)*, May 2002.
- [9] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, pp. 246–251, February 2004.
- [10] A. Razajindraibe, M. Robert, and P. Maurine, "Improvement of dual rail logic as a countermeasure against DPA," in *Proceedings of the IFIP International Conference on Very Large Scale Integration (VLSI-SoC '07)*, pp. 270–275, Atlanta, Ga, USA, October 2007.
- [11] K. Tiri and I. Verbauwhede, "Prototype IC with WDDL and differential routing DPA resistance assessment," in *Cryptographic Hardware and Embedded Systems—CHES 2005*, vol. 3659 of *Lecture Notes in Computer Science*, pp. 354–365, Springer, Heidelberg, Germany.
- [12] P. Yu and P. Schaumont, "Secure FPGA circuits using controlled placement and routing," in *Proceedings of the 5th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*, pp. 45–50, Salzburg, Austria, October 2007.
- [13] R. P. McEvoy, C. C. Murphy, W. P. Marnane, and M. Tunstall, "Isolated wddl: a hiding countermeasure for differential power analysis on FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 1, pp. 1–23, 2009.
- [14] K. Baddam and M. Zwolinski, "Divided backend duplication methodology for balanced dual rail routing," in *Cryptographic Hardware and Embedded Systems—CHES 2008*, vol. 5154 of *Lecture Notes in Computer Science*, pp. 396–410, Springer.
- [15] S. Guilley, S. Chaudhuri, L. Sauvage et al., "Place-and-route impact on the security of DPL designs in FPGAs," in *Proceedings of the IEEE International Workshop on Hardware Oriented Security and Trust (HOST '08)*, pp. 26–32, IEEE, Anaheim, Calif, USA, June 2008.
- [16] S. Bhasin, S. Guilley, Y. Souissi, T. Graba, and J. Danger, "Efficient dual-rail implementations in FPGA using block RAMs," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '11)*, pp. 261–267, December 2011.
- [17] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard, "Evaluation of the masked logic style MDPL on a prototype chip," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, vol. 4727 of *Lecture Notes in Computer Science*, pp. 81–94, Springer, Vienna, Austria, September 2007.
- [18] C. Zhimin and Z. Yujie, "Dual-rail random switching logic: a countermeasure to reduce side channel leakage," in *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems*, vol. 4, Springer, Berlin, Germany.
- [19] S. Guilley, F. Flament, R. Pacalet, P. Hoogvorst, and Y. Mathieu, "Security evaluation of a balanced quasi-delay insensitive library," in *Proceedings of the Design of Circuits and Integrated Systems (DCIS '08)*, Session 5D-Reliable and Secure Architectures, p. 6, IEEE, Grenoble, France, November 2008.

- [20] S. Bhasin, J. Danger, F. Flament et al., "Combined SCA and DFA countermeasures integrable in a FPGA design flow," in *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig '09)*, pp. 213–218, IEEE Computer Society Quintana Roo, Quintana Roo, Mexico, December 2009.
- [21] M. Nassar, S. Bhasin, J. Danger, G. Duc, and S. Guilley, "BCDL: a high speed balanced DPL for FPGA with global precharge and no early evaluation," in *Proceedings of the Design, Automation and Test in Europe (DATE '10)*, pp. 849–854, IEEE Computer Society Dresden, Dresden, Germany, March 2010.
- [22] Z. Marrakchi, H. Mrabet, E. Amouri, and H. Mehrez, "Efficient tree topology for FPGA interconnect network," in *Proceedings of the 18th ACM Great Lakes Symposium on VLSI (GLSVLSI '08)*, pp. 321–326, Orlando, Fla, USA, March 2008.
- [23] V. Betz, A. Marquardt, and J. Rose, *Architecture and CAD for Deepsubmicron Fpgas*, Kluwer Academic Publishers, 1999.
- [24] G. Lemieux, E. Lee, M. Tom, and A. Yu, "Directional and single-driver wires in FPGA interconnect," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 41–48, Brisbane, Australia, December 2004.
- [25] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, 1948.
- [26] J. P. Uyemura, *Introduction to VLSI Circuits and Systems*, John Wiley and Sons, 2001.
- [27] NIST/ITL/CSD, "Data Encryption Standard, FIPS PUB 46-3," 1999, <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>.
- [28] ALTERA, *Benchmark Designs for the Quartus University Interface Program (QUIP), Version 1. 0*, ALTERA, 2005.
- [29] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of the ACM 3rd International Symposium on Field-Programmable Gate Arrays (FPGA '95)*, pp. 111–117, February 1995.
- [30] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference (DAC '99)*, M. J. Irwin, Ed., pp. 343–348, ACM, New York, NY, USA, 1999.
- [31] D. A. Papa and I. L. Markov, "Hypergraph partitioning and clustering," Tech. Rep., University of Michigan, EECS Department.
- [32] K. Tiri and I. Verbauwhede, "A digital design flow for secure integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1197–1208, 2006.