International Journal of Reconfigurable Computing

Selected Papers from SPL 2009: Programmable Logic and Applications

Guest Editors: Elías Todorovich and Valentin Obac Roda

Selected Papers from SPL 2009: Programmable Logic and Applications

International Journal of Reconfigurable Computing

Selected Papers from SPL 2009: Programmable Logic and Applications

Guest Editors: Elías Todorovich and Valentin Obac Roda

Contents

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 714270, 2 pages
doi:10.1155/2010/714270

Editorial

Selected Papers from SPL 2009: Programmable Logic and Applications

Elías Todorovich[1] and Valentin Obac Roda[2]

[1] *UNICEN University, Campus Universitario, Tandil B7001BBO, Buenos Aires, Argentina*
[2] *Departamento de Engenharia Elétrica, EESC/USP, Avenida. Trabalhador São-carlense 400 13566-590 São Carlos, SP, Brazil*

Correspondence should be addressed to Elías Todorovich, etodorov@exa.unicen.edu.ar

Received 17 February 2010; Accepted 17 February 2010

Basically, FPGA devices contain programmable logic blocks and a hierarchy of programmable interconnects that allow data routing between the blocks and to the output pins. In this way FPGAs can be used to implement any logical function that an ASIC could perform. Furthermore, the ability to update the functionality after shipping and the low nonrecurring engineering costs relative to an ASIC design, offers advantages for many applications. Many emerging applications in communications, computing, and consumer electronics industries demand that their functionality stays flexible after the system has been manufactured. Such flexibility is required in order to cope with changing user requirements, improvements in system features, changing protocols and data-coding standards, demands to support variety of different user applications, and so forth. Like microprocessors, RAM-based FPGAs can be infinitely reprogrammed. Design revisions, even for a fielded product, can be implemented quickly and painlessly. Nowadays the FPGA market is a $2.75 billion one, with more than 100,000 designs starting in 2010.

The interest on FPGAs is reflected in several first class conferences on programmable logic around the world and the number of papers published by the research community. The Southern Conference on Programmable Logic (SPL, http://www.splconf.org/) is the austral meeting point for researchers interested in FPGA technology. Started in 2005, SPL has since 2007 the technical cosponsorship of the IEEE Circuits and Systems Society (CAS). The selection of articles presented in this special issue is coming from the last SPL conference (V Southern Conference on Programmable Logic), held in São Carlos, Brazil, during April 1 to 3, 2009.

Thirty two researchers helped us in the revision process to select the final seven contributions. The issue begins with a paper by Hanyu Liu and Ali Akoglu, "Timing-driven nonuniform Depopulation Based Clustering" where a timing-driven nonuniform depopulation-based clustering technique that targets critical path delay and channel width constraints simultaneously is presented. Next, in "Flexible interconnection network for dynamically and partially reconfigurable architectures" L.Devaux et al. study various communication architectures in the context of dynamic reconfiguration, in particular interconnection networks. In "Parameterized hardware design on reconfigurable computers: an image processing case study" M.Huang et al. show how the speedup a reconfigurable computer can reach depends on the intrinsic parallelism of the target application as well as the characteristics of the target platform.

Yuet Ming Lam et al. in "Multi-loop parallelisation using unrolling and fission" present a technique for parallelising multiple loops in a heterogeneous computing system. Experimental results show that a maximum speedup of 34 is achieved on a 274 MHz FPGA for the N-Body over a 2.6 GHz microprocessor, which is 4.1 times higher than an approach without unrolling. "Power characterisation for fine-grain reconfigurable fabrics" by Tobias Becker et al. presents a benchmarking methodology for characterising the power consumption of the fine-grain fabric in reconfigurable architectures.

In "High speed FPGA 10's complement adders-subtractors" Gery Bioul et al. redesign BCD adders to fit within FPGA's platforms with promising results that enable this technology to implement new decimal floating-point

cores according to the IEEE 754-2008 standard. Finally, in "Concurrent calculations on reconfigurable logic devices applied to the analysis of video images" Sergio Geninatti et al. present the design and implementation of an algorithm for computing similarities between neighbouring frames in a video sequence using luminance information on FPGA.

We would like to express our sincere thanks to the reviewers for their hard work, to Dr. René Cumplido, the Editor-in-Chief, and to the editorial staff of Hindawi. We hope that you enjoy this special issue and that it inspires more research to overcome future challenges in the areas related to programmable logic.

Elías Todorovich
Valentin Obac Roda

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 158602, 11 pages
doi:10.1155/2010/158602

Research Article

Timing-Driven Nonuniform Depopulation-Based Clustering

Hanyu Liu and Ali Akoglu

Department of ECE, University of Arizona, 1230 E. Speedway Blvd., Tucson, AZ 85721, USA

Correspondence should be addressed to Ali Akoglu, akoglu@ece.arizona.edu

Received 14 June 2009; Accepted 16 November 2009

Academic Editor: Elías Todorovich

Low-cost FPGAs have comparable number of Configurable Logic Blocks (CLBs) with respect to resource-rich FPGAs but have much less routing tracks. For CAD tools, this situation increases the difficulty of successfully mapping a circuit into the low-cost FPGAs. Instead of switching to resource-rich FPGAs, the designers could employ depopulation-based clustering techniques which underuse CLBs, hence improve routability by spreading the logic over the architecture. However, all depopulation-based clustering algorithms to this date increase critical path delay. In this paper, we present a timing-driven nonuniform depopulation-based clustering technique, T-NDPack, that targets critical path delay and channel width constraints simultaneously. T-NDPack adjusts the CLB capacity based on the criticality of the Basic Logic Element (BLE). Results show that T-NDPack reduces minimum channel width by 11.07% while increasing the number of CLBs by 13.28% compared to T-VPack. More importantly, T-NDPack decreases critical path delay by 2.89%.

1. Introduction

Field-programmable gate arrays (FPGAs) were first introduced in 1980s. While they are less efficient than ASICs, FPGAs are becoming more popular because of their low nonrecurrent engineering cost and fast time-to-market. Currently, commercial FPGAs can be categorized as low-cost and resource-rich families. As shown in Table 1, low-cost FPGA family (Spartan) has comparable number of Configurable Logic Blocks (CLBs) with resource-rich family (Virtex), but less memory, multipliers, and routing tracks. Limitation on interconnect resources increases the probability of nets being routed through longer paths and nets becoming unroutable due to congestion. For the sake of routability, when nets go through longer paths, critical path delay may also increase. We may solve these problems by migrating to the resource-rich FPGA device which has more routing resources by paying 7× price. In order to avoid this, FPGA CAD flow must improve the routability as well as timing performance to make the low-cost device a feasible option.

FPGA CAD flow includes four stages: technology *mapping* to form a netlist of logic blocks, *clustering* to combine blocks into CLBs, *placement* to allocate physical positions to each CLB, and *routing* to define paths for all nets in the design. Clustering is the foundation of layout and has strong influence on area efficiency, timing, and power [1]. Figure 1 categorizes clustering techniques. Based on the target utilization objectives, we identify two types of clustering techniques: targeting maximum logic utilization and targeting less than maximum logic utilization. Most clustering approaches fully populate CLBs with optimization goal for routability (area), timing, or power.

However, maximum logic utilization may cause routing congestion in some parts of the FPGA. A CLB contains N basic logic elements (BLEs), where a typical BLE used in many academic studies is formed of a 4-input LUT, a flip-flop, and a MUX to choose the output from either the LUT or the flipflop. A group of CLBs is strongly connected if they share a large number of nets. After placement, such CLBs appear close together in a specific region on the FPGA. Filling these CLBs to the limit (N) increases the demand on the interconnect resources through this region to be able to route the connections among them. As a result, channel width requirements for such regions become higher than others. This leads to an increase in peak channel width and hence the design requires more routing resources.

Table 1: Features and prices of two xilinx FPGA families (spartan: low-cost and virtex: resource-rich).

Device	CLBs	RAM	Multi- pliers	Interconnect dble.+hex+1 g.[1]	Price (2008)
Spartan 3s500e	1164	36 k	20	8 + 8 + 24	$23
Virtex 2vp7	1232	792 k	44	40 + 120 + 24	$179
Spartan 3s1200e	2168	504 k	28	8 + 8 + 24	$42
Virtex 2vp20	2320	1584 k	88	40 + 120 + 24	$304

[1]dble.:double, hex, lg.: long (types of wires).

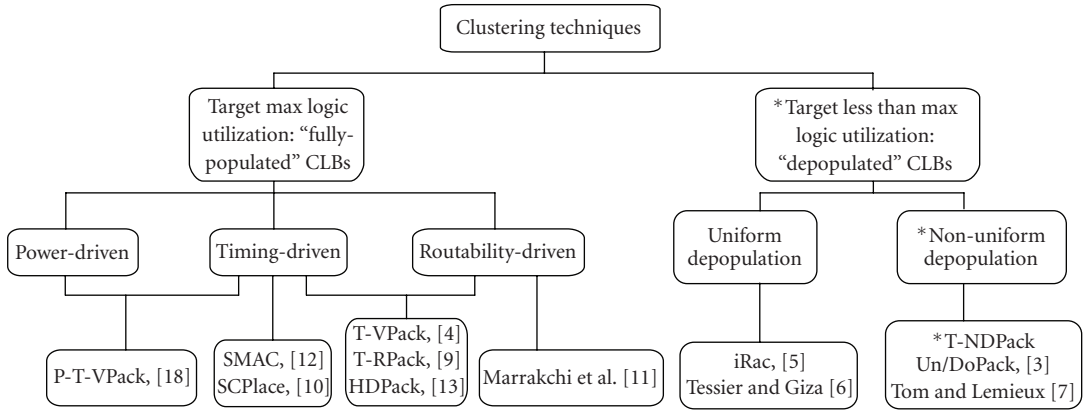


Figure 1: Categorization of clustering techniques based on logic utilization approach and optimization goals. *represents the category of our technique (T-NDPack).

It has long been known that, as CLBs are depopulated, better channel widths can be achieved. First proposed in [2], the depopulation-based clustering techniques can lower peak channel width and improve routability. Instead of targeting maximum logic utilization, depopulation is a technique that underuses CLBs by not filling them to capacity. The regions with strongly connected CLBs are spread over a larger area on the FPGA. This reduces the demand for routing resources; hence in such a region, more resources become available to route the connections.

However depopulation leads to more number of external connections among CLBs and typically results with an increase in critical path delay, because the inter-CLB delays are much larger than the intra-CLB delays [1]. For example, the latest depopulation-based clustering technique [3] decreases minimum channel width by 15% while increasing the number of CLBs by 17.32% compared to T-VPack [4]. Additionally, total area increases by 5%, along with 7% increase in critical path delay. All depopulation-based clustering algorithms [3, 5, 6] increase critical path delay, while enhancing the routability.

In this paper, we propose the first depopulation-based clustering approach that takes timing into account. Categorized in Figure 1, we develop a seed-based routability and timing-driven nonuniform depopulation technique, T-NDPack. We adjust the CLB capacity under construction based on the criticality of the BLE under consideration. For example, we cluster the nets on the critical path to full capacity. That way, we reduce the inter-CLB delay which helps decrease critical path delay. Meanwhile, we depopulate on the paths with low criticality to avoid routing congestion and hence reduce the channel width requirements. To achieve this idea, we modify both the algorithm flow and cost function of T-VPack. Results show that T-NDPack decreases minimum channel width by 11.07% while increasing the number of CLBs by 13.28% compared to T-VPack. More importantly, as opposed to the trend we see in other depopulation techniques, T-NDPack decreases critical path delay by 2.89%. With the new technique, instead of moving to a resource-rich FPGA (in the case of Spartan 3s500e versus Virtex 2vp7 of Table 1), designers may, for example, move to Spartan 3s1200e and pay 2 × instead of 7 × cost. Furthermore, this paper stands as a guide when it comes to understanding the effects of depopulation on area and delay performance for FPGAs.

Rest of the paper is organized as follows. Section 2 presents the review of the related work on depopulation-based clustering techniques. Section 3 introduces our clustering technique, T-NDPack. Section 4 presents and analyzes our experimental results. Section 5 compares our work with both depopulation- and nondepopulation-based approaches. Section 6 presents our conclusion and future work.

2. Related Work

Several depopulation techniques were proposed previously. We categorize them into two types (Algorithm 1): uniform depopulation [5, 6] and nonuniform depopulation [3, 7]. Uniform depopulation sets a fixed "upper limit" per CLB and each CLB is filled to that "upper limit" capacity. In nonuniform depopulation, the "upper limit" varies

among CLBs. Let us assume that cluster size is 8. While a uniform depopulation scheme may use a fixed "upper limit" of 6 for all CLBs, a nonuniform scheme will result in a CLB distribution with sizes from 1 to 8. nonuniform depopulation sets a very low "upper limit" to prioritize routability for a congested area and sets a high "upper limit" for the congestion free area to save more CLBs. Therefore, nonuniform depopulation results with better routability in congested area and higher CLB utilization in uncongested area compared to uniform scheme.

Tom and Lemieux [7] proposes the first nonuniform depopulation methodology. Tom uses 20 MCNC benchmark circuits [8] and connects them with three different topologies (independent, pipelined, and clique). Each topology represents an SoC. Each benchmark is an IP block and uses its own "upper limit." Results show that the SoC design with the help of depopulation technique requires less channel width compared to T-VPack. However, total area increases while maintaining similar critical path delay relative to T-VPack. Tom's approach [7] stands as a good study in terms of showing the potential benefit of nonuniform depopulation. However, the methodology determines the "upper limit" for each IP block manually based on the congestion inside the same IP block and no algorithm is given.

The nonuniform depopulation technique, Un/DoPack, was proposed by Tom et al. in [3]. This technique runs the FPGA CAD flow twice. First iteration is the regular CAD flow. In the second iteration, clustering stage uses the layout result of the first iteration and depopulates the congested regions. While reducing the channel width, Un/DoPack, similar to the other depopulation-based clustering approaches, observes an increase in total area and critical path delay.

3. T-NDPack

In this section, we describe our seed-based routability and timing-driven nonuniform depopulation clustering technique, T-NDPack. We present the pseudocode and notable implementation insights.

3.1. Algorithm Flow. T-NDPack chooses the seed block based on criticality first. The first block that is clustered into a CLB is called the seed block of this CLB. Then T-NDPack packs more blocks into the CLB by following the nonuniform depopulation clustering scheme.

We define two strategies for depopulation:

(i) BLE-limit: limit the number of BLEs used in a CLB [3, 6, 7],

(ii) input-limit: limit the number of inputs used for a CLB [5].

T-NDPack employs either "BLE-limit" or "input-limit" strategy to achieve variable utilization level. We evaluate the performance of each and present their effect on minimum channel width and critical path delay separately. In this paper, the "utilization level" measures the amount of resources used by a CLB in terms of the number of BLEs or inputs where "high utilization level" means most resources are used. For the "BLE-limit" strategy, utilization level refers to the number of BLEs used in a CLB, and for the "input-limit" strategy, utilization level refers to the number of inputs used by a CLB.

Table 2: A maximum utilization table (MUT).

The ranking percentage of the seed's criticality	Maximum utilization level	Cluster size
90%–100%	8	8
40%–90%	7	8
0%–40%	6	8

Algorithm 1 shows the pseudocode for T-NDPack. First, the algorithm computes the criticality of each block (Ln. 1) and sorts them based on their criticality (Ln. 2).

Then T-NDPack begins to fill CLBs. Algorithm keeps clustering blocks into CLBs until no unclustered block is left (Ln. 3). In each iteration, we have the following.

(1) T-NDPack packs the seed block that has the maximum criticality (Ln. 4). We regard that the criticality of the seed block represents the criticality of the net. Therefore, we determine the "maximum utilization level" based on the ranking of the seed's criticality value (Ln. 5). In this paper, the "maximum utilization level" means the maximum number of BLEs or inputs that are allowed to be used for a CLB. If the ranking of the seed's criticality value is high, algorithm sets the "maximum utilization level" to a high value to decrease the critical path delay. Nevertheless, if the ranking is low, the "maximum utilization level" is set to a low value to reduce the routing requirement.

Table 2 shows a "*maximum utilization table" (MUT)* used to look up the value of "maximum utilization level" for "BLE-limit" strategy. This MUT allows more BLEs to be used in a CLB for the seed with 95% criticality ranking than the seed with 50% criticality ranking. We explain how to generate MUT in Section 4.2.

(2) Then, T-NDPack starts to cluster blocks, till the CLB under consideration reaches its maximum utilization level (Ln. 6). Figure 2 shows the algorithm flow for packing one block into the CLB (Ln. 7–Ln. 20).

(a) T-NDPack builds a candidate block list (Ln. 7) by comparing the criticality of blocks with respect to the "*candidate block threshold" (CBT)*. CBT increases as the "current utilization level" of the CLB under consideration increases. We define "current utilization level" as the number of currently used BLEs or inputs for the CLB under consideration. The candidate block list includes all the blocks whose criticality is above the CBT value. If the CLB is already highly utilized, only the blocks with high criticality are allowed to be clustered. Furthermore, if the list is empty, T-NDPack stops clustering more blocks into this CLB (Ln. 8–Ln. 10).

(b) Then a cost function, described in Section 3.2, computes the gain value for each block in the candidate block list.

```
(1) Compute the criticality of each block
(1) Sort the blocks with criticality
(3) While unclustered blocks available
(4)    Find a seed with maximum criticality
(5)    Determine the maximum utilization level in the cluster based on the ranking of the seed's criticality
       through looking up "maximum utilization table" (MUT)
(6)    While maximum utilization level is not reached
(7)       Build the candidate block list that criticality > "candidate block threshold" (CBT) (CBT is
          determined by the current utilization level)
(8)       If candidate block list is empty
(9)          Break
(10)      End if
(11)      If the related blocks available
(12)         Choose a related block with highest gain from the candidate block list
(13)      Else if (current utilization level < "unrelated block threshold" (UBT))
(14)         Choose a unrelated block with the highest gain from the candidate block list //unrelated
             block Clustering
(15)      Else
(16)         Break
(17)      End if
(18)      Remove block from unclusered block list
(19)      Add block to current cluster
(20)      Current clustered block number ++
(21)   End while
(22) End while
```

Algorithm 1: The pseudocode for T-NDPack.

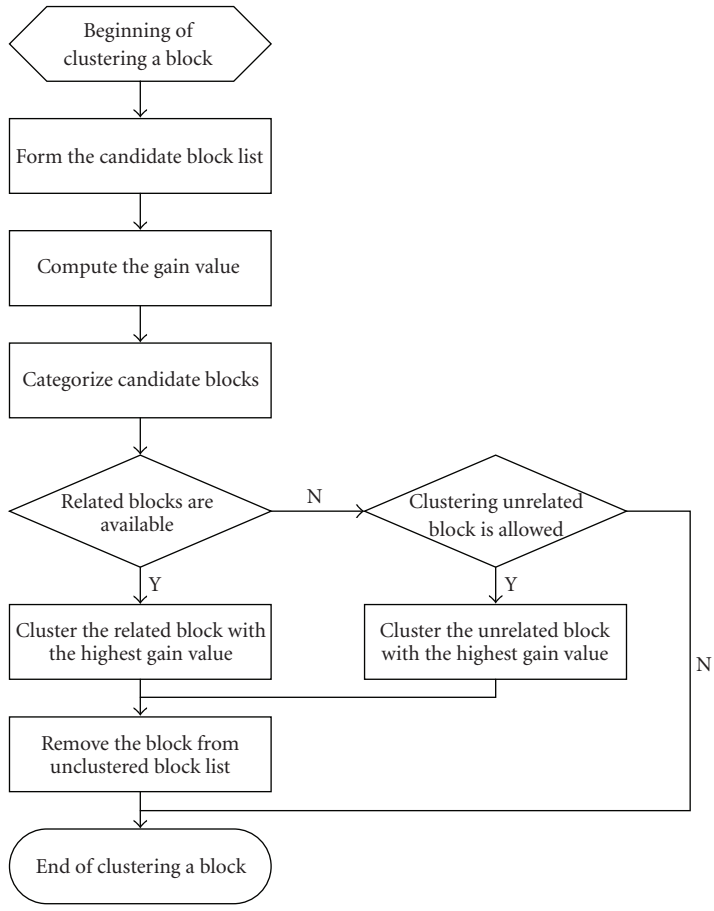


Figure 2: Algorithm flow for clustering a candidate basic logic element (BLE) into configurable logic block (CLB).

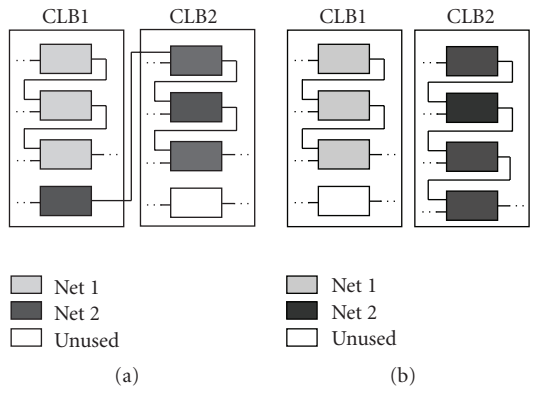


Figure 3: Impact of unrelated block clustering, Net 1 followed by Net 2. (a) introduces an inter-CLB delay and an external connection between CLB1 and CLB2. (b) no inter-CLB delay, no external connection, and less routing resource demand when CLB1 is underutilized.

(c) Next, we categorize the blocks in the candidate block list into related and unrelated blocks: a block is related if it shares inputs or outputs with the current CLB under consideration.

(d) T-NDPack tries to cluster the related block with the highest gain value first (Ln. 11-Ln. 12). If the related block is not available and clustering the unrelated blocks is allowed, T-NDPack clusters the unrelated block with the highest gain value (Ln. 13-Ln. 14). We explain this mechanism in Section 3.3.

(e) Finally, T-NDPack removes the block from the unclustered block list with the next iteration.

3.2. Cost Function. The cost function in T-NDPack considers the criticality in terms of delay and routability simultaneously (1) similar to the clustering cost function of the T-VPack [4]. The "α" parameter balances the criticality and the routability. The criticality is defined in [4] and calculated based on the sensitivity of a connection to the delay of the whole circuit. T-NDPack introduces the current utilization level as a factor to the routability component in the clustering cost function of the T-VPack. As current utilization level increases, the probability of sharing inputs and outputs increases. Therefore the value of the routability component increases. T-NDPack gradually scales more on routability part to provide informed attention to criticality:

$$\text{gain} = \alpha * \text{criticality} + (1-\alpha) * \left(\frac{|\text{NetA} \cap \text{NetB}|}{G}\right) \Big/ \text{current utilization level}. \quad (1)$$

3.3. Unrelated Block Clustering. In this section, we explain how to cluster an unrelated block. T-NDPack tries to cluster the related block with the highest gain value first. If no related block is available and only if the current utilization level is less than the "*unrelated block threshold*" (UBT), T-NDPack allows clustering the unrelated block (Ln. 13 - Ln. 14). This rule avoids clustering very few unrelated blocks and the possible inter-CLB delay. Also, this rule reduces the connections between CLBs to improve routability. For example, as shown in Figure 3, we want to cluster two nets into two CLBs, in the order of Net 1 followed by Net 2. In Figure 3(a), after Net 1 is clustered, a block of Net 2 is clustered in CLB 1. This introduces an inter-CLB delay for Net 2 and a connection between CLB 1 and CLB 2. As an alternative solution, in Figure 3(b), all blocks of Net 2 are clustered in CLB 2. In this solution, there is no inter-CLB delay or connection between CLBs. Compared to Figure 3(b), the solution in Figure 3(a) requires more routing resources and has a larger delay. Therefore if few available BLEs are left in a CLB and related block is not available, it is wiser to leave the BLEs unused.

Typically, clustering techniques modify the cost function ([4, 5, 9]) or the algorithm flow [10] or both ([11–13]). Here we summarize in what capacity the well-known approaches enhance the clustering flow and highlight where our approach stands relative to them.

T-RPack [9] uses the same algorithm flow as T-VPack and modifies the cost function. T-RPack modifies the routability part in the cost function by taking into account the individual contributions of both shared and nonshared nets between the CLB under construction and the block under consideration. T-RPack improves minimum channel width compared to T-VPack.

iRAC [5] develops a new method to choose the seed block. This technique chooses the unclustered block with the most used inputs and minimum connectivity as the seed block. iRAC then clusters each BLE into the CLB under construction using a new cost function that is based on the weight of the intersecting net and its pins that are already in the CLB. Furthermore, it uses the uniform depopulation with input-limit strategy. The algorithm flow is similar to T-VPack. However, with the modifications, iRAC achieves large reduction in the number of external nets which leads to reduction in minimum channel width.

The latest clustering technique, HDPack [13], uses a global placer to determine approximate BLE locations. Then the algorithm uses this placement information (physical information) in the clustering cost function. HDPack further incorporates a prepacking step. However, major contribution for improvement is based on clustering with the usage of physical information. The prepacking step leads to little improvement over the modified cost function.

In summary, as shown in Algorithm 1 and Figure 2, we adjust the cost function of T-VPack to pay informed attention to routability and timing by taking utilization level into account. We also modify the clustering algorithm significantly by

(i) adjusting the "maximum utilization level" at run time with maximum utilization table (MUT);

(ii) forming the "candidate block list" with candidate block threshold (CBT);

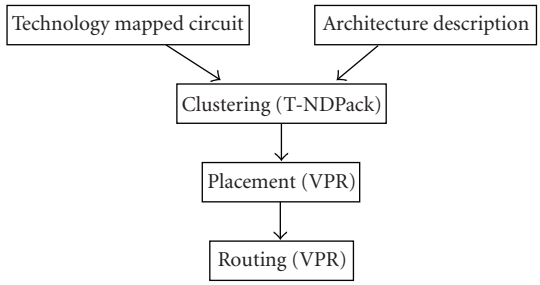


Figure 4: CAD flow.

Table 3: FPGA architecture parameters.

Architecture feature	Value
LUT inputs	4
CLB size	8
Inputs/CLB	18
Segment length	1
Fc	0.5
Fs	3

(iii) setting the "unrelated block threshold" (UBT) for clustering.

4. Experimental Results

4.1. Methodology. We implement T-NDPack based on T-VPack and conduct several experiments with the 20 largest MCNC benchmarks. We examine the performance of our proposed clustering technique and explore the effects of two depopulation strategies ("*BLE-limit*" and "*input-limit*"). Table 3 lists the main architecture parameters that we used in the experiments where segment length is the number of CLBs that a wire length spans, Fc describes the flexibility of connection blocks, and Fs describes the flexibility of switch blocks [14]. Figure 4 shows the CAD flow. The VPR version used in the experiments is v4.30.

As opposed to [3], our method runs the CAD flow once. Technology-mapped circuit and the architecture description are the inputs to the clustering stage. T-NDPack carries out clustering and VPR [15] handles placement and routing. We obtain the number of used CLBs, minimum channel width, and critical path delay for performance comparison against [4, 5, 9, 13].

4.2. Tuning the Parameters for BLE-Limit. We tune various parameters in our algorithm to identify the configuration which gives the best performance. In order to find the suitable value of "α", "UBT", "MUT", and "CBT", we performed a set of experiments following the CAD flow described in Section 4. Here we only discuss the parameter tuning study based on "BLE-limit" strategy. The parameter values for "input-limit" strategy rely on the observations on the "BLE-limit." We will discuss this in Section 4.3.

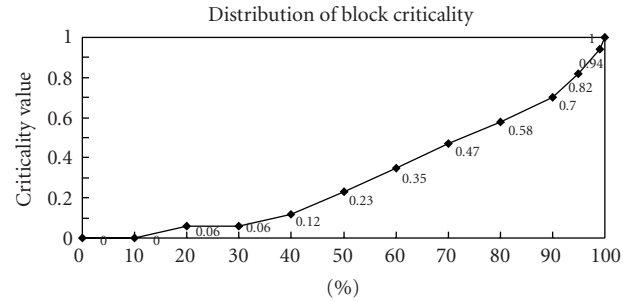


Figure 5: Distribution of CLBs based on criticality value: CLBs are uniformly distributed between 40% and 96% forming the range of Maximum Utilization Table (MUT).

Table 4: Maximum utilization table (MUT) with four partitions shows three examples for potential range configurations. configuration in table (b) results with good quality measurements; then we fine tune the range and form table in (c).

(a)

The ranking percentage of the seed's criticality	Maximum utilization level
95%–100%	8
40%–95%	7
0%–40%	6

(b)

The ranking percentage of the seed's criticality	Maximum utilization level
95%–100%	8
50%–95%	7
0%–40%	6

(c)

The ranking percentage of the seed's criticality	Maximum utilization level
95%–100%	8
51%–95%	7
0%–40%	6

(i) α: This coefficient balances the tradeoff between routability and delay. Marquardt et al. [4] shows that the value of 0.75 results with best area and delay efficient design. We believe that the behavior of the α value in our cost function is similar to [4]. Therefore, we varied α within 0.6, 0.7, and 0.75 in our experiments.

(ii) "UBT": Unrelated block threshold is used for allowing an unrelated block to be clustered into CLB. We assigned the values of 2, 4, and 6 for this parameter. During our preliminary experiments, we observed that a large value led to CLBs with too many unrelated BLEs, whereas a small value led to under utilization of CLBs. Therefore we fix UBT to 4.

(iii) "MUT": Maximum utilization table is used for setting the maximum utilization level for a CLB. We

```
(1) For (each α value)
(2)   For (each MUT)
(3)     For (each CBT)
(4)       For (each benchmark)
(5)         Run T-NDPack to obtain the number of used CLBs
(6)         Run VPR to obtain the minimum channel width and critical
            path delay
(7)       End for
(8)       Caculate the average number of used CLBs, minimum channel
          width and critical path delay
(9)     End for
(10)  End for
(11) End for
```

Algorithm 2: Experiments for identifying best configuration values for α, MUT, and CBT (note that UBT is set to be 4).

divide 0% to 100% range into 2 to 5 partitions. The maximum utilization level for the partition with the highest range is set to be the CLB capacity, 8, and this value descends by 1 relative to the ranking. Figure 5 shows the criticality value distribution of netlist "elliptic." We observe that 40% of the CLBs have criticality less than 0.12, and afterwards, criticality value is more or less evenly distributed till 0.82 criticality value (40% to 95% range). We also observe that few CLBs have criticality value larger than 0.82. We capture the nature of this distribution in MUT. Firstly, we set the upper boundary of the partition with lowest range near 40% and the lower boundary of the partition with highest range near 95%. We then partition 40% to 95% evenly based on the MUT size. Table 4(a) shows an MUT with three partitions. We also adjust the boundary by 5% or 10% to derive alterative MUTs as shown in Table 4(b). If any of the MUTs results with a good performance, we fine tune that MUT by adjusting the boundary by 1% or 3% (Table 4(c)). If not, we continue adjusting the boundary by 5% or 10%. After finding the MUT configuration that results with a good performance, it is fixed and used for all benchmarks. We do not use a different MUT for each benchmark.

(iv) "CBT": Candidate block threshold is used for allowing clustering a block into a CLB based on its criticality. CBT ranges from 0 to 1. If the current utilization level of the CLB is low (the number of BLEs used at that time for this CLB is smaller than 6), then we do not take criticality into account, and set CBT to be 0 to focus on routability. Otherwise we set CBT to be 0.2 or 0.4 for current utilization level of 6 and set CBT to be 0.8 or 0.9 for current utilization level of 7.

4.3. Effect of BLE-Limit and Input-Limit Exploration. As shown in Algorithm 2, we sweep through α, MUT, and CBT within their predefined ranges to evaluate the "BLE-limit" strategy. For each configuration, we run the clustering algorithm over 20 MCNC benchmarks and compute averages for the number of used CLBs, minimum channel width and critical path delay. Figure 6 shows the minimum channel width and critical path delay reduction of T-NDPack with "BLE-limit" relative to T-VPack. "x-axis" shows the increase in the number of CLBs and "y-axis" shows the reduction in minimum channel width and critical path delay. For each configuration, we generate two data points: a triangle representing channel width reduction and a diamond representing critical path delay reduction. We show each pair of data points (a triangle and a diamond) with a link indicating that they use the same parameter configuration. We then draw solid lines passing through the data points resulting with best reduction value in channel width and critical path delay separately. We then label the points on the line with solid triangle and diamond. We will use these solid points for analysis in Section 4.4.

For the "input-limit" strategy, instead of sweeping all parameters, we choose sample points from Figure 6 that are on the best-line (solid triangle and diamond points) and run them with "input-limit" constraint. In our experiments, cluster size (N) and the number of inputs per CLB (I) hold $I = 2N+2$ expression, which generates the best area and delay

Table 5: Conversion of parameters for input-limit strategy.

(a) Maximum Utilization Table (MUT) conversion.

The ranking percentage of the seed's criticality	Maximum utilization level (BLE-limit)	Maximum utilization level (input-limit)
95%–100%	8	18
40%–95%	7	16
0%–40%	6	14

(b) Candidate Block Threshold (CBT) conversion.

Current utilization level (BLE-limit)	Current utilization level (input-limit)	CBT
7	16	0.8 or 0.9
6	14	0.2 or 0.4
0–5	0–12	0

Table 6: α, UBT, MUT, CBT: best configuration for performance analysis of T-NDPack.

Parameter	Value	
α	0.65	
UBT	4	
MUT	The ranking percentage of the seed's criticality	Maximum utilization level
	95%–100%	8
	45%–95%	7
	0%–45%	6
CBT Table	Current utilization level	CBT
	7	0.9
	6	0.2
	0–5	0

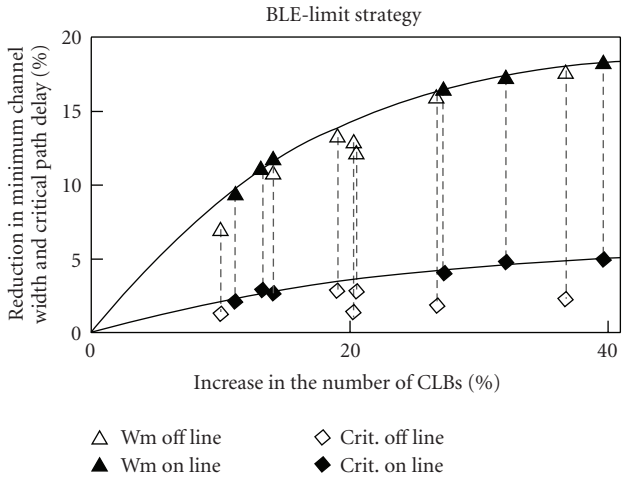


Figure 6: Minimum channel width (Wm) and critical path delay (crit) reduction of T-NDPack with "*BLE-limit*" strategy relative to T-VPack. Triangles represent channel width savings of T-NDPack relative to T-VPack. Diamonds represent critical path delay savings. Solid line represents best line drawn over the data points. Points that are on the solid line are filled (Wm on line and crit. on line). Dotted line represents triangle and diamond data points collected based on the same experimental configuration.

Table 7: Execution time over 20 MCNC benchmarks on pentium iv, core 2 Duo, 2.8 GHz with 4 GB RAM. timing measurements are based on running experiments ten times and taking their average.

Clustering algorithm	Clustering	Clustering + placement + routing
T-VPack	0.4 seconds	30 minutes 41 seconds
T-NDPack	0.56 seconds	32 minutes 58 seconds

product [16]. (As Table 3 shows, $N = 8$ and $I = 18$ in our architecture.) Therefore, we use this relationship and adjust the MUT and CBT used for "BLE-limit" to accommodate "input-limit" strategy as shown in Table 5 (converted based on Table 4(a)). Similarly, we adjust UBT to 10. Figure 7 shows the minimum channel width and critical path delay reduction of T-NDPack for the "input-limit" strategy. We will also use the solid points on this chart for analysis in Section 4.4.

Table 8: T-NDPack versus other clustering techniques: based on reported performance over T-VPack (positive percentage means improvement), #CLBs: increase in number of CLBs; Wm: minimum channel width reduction; Crit.: critical path delay reduction.

Clustering tech.	# CLBs	Wm	Crit.
[1]iRAC + iRAP	−8.78%	25.09%	N.A.
[2]Un/DoPack	−17.32%	15%	−7%
[3]Un/DoPack	−52.78%	40%	−20%
[4]T-NDPack	−13.28%	11.07%	2.89%
[5]T-NDPack	−27.31%	16.31%	4.13%

[1]Not comparable because of different placement tool. [2 3]Un/DoPack in moderate and aggressive amounts of depopulation, respectively. [4 5]T-NDPack in moderate and aggressive amounts of depopulation, respectively.

4.4. Evaluation of BLE-Limit and Input-Limit. Based on Figures 6 and 7, we tune various parameters in our algorithm to identify the good configurations whose performances are shown in Figures 8 and 9. Figure 8 shows reduction in minimum channel width and Figure 9 shows reduction in critical path delay for T-NDPack with respect to T-VPack based on "BLE-limit" and "input-limit" strategies, respectively. Solid line represents "BLE-limit" strategy and dashed line represents "input-limit" strategy. Each point with the same x-value in Figures 8 and 9 is generated with the same configuration of the parameters. Figures 8 and 9 show the following.

(i) As the number of CLBs increases, we observe a reduction in both channel width and critical path delay.

(ii) The amount of channel width and critical path delay savings gradually decreases as the number of CLBs increases.

Furthermore, we observe that the "BLE-limit" strategy is better than the "input-limit" strategy. We see a couple of reasons for this behavior. For example, if the criticality of the seed block is high, algorithm sets a high value for

Table 9: Comparison between T-NDPack and T-VPack on the critical path delay, minimum channel width, and number of used CLBs metrics over 20 MCNC benchmark circuits (positive percentage means improvement).

	Critical pathdelay(10^{-8}s)			Minimum channel width			Number of used CLBs		
	T-VPack	T-NDPack	Change	T-VPack	T-NDPack	Change	T-VPack	T-NDPack	Change
alu4	3.47	3.61	−4.03%	41	37	9.76%	193	219	−13.47%
apex2	4.30	4.46	−3.88%	58	49	15.52%	240	272	−13.33%
apex4	4.23	4.43	−4.73%	58	55	5.17%	165	183	−10.91%
bigkey	2.31	2.06	10.75%	28	27	3.57%	214	233	−8.88%
clma	10.5	7.84	25.72%	72	64	11.11%	1055	1234	−16.97%
des	4.42	5.12	−15.86%	25	21	16.00%	200	227	−13.50%
diffeq	4.19	4.13	1.24%	34	27	20.59%	189	220	−16.40%
dsip	2.52	2.07	18.01%	24	23	4.17%	172	179	−4.07%
elliptic	6.56	5.82	11.28%	53	45	15.09%	454	511	−12.56%
ex1010	6.80	6.60	2.81%	63	56	11.11%	601	679	−12.98%
ex5p	3.96	4.36	−10.30%	55	47	14.55%	138	154	−11.59%
frisc	7.96	7.30	8.25%	59	54	8.47%	446	514	−15.25%
misex3	4.00	3.47	13.16%	46	42	8.70%	179	203	−13.41%
pdc	6.61	6.47	2.11%	88	74	15.91%	582	659	−13.23%
s298	7.50	7.09	5.40%	34	33	2.94%	243	274	−12.76%
s38417	4.89	4.98	−1.85%	43	41	4.65%	802	947	−18.08%
s38584	4.17	4.15	0.54%	45	41	8.89%	806	946	−17.37%
seq	3.87	4.20	−8.39%	51	43	15.69%	221	252	−14.03%
spla	5.81	5.45	6.24%	67	63	5.97%	469	535	−14.07%
tseng	4.21	4.15	1.31%	34	26	23.53%	133	150	−12.78%
Ave.			2.89%			11.07%			−13.28%

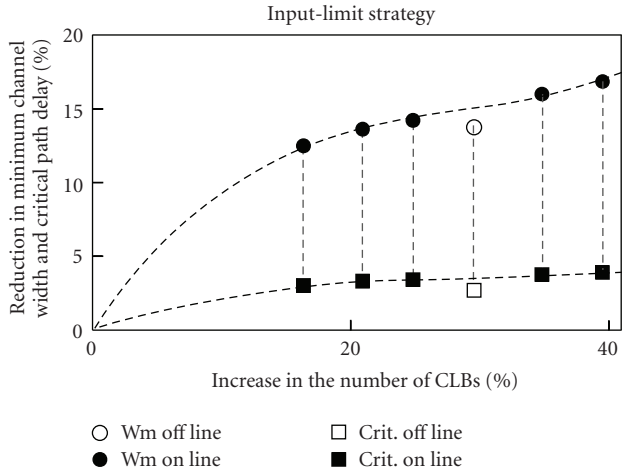


Figure 7: Minimum channel width (Wm) and critical path delay (crit) reduction of T-NDPack with "*input-limit*" strategy relative to T-VPack. Circles represent channel width savings of T-NDPack relative to T-VPack. Squares represent critical path delay savings. Dashed line represents best line drawn over the data points. Points that are on the line are filled (Wm on line and crit. on line). Dotted line represents circle and square data points collected based on the same experimental configuration.

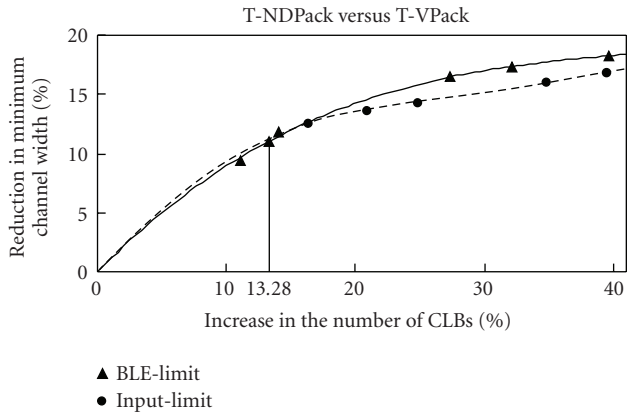


Figure 8: "*x*-axis": increase in the number of CLBs, "*y*-axis": reduction in minimum channel width relative to T-VPack. Solid line represents: T-NDPack with "BLE-limit" strategy. Dashed line represents: T-NDPack with "input-limit" strategy. As number of CLBs increase, minimum channel width decreases for both strategies.

the maximum utilization level of the CLB under construction (e.g., 16 out of 18 inputs). This affects the logic utilization significantly in a CLB. We observed cases like usage of 4 out of 8 BLEs. In another case, for a seed that has low criticality, our algorithm allows 12 inputs for that CLB. However, due to the input sharing, most of the inputs were absorbed (6 BLEs). Therefore "input-limit" technique in some cases worked against the objective of depopulation technique.

Based on these observations, we choose "BLE-limit"-based technique for performance comparison against other

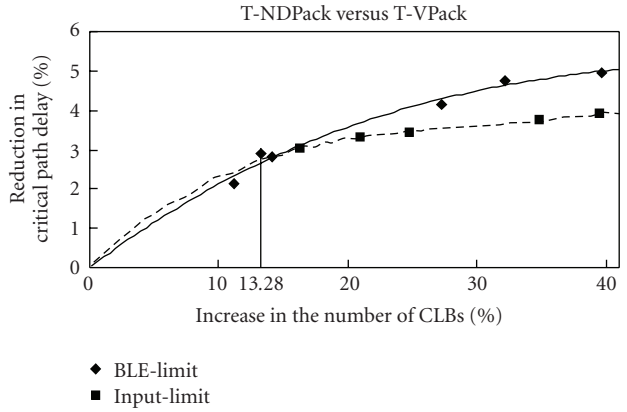


Figure 9: "x-axis": increase in the number of CLBs; "y-axis": reduction in critical path delay relative to T-VPack. Solid line represents: T-NDPack with "BLE-limit" strategy. Dashed line represents: T-NDPack with "input-limit" strategy. As number of CLBs increase, critical path delay decreases for both strategies.

clustering techniques. As shown in Figure 8, the channel width increases along with an increase in the number of CLBs. We decompose total area into logic and routing and use (2) as a model to derive the area estimate. In this paper, we regard 70% for routing area as a good estimation for the commercial FPGAs [5]. As used in [3], let "L" be the number of CLBs and let "W" be the channel width, then

$$\frac{\text{Area}_{\text{new}}}{\text{Area}_{\text{old}}} = 0.3 * \frac{L_{\text{new}}}{L_{\text{old}}} + 0.7 * \frac{W_{\text{old}}}{W_{\text{new}}} * \frac{L_{\text{new}}}{L_{\text{old}}}, \qquad (2)$$

where new represents after depopulation, and old represents before depopulation.

Among the points in Figures 8 and 9, we find that 13.28% average increase in the number of CLBs is the data point that leads to the best area-delay product. Table 6 shows the parameter values used for this data point. We run 20 MCNC benchmarks with the configuration parameters shown in Tables 3 and 6. We then compare minimum channel width, critical path delay, and the number of CLBs with T-VPack in Table 9. On average T-NDPack reduces minimum channel width by 11.07%. This results with 4.50% area increase. On average, the critical path delay decreases by 2.89%. Spreading the logic among the available CLBs is expected to increase the critical path delay. We observe this trend for some of the benchmarks with T-NDPack; however for most of the benchmarks we observe a reduction in critical path delay.

4.5. Run-Time. Table 7 compares T-VPack and T-NDPack based on the time it takes to run the clustering stage for all 20 MCNC benchmarks. Adjusting the level of depopulation-based on the criticality contributes to the execution time; therefore T-NDPack increases the run-time of the clustering stage on average by 0.16 seconds. However, this overhead is minor when the execution time for the CAD flow is considered. Since T-NDPack generates more number of CLBs to be placed and routed, we also observe an increase in the execution time for the placement and routing stages.

5. Discussion

In this section, we compare T-NDPack with other depopulation-based state-of-the-art clustering techniques and Table 8 summarizes it.

Un/DoPack [3] is a nonuniform depopulation technique. Un/DoPack achieves up to 40% channel width reduction through aggressive depopulation with a critical path delay penalty of 20%. In contrast, T-NDPack reduces critical path delay as the intensity of the depopulation increases. The trend line in Figure 8 shows that T-NDPack can further improve on channel width and continue reducing the critical path delay by using more CLBs (e.g., T-NDPack[4] versus T-NDPack[5] in Table 8). However, this leads to a significant area penalty which may prevent the designer from mapping the design onto a low-cost FPGA.

iRAC [5] is a routability driven uniform depopulation clustering technique. iRAC achieves 25.09% reduction in channel width. However, [5] reports its results based on a different placement algorithm, iRAP, which reduces channel width over VPR. We use VPR for the placement. Since neither iRAC nor iRAP is publicly available, it is not feasible to make a fair comparison without implementing their algorithms. iRAC [5] does not report timing results. It is also not feasible to reach a conclusion on overall performance without considering area and delay simultaneously.

6. Conclusion and Future Work

It has long been known that, as CLBs are depopulated, better channel widths can be achieved. However depopulation leads to more external connections among CLBs and typically results with an increase in critical path delay. While enhancing routability through depopulation is essential for utilizing the low-cost FPGAs, at the same time there is a need for addressing the critical path delay. We achieve this goal with T-NDPack by adjusting the capacity of the CLB under construction based on the criticality of the logic block under consideration.

In this study, we show that the depopulation-based clustering techniques while reducing the stress on routing can also achieve reduction in critical path delay. This is significant as this study shows that depopulation-based clustering potentially allows the designer to stay with the low-cost FPGA family instead of migrating to the costly resource-rich FPGA family.

In [17, 18], Pandit introduces a wirelength prediction techni-que that accurately estimates postplacement individual wirelength information for a given netlist before the clustering stage. As future work, we plan to incorporate this mechanism into our clustering cost function to further improve the performance of the T-NDPack.

References

[1] A. Marquardt, V. Betz, and J. Rose, "Speed and area tradeoffs in cluster-based FPGA architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, pp. 84–93, 2000.

[2] A. DeHon, "Balancing interconnect and computation in a reconfigurable computing array," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '99)*, pp. 69–78, Monterey, Calif, USA, 1999.

[3] M. Tom, D. Leong, and G. Lemieux, "Un/DoPack: re-clustering of large system-on-chip designs with interconnect variation for low-cost FPGAs," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '06)*, pp. 680–687, San Jose, Calif, USA, November 2006.

[4] M. Marquardt, V. Betz, and J. Rose, "Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *Proceedings of the ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays (FPGA '99)*, pp. 37–46, Monterey, Calif, USA, February 1999.

[5] A. Singh and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '02)*, pp. 59–66, Monterey, Calif, USA, 2002.

[6] R. Tessier and H. Giza, "Balancing logic utilization and area efficiency in FPGAs," in *Proceedings of the International Workshop on Field Programmable Logic and Applications (FPLA '00)*, pp. 535–544, Villach, Austria, 2000.

[7] M. Tom and G. Lemieux, "Logic block clustering of large designs for channel-width constrained FPGAs," in *Proceedings of the Design Automation Conference*, pp. 726–731, Anaheim, Calif, USA, 2005.

[8] S. Yang, "Logic synthesis and optimization bench-marks, version 3.0," Tech. Rep., Microelectronics Center of North Carolina, 1991.

[9] E. Bozorgzadeh, S. O. Memik, X. Yang, and M. Sarrafzadeh, "Routability-driven packing: metrics and algorithms for cluster-based FPGAs," *Journal of Circuits, Systems and Computers*, vol. 13, no. 1, pp. 77–100, 2004.

[10] G. Chen and J. Cong, "Simultaneous timing driven clustering and placement for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPLA '04)*, pp. 158–167, Antwerp, Belgium, August 2004.

[11] Z. Marrakchi, H. Mrabet, and H. Mehrez, "Hierarchical FPGA clustering to improve routability," in *Proceedings of IEEE International Conference on Reconfigurable Computing and FPGAs*, Puebla, Mexico, 2005.

[12] J. Y. Lin, D. Chen, and J. Cong, "Optimal simultaneous mapping and clustering for FPGA delay optimization," in *Proceedings of the Design Automation Conference*, pp. 472–477, San Francisco, Calif, USA, 2006.

[13] D. T. Chen, K. Vorwerk, and A. Kennings, "Improving timing-driven FPGA packing with physical information," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 117–123, Amsterdam, The Netherlands, August 2007.

[14] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, Dodrecht, The Netherlands, 1999.

[15] V. Betz and J. Rose, "VPR: a new packing placement and routing tool for FPGA research," in *Proceedings of the International Workshop on Field-Programmable Logic and Application (FPLA '97)*, pp. 213–222, London, UK, 1997.

[16] V. Betz and J. Rose, "Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size," in *Proceedings of the Custom Integrated Circuits Conference*, pp. 551–554, Los Alamitos, Calif, USA, 1997.

[17] A. Pandit and A. Akoglu, "Wirelength prediction for FPGAS," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 749–752, Amsterdam, The Netherlands, August 2007.

[18] J. Lamoureux and S. J. E. Wilton, "On the interaction between power-aware CAD Algorithms for FPGAs," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '03)*, pp. 701–708, San Jose, Calif, USA, November 2003.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 390545, 15 pages
doi:10.1155/2010/390545

Research Article

Flexible Interconnection Network for Dynamically and Partially Reconfigurable Architectures

Ludovic Devaux, Sana Ben Sassi, Sebastien Pillement, Daniel Chillet, and Didier Demigny

IRISA, Université de Rennes, 6 rue de Kerampont, BP 80518, 22302 Lannion, France

Correspondence should be addressed to Ludovic Devaux, ludovic.devaux@irisa.fr

Received 1 July 2009; Accepted 30 November 2009

Academic Editor: Elías Todorovich

The dynamic and partial reconfiguration of FPGAs enables the dynamic placement in reconfigurable zones of the tasks that describe an application. However, the dynamic management of the tasks impacts the communications since tasks are not present in the FPGA during all computation time. So, the task manager should ensure the allocation of each new task and their interconnection which is performed by a flexible interconnection network. In this article, various communication architectures, in particular interconnection networks, are studied. Each architecture is evaluated with respect to its suitability for the paradigm of the dynamic and partial reconfiguration in FPGA implementations. This study leads us to propose the DRAFT network that supports the communication constraints into the context of dynamic reconfiguration. We also present DRAGOON, the automatic generator of networks, which allows to implement and to simulate the DRAFT topology. Finally, DRAFT and the two most popular Networks-on-Chip are implemented in several configurations using DRAGOON, and compared considering real implementation results.

1. Introduction

Steady technological evolutions, increasingly complex applications, can be supported by reconfigurable architectures. This is particularly true into the framework of complex signal processing applications. However, when the number of tasks constituting an application exceeds the available hardware resources, designers have two options: increasing the number of resources (which increase the complexity of the systems) or implementing only the tasks that should be executed at a given time. In the latter case, tasks are swapped at the end of their execution by freeing logical resources for the others. However, this concept is relevant only if a new task can be implemented instead of a former one without disrupting the execution of other tasks. This concept of hardware preemption of the tasks is called *Dynamic and Partial Reconfiguration* (DPR).

The objective of the FOSFOR research project (Flexible Operating System FOr Reconfigurable devices) [1] is to specify and to implement an *Operating System* (OS) that provides an abstraction of technology for future applications. For this purpose, the FOSFOR OS operates the DPR in order to support complex applications that cannot be statically implemented due to physical restrictions. Several services of the OS are implemented in hardware whereas others are computed in software. Physical implementation of some services allows the OS to efficiently manage hardware and software tasks. In this direction, the FOSFOR project focuses on the hardware implementation of the main services: the task placer and scheduler, the memory manager, and the communication service.

In this work, the communication service is investigated. The objective is to define and to implement in an FPGA a generic interconnection architecture which should support the diversity of applications and the dynamic management of the tasks. For this purpose, the architecture takes into account the constraints imposed by the DPR. Through the physical interconnection architecture and its control, the communication service provides a flexible way for transferring data between every *Communicating Element* (CE) in an FPGA. Since an application task can be implemented in hardware or processed in software by a hardware processor,

CEs are defined as the hardware elements which exchange data. So, a CE can be the hardware implementation of a task (static or dynamic), a shared element (memory, input/output), or a hardware processor running software tasks.

The paper is organized as follows. Assumptions induced by the DPR that should be supported by an interconnection architecture are presented in Section 2. Then, current interconnection architectures are detailed and reviewed in Section 3. The interconnection network called *Dynamic Reconfiguration Adapted Fat-Tree* (DRAFT), which is specifically designed to support the DPR requirements in FPGAs, is presented in Section 4. *Dynamically Reconfigurable Architectures compliant Generator and simulatOr Of Network* (DRAGOON), the automatic generator of networks supporting the DRAFT topology is introduced in Section 5. Then, the comparison between DRAFT and the two more popular *Networks-on-Chip* (NoC) topologies is presented with respect to implementation costs and network performances in Section 6. Finally, an implementation of the DRAFT network into the framework of an application from the FOSFOR project is detailed in Section 7.

2. Assumptions on Considered Interconnection Architectures

Keeping in view of a large range of applications, an interconnection architecture should support several constraints into the framework of an implementation in FPGAs. Current applications are very complex and their task graphs exhibit a large degree of parallelism. Thus, from the interconnection point of view, the architecture must provide the possibility to realize several communications in parallel. Furthermore, dynamic placement and scheduling of CEs in an FPGA require a high level of flexibility, since the placement and the scheduling of the CEs are dynamic. So, neither the location of CEs nor the data traffic (uniform, all to one, etc.) can be predicted at compile time. These requirements of flexibility should be considered by the network topology, the routing protocol, and the available network performances (bandwidth and latency). An application is typically split into dynamic and static tasks, and there are no reason for every task to be implemented in homogeneous hardware CEs. This point differs from approaches like [2] where CEs are supposed to be homogeneous so that the interconnection topology can be dynamically reconfigured into an FPGA in order to fit the application. Heterogeneous hardware CEs are considered in this work. Hence, a single interconnection topology compliant with this heterogeneity is preferred to dynamically reconfigurable topologies considering their placement constraints as the placement of the CEs themselves.

Considering current FPGAs, Altera devices provide abundant programmable resources but do not support the partial reconfiguration of everyone of them [3, 4]. Atmel series AT40K5AL to AT40K40AL are compliant with the DPR of their resources but do not provide more than 50 K gates which are few when considering complex applications

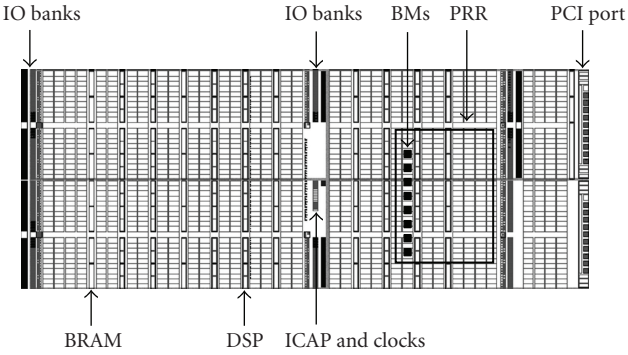


Figure 1: View of a Xilinx Virtex5 5VSX50T FPGA captured from the PlanAhead Xilinx software [7].

[5]. Hence, Xilinx FPGAs (especially the Virtex 2Pro, Virtex 4, Virtex 5, and Virtex 6) are the only ones that both provide sufficient programmable resources to implement complex applications and support the DPR paradigm [6]. Xilinx architectures offer a column-based distribution of their resources, as shown Figure 1. In a Virtex family (except in Virtex 2Pro), the center column is very specific because it contains not only Input/Output (IO) banks, but also clock managers and DPR ports (ICAP). So, despite their column-based structure, FPGAs are very heterogeneous. Consequently, the interconnection should support the heterogeneity of the FPGAs.

Xilinx DPR takes place in specific reconfigurable regions called *Partially Reconfigurable Regions* (PRRs), which hence constitute the dynamic parts of a system. Dynamic CEs are allocated in these regions. Static CEs (static hardware tasks, memories, processors, etc.) are implemented all around the PRRs and constitute the static part of the system. Communications between dynamic and static regions are performed through interfaces called Bus Macros (BMs) in Xilinx architectures [8]. Since PRRs and BMs are defined statically at compile time, every CE is connected to the interconnection architecture through static interfaces (despite the dynamic locations of the CEs). So, the interconnection architecture can be static if it provides sufficient flexibility to support the DPR paradigm.

3. Interconnection Architectures: State of the Art

In this section, several interconnection architectures, suitable for supporting the paradigm of the DPR, are analyzed and evaluated. The main constraints are the parallelism of the communications, the flexibility, and the compliance with the connection of heterogeneous CEs in heterogeneous FPGAs.

3.1. Bus-Based Architectures. Buses are interconnection architectures which are simple to implement and control, while requiring few hardware resources.

3.1.1. Main-Bus-Based Architectures. One of the first bus-based approaches designed for DPR was Core Unifier [9].

Core Unifier is a tool that allows connecting on the fly dynamic CEs on a bus. For this, the tool adds 3-states buffers to the bus on which dynamic CEs are connected. However, at compile time, Core Unifier needs the knowledge of when and where the CEs should be allocated. So, this approach is not compliant with systems in which scheduling and placement are dynamic.

The Bus-Macro-based architecture [10] and Recobus [11] are the most recent bus-based dynamic interconnection architectures. The communication interfaces of the Recobus approach require less logical resources than the Bus-Macro-based architecture. However, both approaches are based over a horizontal bus connecting vertically placed CEs. CEs are implemented using all the logical resources of a column. So, those approaches are only compliant with the 1D placement (CEs are allocated using several columns of resources of the FPGA) of the dynamic CEs. It is the major limitation of these architectures because it leads to a waste of hardware resources, depending on the size of the CEs. Furthermore, current FPGAs (except the Xilinx Virtex 2Pro series) are compliant with the 2D placement (rectangular region based) of CEs. So, none of these approaches seems suitable to implement a system using DPR in latest FPGAs.

Another architecture is the HoneyComb [12], which is a static interconnection architecture connecting static CEs but differing from other bus-based structures. This network, made of regular hexagonal cells linked together through bidirectional buses, allows data routing. HoneyComb offers flexibility at the data path level while using few hardware resources. However, the regular structure is not compliant with current FPGAs nor with dynamic and heterogeneous CEs. So, Honeycomb brings more flexibility than other bus-based architectures but it does not support the DPR paradigm due to its regular structure.

3.1.2. Limitations of Buses. Buses exhibit several drawbacks when considering an application using the DPR. A major constraint for the bus-based interconnection architecture is the necessity to support several data transfers in parallel. One bus supports only a single transfer at a time. There are two solutions for this problem. One is to use several communication lines (multiple buses [13]); the other is to split a bus into a set of independent segments interconnected through several bridges (GALS approach [14]). However, either solution implies an increase of the required hardware resources: the connection of two buses generates a need of buffers (synchronous or asynchronous FIFOs). In order to estimate the size of these buffers, the knowledge of the data traffic is required to avoid bottleneck risks. In a dynamic system where the locations of CEs and the data traffic are unknown at compile time, buffers should be designed to support the worst cases of communication. For instance, the RMBoC approach [15] uses multiple and segmented buses for DPR compliance. However, despite the estimation of buffer sizes, this approach requires a lot of hardware resources to connect the CEs.

To date, the bus-based architectures have not been implemented efficiently to answer the DPR paradigm. Poor flexibility and scalability also with placement limited to 1D are the main drawbacks of bus-based approaches which motivated many researchers to consider the *Network-on-Chip* (NoC) paradigm [16].

3.2. Network-on-Chip Based Architectures. A NoC can be seen as a set of routers, links and network interfaces. The communication topologies and routing protocols (including data flow control, scheduling policies, etc.) are key domains of research [17]. Nowadays, there are a lot of NoCs which can be classified in a few basic families, depending on their topology. Hence, we shall concentrate on the meshes, the application-dependent topologies, the rings and the trees (Figure 2).

3.2.1. Mesh Topology. A mesh topology offers a simple connection scheme (Figure 2(a)), based on a matrix of routers interconnecting regularly implemented CEs. Many meshes were implemented such as HERMES [18]. If the number of connected CEs is N and if D is the radix of the mesh, then the number of routers needed to build a square mesh is

$$R_{\text{mesh}} = \left\lceil \sqrt{N} \right\rceil^2 = D^2. \tag{1}$$

Whereas the number of needed communication links is

$$L_{\text{mesh}} = N + 2\left(D^2 - D\right). \tag{2}$$

The connection of heterogeneous CEs to a regular matrix based network may be problematic. A solution consists in considering that tasks are implemented using one or several homogeneous CEs [19] which are interconnected regularly by the mesh. This solution is used, for example, by the DyNoC approach [20]. Unfortunately, logical resources could be wasted when a small task is executed by a large CE. When a large task is implemented using several CEs, some logical resources are also wasted depending on the granularity of the CEs (Figure 3).

When a task is implemented using several CEs, then CEs can communicate in two ways. The first one is to consider that the communications between the CEs are performed through the network. However, this solution constrains the design of the tasks because each internal communication should be designed to be performed by the network. So, with this solution, the complexity of the tasks increases and the number of required resources too. Thus, this solution is impractical for designers. Another solution is to consider that a task is implemented with dedicated links between the CEs. However, the compliance between the task and the technology becomes difficult due to the limited (and reduced) number of available communication links.

A truly regular mesh is based on the assumption that the FPGA is intrinsically homogeneous. So, every CE presents the same hardware properties for task implementation. Considering the heterogeneous structure of available FPGAs, it seems quite difficult to implement a truly regular mesh. Finally, another problem of the mesh structure lies in the connection of shared elements like memories or IOs. Indeed,

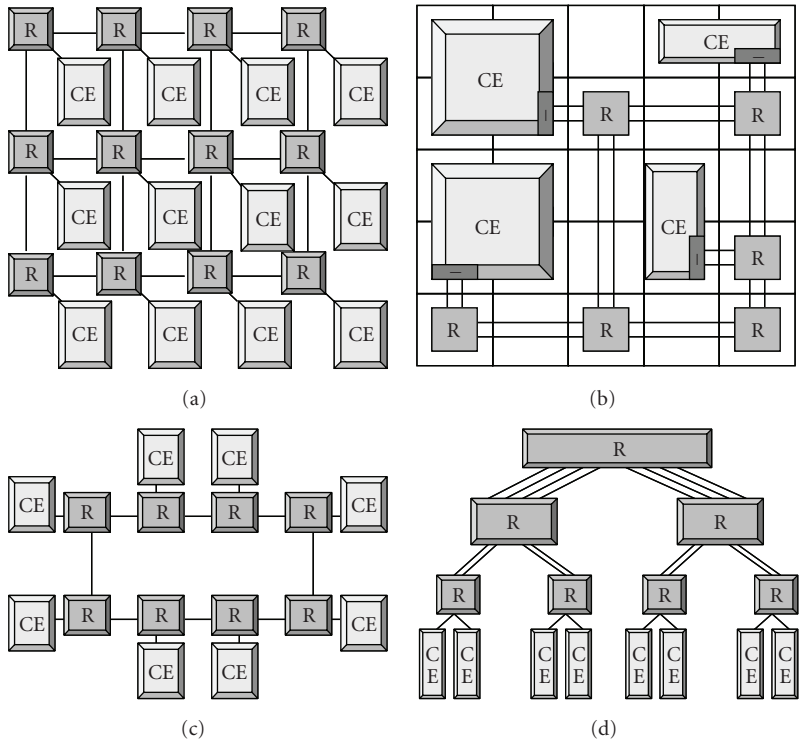


Figure 2: Network topologies: (a) mesh, (b) application-dependent topology, (c) ring, and (d) tree. Routers are denoted with "R" and communicating elements with "CE".

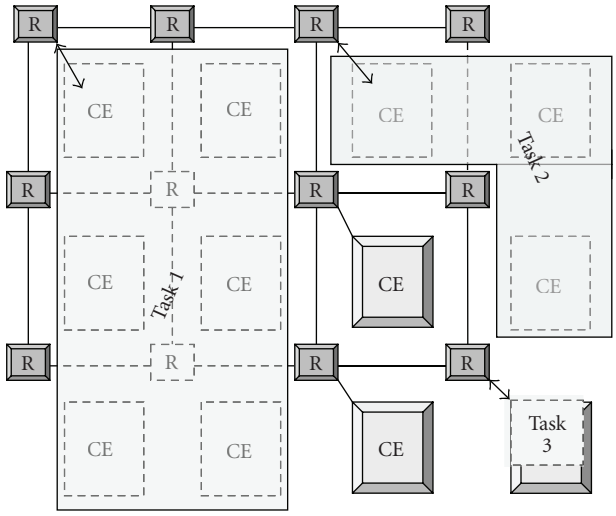


Figure 3: Implementation of three different tasks (executed by one or several CEs) using a mesh network.

the communication requirements between tasks (hardware or software) and shared elements induce the creation of hotspots, which could increase the likelihood of livelock and deadlock.

3.2.2. Application-Dependent Topology. CoNoChi (Figure 2(b)) shows an example of network which topology differs depending on the application [21]. In this approach, the FPGA is divided into regular dynamically reconfigurable regions, with each being defined dynamically as a router, a set of interconnection links, or a CE.

Despite many advantages, the concept of dynamic topology is impractical since PRRs are statically defined in current FPGAs. Hence, their use for link implementation implies wasting a lot of logical resources. Moreover, considering the resulting topology of CoNoChi which differs with the application, the design time can be important to obtain optimal network performances. Indeed, the routing protocol should take into account the network topology which is application-dependent. Keeping in mind these drawbacks, static and generic topologies are preferred to dynamic ones in order to limit the design times and to increase the predictable nature of the network performances.

3.2.3. Ring Topology. Rings (Figure 2(c)) are presently used by industrials like IBM with the Cell Broadband Engine [22]. This architecture can be very efficient. However, in some applications, it offers lower performances than a mesh [23]. Indeed, the available bandwidth depends on the characteristics of the routers and on the number of interconnected CEs. Another drawback of the rings lies in the dependence between latencies and CE locations. So, in the context of the DPR, rings should be interesting if bandwidth did not decreased when the number of connected CEs increases, and furthermore if the latency was not dependent with the placement of the CEs.

3.2.4. Fat-Tree Topology. Trees, and more precisely fat-trees (Figure 2(d)), are indirect interconnection networks. Some routers are only used for data transfers and do not connect directly the CEs. A fat-tree is based on complete binary tree. Every CE is connected to a router located at the base-level of the tree. Each hierarchical level of the fat-tree is linked to upper and lower levels through bidirectional links [23]. The main characteristic of a fat-tree is the aggregative bandwidth (offered by all the links located at the base of the tree) which remains constant between each hierarchical level all the way to the root. A fat-tree offers many advantages compared with other topologies, two of which are a large bandwidth and a low latency [24]. A fat-tree can also simulate every other topology at the cost of the appropriate control [25].

Since every CE is connected to a base-level router of the fat-tree, they are not distributed into the network structure. This point is important because CEs can be heterogeneous from the resource consumption point of view. Indeed, the resource heterogeneity does not impact the transfer times like in a mesh. Furthermore, the structure of the tree does not need to be implemented regularly. So, a fat-tree is compliant with current FPGAs.

Thanks to its constant bandwidth between every hierarchical level, the fat-tree avoids deadlock risks which exist in other topologies without an appropriate control [26].

However, a fat-tree needs many logical resources for routing purpose when the number of connected CEs increases significantly. If the number of connected CEs is N and the number of communication ports for each router is k, then the number of routers in a fat-tree [26] is

$$R_{\text{fat-tree}} = \frac{2N}{k}\left(\log_{k/2} N\right). \tag{3}$$

In this formula, assuming that the fat-tree is complete in terms of connected CEs, N is expressed by $N = 2^x$ where x is an integer and $x \geq 1$. When N does not match the previous formula, designers should build the network considering the admissible value of N just higher in order to keep the complete tree-based structure of the network. The number of connection links needed by the fat-tree topology [26] is

$$L_{\text{fat-tree}} = N\left(\log_{k/2} N\right). \tag{4}$$

To limit the resource consumption, the XGFT [27] allows the connection of several CEs to only one communication port of a router. Indeed, the fat-tree connects several sets of CEs, which are interconnected by a bus. This approach is very interesting in the context of a static application because this topology optimizes the number of connected CEs in comparison with used resources for routing purpose. However, the XGFT is not optimal into the framework of the DPR. Indeed, the sets of CEs have the same drawbacks as bus-based architectures (control time, one communication at a time without multiple buses, etc.).

3.3. Summary of the Interconnection Architectures. The compliance between presented interconnections and the constraints of the DPR paradigm are summarized in Table 1.

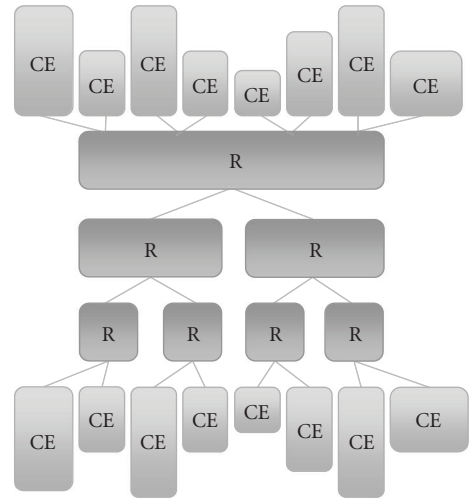


Figure 4: The DRAFT topology. Communicating elements "CEs" are connected to the root and base-levels of a fat-tree network.

From Table 1, a fat-tree is best adapted to the DPR paradigm and applicative requirements. However, its resource consumption remains the main drawback for an implementation into an FPGA. Thereby, this study of current interconnections leads to the *Dynamic Reconfiguration Adapted Fat-Tree* (DRAFT) network.

4. Flexible Interconnection: DRAFT

From the comparison of current NoCs, the fat-tree appears to be the most suitable interconnection architecture to support the DPR paradigm and applicative requirements. DRAFT is a fat-tree-based network whose main characteristic lies in the reduction of needed resources for routing purpose. Like a fat-tree, DRAFT interconnects several CEs, which could be implementations of hardware tasks (static or dynamic), processors running software tasks, and shared elements like shared memories.

4.1. DRAFT Topology. The concept proposed in DRAFT is to directly connect half of the CEs to the root-level routers of a fat-tree (Figure 4). This concept reduces a lot the number of hardware resources used for routing purpose compared with the number of connected CEs. Indeed, for the same number of connected CEs, the number of routers is divided by two when compared with the fat-tree topology.

Concerning network performances, the distances between the root and base-level CEs are constant whatever their locations. So, the minimal latencies of these communications are constant. However, for the effectiveness of this topology, it is necessary that the CEs connected to the root communicate only with the base-level connected ones. This assumption avoids the creation of hotspots in the root-level router. Communications between the CEs connected to the root would require additional hierarchical levels (leading to the fat-tree topology) in order not to increase the load on the root-level router. However, the base-level connected CEs can

Table 1: Compliance between current interconnection structures and DPR constraints.

DPR requirements	Buses	Mesh	CoNoChi	Ring	Fat-tree
Dynamic scheduling	NO	OK	OK	OK	OK
2D placement	NO	OK	OK	OK	OK
Heterogeneous CEs	OK	NO	OK	OK	OK
Heterogeneous FPGAs	OK	NO	NO	OK	OK
Resource consumption	OK	OK	NO	OK	NO
Routing flexibility	NO	OK	—	NO	OK
Communication parallelism	NO	OK	OK	OK	OK
Bandwidth	—	NO	—	NO	OK
Latencies	NO	NO	—	NO	OK

freely communicate with every other CE. This assumption is very important and while it is observed, there is no limitation concerning the nature of the CEs. A designer is free to connect its shared elements to the root-level of DRAFT, but also some hardware tasks (static or dynamic) or even processors. So, at the cost of this assumption, DRAFT is completely flexible.

4.2. Hardware Requirements. The hardware resources required by DRAFT should be considered first. If the number of connected CEs is N and the number of communication ports for each router is k, then the number of routers in DRAFT is

$$R_{\text{DRAFT}} = \frac{N}{k}\left(\left(\log_{k/2} N\right) - 1\right). \quad (5)$$

In this formula, assuming that DRAFT is complete in terms of connected CEs, N should be in the form of $N = 2^x$ where x is an integer and $x \geq 2$. If the number of CEs does not match previous formula at design time, then designers should build the network considering the just higher admissible value of N. Similarly, the number of connection links needed by DRAFT is

$$L_{\text{DRAFT}} = \frac{N}{2}\left(\log_{k/2} N\right). \quad (6)$$

From this last formula, DRAFT uses two times less connection links than a fat-tree. This is an advantage for the implementation in current FPGAs.

There are several ways to see a fat-tree and so is DRAFT. Thereby, the two fat-trees presented in Figures 4 and 5 have the same properties regarding the CEs. Indeed, a router in Figure 4 can be broken up into a set of several unitary routers called fat-node (Figure 5). This permits to build the network by using a single router type, which is generically defined, and makes the automatic generation of DRAFT easier. However, the latter structure is not fully compliant with the connexion of the CEs to the root-level, since there is only one admissible data path between base- and root-level CEs. The router-based structure (Figure 4) is more flexible due to multiple data paths. If a communication link is already used, data can be transferred through another one to the same destination. This traffic adaptive approach is not possible in the fat-node-based structure for the communications between CEs from the root and base-levels. However, in both structures, a traffic adaptive routing can be implemented for the communications between two base-level connected CEs. So, the fat-node-based structure is less flexible than the router-based one but more generic. Furthermore, it allows to demonstrate the viability of DRAFT even if it constrains the data paths. In a first time, the focus is over the fat-node-based structure due to the generic routers. In future works, the structure will be switched to the router-based one in order to support applications requiring multiple data paths between elements from the base and root-levels.

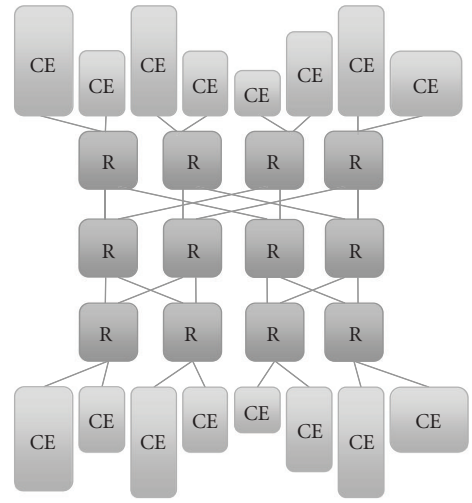


Figure 5: Fat-node view of DRAFT architecture. A fat-node is a set of unitary routers.

4.3. Principles of Connection of Various CEs. Since many applications require data transfers between static and dynamic CEs, they should be connected to DRAFT. For this purpose, designers can connect the static CEs directly to the routers, or with a bus-based sub-network (multiple buses e.g.,) connected to a single router, like for the XGFT network. In the latter structure, while CEs are static, the sub-network can be optimally designed for their communication needs. Then, the sub-network and its connected CEs (statically implemented tasks, processors, shared elements, etc.) are viewed by DRAFT as a single CE. On the other hand,

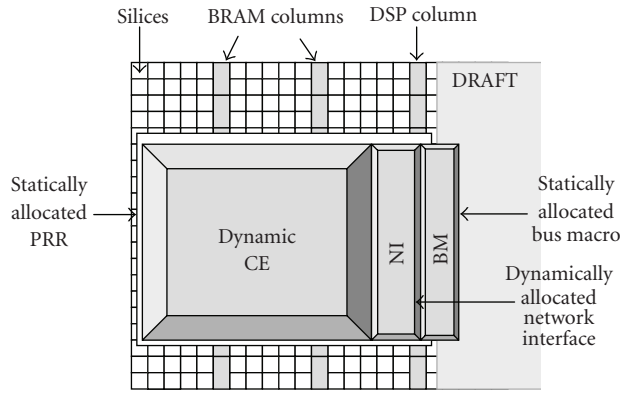


Figure 6: PRR receiving a dynamic CE connected to DRAFT through the dynamic NI and the static BMs.

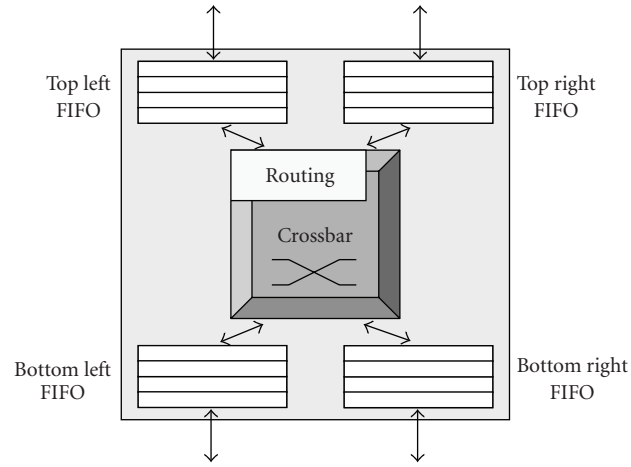


Figure 7: View of the DRAFT unitary router architecture.

the static CEs which do not exchange data with dynamic ones should be interconnected through a separate network optimally designed for this purpose. Similarly, the shared elements which do not communicate with dynamic CEs but only with static ones should be connected to a sub-network rather than to DRAFT. These principles are advised to reduce the size of DRAFT and so its resource consumption. Doing so, DRAFT can be seen as an independent core of network or even as an *Intellectual Property* (IP) block connecting each part of the application, while providing the flexibility required by the DPR paradigm.

For applications using the DPR, DRAFT does not directly connect the tasks through their network interfaces, but through the Bus Macros (BMs). So, BMs are the interfaces between DRAFT and the dynamic CEs (including their network interfaces (NIs)) as presented in Figure 6.

This concept of dynamically reconfigurable NIs is important because they can be designed optimally for their corresponding CEs. This allows to reduce the hardware cost of the NIs when a CE does not have the same interface as the others. So during the dynamic reconfiguration of a given PRR, DRAFT interface remains the same even if the newly allocated CE presents a specific interface. So, this concept makes DRAFT more generic and more flexible considering the location of the CEs.

4.4. Hardware Characteristics of the Routers. The router architecture (Figure 7) is based over four bidirectional communication ports, each including an asynchronous FIFO. FIFO sizes are defined by the designer depending on the flit width (data are fractioned into several small sets of bits called flits), and on the number of flits to store, according to the data flow and the livelock management protocol (credit based, priority, round robin, etc.). A crossbar, controlled by a routing manager which protocol is based on a Turn-Back algorithm [28], performs the routing of data.

The next destination of a message is computed by each router receiving it. The decision is made using several masks over the message source and destination addresses included into the message header. Each router is identified by an internal address which indicates its hierarchical level and its location into this level. Similarly, every CE connected to DRAFT is identified by an internal address which is used to specify the source and the destination of a message. This addressing of the CEs and routers is presented in Figure 8.

Thanks to these addresses, each router uses a hierarchical level dependent mask to determine if a flit should be routed toward an upper hierarchical level in order to reach a different part of the tree. So, each data is routed toward the high until it is able to go down to the desired half (or subpart) of the tree. This lowering routing is directly applied to the CEs connected to the root-level of DRAFT. Destination addresses are sufficient to determine toward which part of the tree a data should be routed. This algorithm, presented in Algorithm 1, provides a minimal distance to the data transfers and the guaranty that there is no deadlock risk [27]. In this algorithm, the source and destination addresses of a data are called, respectively, *CEsrc* and *CEdest*. The *mask* is directly calculated from the *Y* address of the router corresponding to its hierarchical level. As an example, for router X:0010 Y:0001, the corresponding *mask* is 0011. The *Mshift* parameter is the *mask* previously calculated shifted right of one bit set to 1. Thus, in this example, *Mshift* is 1001. Similarly, the *RXshift* is calculated from the X address of the router shifted left of one bit, that is, 0100 in the example.

In order to keep static the DRAFT architecture into the framework of the DPR, addressing of the routers and CEs is generic. However, since many CEs are dynamic, it constrains the designer to make sure that the task placer/scheduler keeps up-to-date a routing table. This table is essential for the network interfaces of the CEs to make the correspondence between the internal addresses and the physical elements (implemented task, memory, etc.).

4.5. Implementation of DRAFT in a Xilinx FPGA. DRAFT placement is important in the conception of a system using the DPR, because it impacts the use of the reconfigurable resources as well as the network performances. Thus, the concept presented in Figure 9 is to implement DRAFT as a central column into the FPGA. This concept is particularly

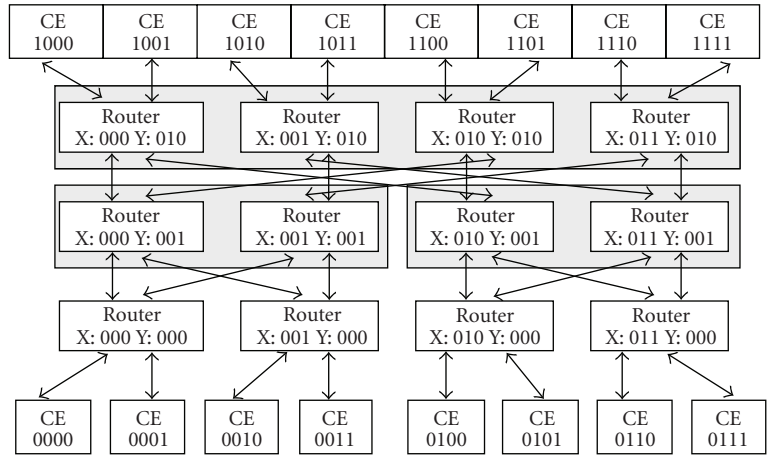


Figure 8: Generic addressing of the CEs and routers depending on their hierarchical levels and their locations into these levels.

```
IF("data from port West or South")
// data arrived from a lower level
        IF((RXshift AND mask) XOR (CEdest AND mask) = 0)
        // is the aimed subpart of DRAFT reachable from this router ?
                "Yes: routing toward lower level"
        ELSE
                "No: routing toward upper level depending on CEsrc"
        END IF
ELSE
// data arrived from an upper level
        IF((RXshift AND Mshift) XOR (CEdest AND Mshift) = 0)
        // which subpart of DRAFT is the data destination ?
                "routing toward the right side"
        ELSE
                "routing toward the left side"
        END IF
END IF
```

Algorithm 1: Routing algorithm implemented into DRAFT routers.

adapted to current technologies supporting the DPR: Xilinx Virtex 4, Virtex 5, and Virtex 6 FPGAs. CEs are implemented into both halves of the FPGA with the static elements of the application (processor, etc.). Since DRAFT is not distributed into the FPGA, the designer is not constrained by the network for the definition of the CEs in terms of sizes and locations. This is an advantage for the implementation of heterogeneous dynamic CEs. Thus, the implementation of DRAFT is fully compliant with current technology and the DPR requirements.

In present technology, PRRs and BMs are defined statically, but there is no physical obstacle to make them dynamic. The limitation is only due to the design tools which do not support the dynamic definition of the partially reconfigurable regions. Consequently, if the design software allows the definition of dynamically locatable PRRs, then every base- (or roots) level router should be reachable from both halves of the FPGA. Doing so, the dynamic relocation of a PRR from one half of the FPGA to the other will be supported.

In Xilinx FPGAs, shared IOs should be located into the central IO bank column. Similarly, it is recommended to locate the shared memories into the BRAM columns the nearest of the central column. Thereby, the communications between the CEs and the shared elements encounter a minimal latency.

5. Presentation of DRAGOON

In this section, the design software called *Dynamically Reconfigurable Architecture compliant Generator and simulatOr Of Network* (DRAGOON) is presented. DRAGOON is a conception environment specifically designed to generate and to simulate the DRAFT topology. It is inspired from ATLAS which was developed to support Hermes NoCs [18]. DRAGOON is also compliant with the fat-tree topology. The conception flow provided by DRAGOON is illustrated Figure 10, in which every step corresponds to a tool.

The NoC generation tool produces the VHDL description of DRAFT and test benches written on SystemC,

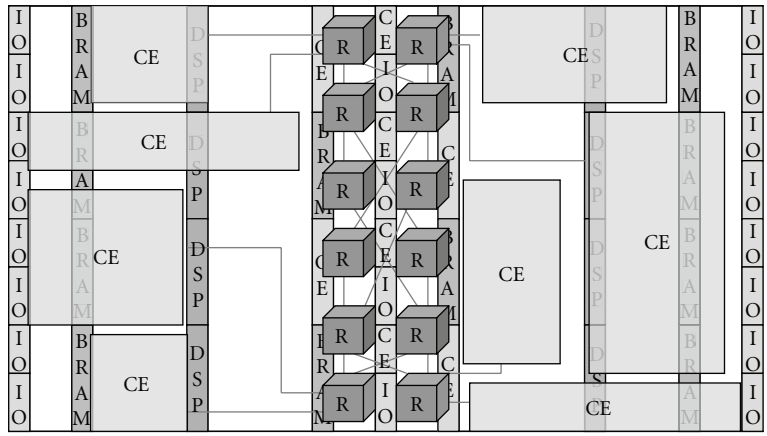


Figure 9: Implementation of DRAFT as a central column interconnecting CEs which are located into both halves of the FPGA.

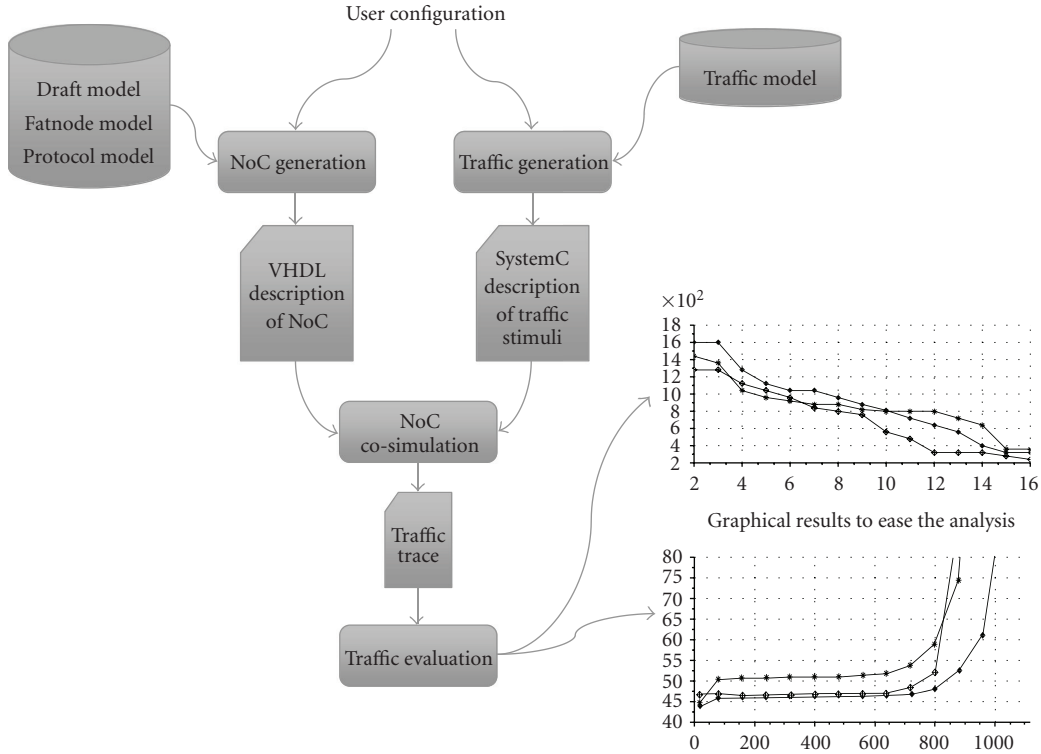


Figure 10: Conception flow of DRAGOON.

according to the configuration chosen by the user. This latter is able to choose the network dimension in terms of connected CEs, and also the flit width or the buffer depth. The flit width parameter determines the length of the data (fractioned into unitary flits) exchanged through the network. Flit width can be set to 16, 32, or 64 bits. The buffer depth indicates how many flits can be stored into one of the four FIFOs of a router. So, a router can store 4, 8, 16, or 32 flits into each of its 4 ports. The number of virtual channels can also be chosen by the user. Using virtual channels, a router is able to support several communications in parallel at the cost of hardware resources. Concerning the data traffic, the type of flow control (credit based or handshake) and the scheduling policy (round robin or priority) are parameterizable. So, the NoC generation tool allows generating DRAFT networks adapted to the requirements of many applications.

The traffic generation tool produces different data traffics (uniform, normal, pareto on/off). Each traffic simulates an application supported by DRAFT. A uniform traffic simulates applications using a constant data flow like video processing. A normal traffic is provided by applications using data dependency like pattern recognition or target tracking. Finally, the pareto on/off traffic simulates the communications between a task and a shared memory where data is transmitted using a burst mode (period of

uninterrupted data transmission followed by a period of silence). Furthermore, the traffic generation tool allows the designer to simulate several configurations of the connected CEs. Indeed, the frequencies of the CEs, the targets of data (random or specific), the number of packets to send, and the number of flits in a packet are parameterizable. Designer can also specify the transmission rate of each CE. Using all these parameters, the traffic generator builds input files containing data to be transmitted through the network.

The simulation tool invokes an external VHDL simulator (ModelSim). This simulator was chosen because it supports mixed VHDL and SystemC. Thus, the simulation tool uses the description of the NoC and the generated traffic. This traffic is injected into DRAFT during the simulation phase, which is concluded when the output files are generated.

The evaluation tool provides the interpretation of the results thanks to the previously generated output files. Results are analyzed and presented through graphics and analysis reports. Network performances like latency and throughput are the main results provided by the evaluation tool.

6. Implementation Results

In this section, DRAFT implementation results are presented. Thanks to the automatic network generator (DRAGOON), DRAFT is compared with the mesh and the fat-tree topologies. This comparison takes into account the use of hardware resources and the network performances. The impacts of the NoC and traffic parameters over hardware and network characteristics are also presented. Hardware resources are obtained thanks to Xilinx ISE 9.2i tool chain [29], and network performances are measured through ModelSim 9.5c [30] and presented thanks to DRAGOON. From this comparison, the viability and the effectiveness of DRAFT are demonstrated.

For a fair comparison of the three different network topologies, some hypothesis must be considered. Every topology is implemented for maximal network performances. So, both DRAFT and fat-tree architectures are based over a complete binary tree whatever the number of connected CEs. Similarly, every implemented mesh presents a square matrix based structure whatever the number of connected CEs. The mesh topology is implemented and simulated using ATLAS while DRAFT and fat-tree topologies are provided by DRAGOON. Since the fat-tree and DRAFT are generated with a fat-node-based structure, the three networks are implemented with the same router architectures and without virtual channels. The routing algorithm is the only component which differs from a topology to another one, so the topologies are compared independently of their router architecture. Every router is clocked at 100 MHz.

6.1. Hardware Resources. DRAFT is defined as a network which supports the DPR requirements and minimizes the hardware resource consumption. In this part, implementation results are investigated, and presented in Figure 11. For this purpose, each router was implemented with a flit width of 32 bits and a buffer depth of 4 flits.

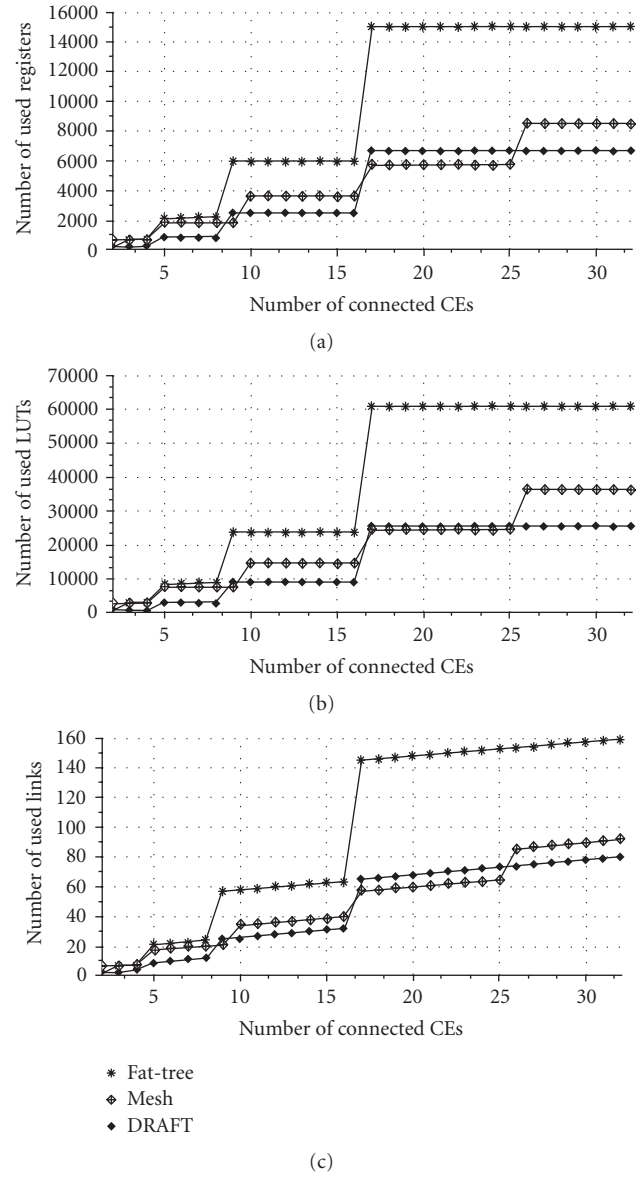


Figure 11: Number of registers (a), LUTs (b), and links (c) used for DRAFT, fat-tree, and mesh implementations in a Xilinx Virtex5 depending on the number of connected CEs.

From these implementation results, as expected, a fat-tree needs more hardware resources and more communication links than every other topology. However, using the root-level connection of the CEs, DRAFT needs less hardware resources and less communication links than a mesh when the number of connected CEs is less than 16. When this number increases, resource consumptions are very close but DRAFT needs less resources when the number of connected CEs becomes close to a power of 2. However, DRAFT outnumbers mesh resources in small ranges like from 17 to 25 connected CEs. This point is due to the assumption that DRAFT is implemented as a complete

binary tree and the mesh as a square matrix. Hence, DRAFT needs 32 routers to connect from 17 to 32 CEs while a mesh needs only 25 routers to connect from 17 to 25 CEs. Then, this latter needs 36 routers to connect from 26 to 32 routers. The number of routing wires increasingly becomes a limiting factor in FPGAs, so the fact that DRAFT requires less links than a mesh is an important matter for the designers. Using the DRAFT network, designers have more communication links available for the implementation of their tasks.

6.2. Network Performances. Usually, latency and throughput results are presented depending on the injection rate. The injection rate corresponds to the percentage of the maximal bandwidth which is used to send data from the CEs point of view. DRAFT and fat-tree topologies have two CEs connected to each base- (or roots) level router. However, in a mesh, only one CE is connected to a router, which directly impacts the injection rate. As an example, a 100% injection rate in a mesh corresponds to a data rate of 3200 Mbit/s per CE whereas it corresponds to 1600 Mbit/s per CE in DRAFT or in a fat-tree. The three topologies are simulated connecting 8 CEs with 32 bits flit width and a buffer depth of 4 flits. The frequency of each CE is 100 MHz, and data are sent with a uniform repartition of their sources and destinations. Results are presented in Figure 12. The number of data to transmit is calculated depending on the injection rates for 1ms of continuous data injection.

For a fair comparison of the three topologies, a comparison of the latencies and throughputs depending on the transfer rates is presented in Figure 13.

From these results, if the NoCs are compared depending on the injection rates calculated from their maximal bandwidth, the mesh topology saturates with an injection rate of 25%. The fat-tree and DRAFT saturate, respectively, around 55% and 60% of injection rates. This point is important because it shows that DRAFT supports a higher injection rate than every other topology. Furthermore, with the comparisons depending on the transfer rates, it appears that DRAFT provides a lower average latency while supporting higher transfer rates than the others. This minimal latency is due to the reduction of the number of routers. DRAFT offers also a higher throughput when considering the data rates. This demonstrates a better use of the routing resources than for mesh and fat-tree topologies.

6.3. Scalability. From previous implementation results, the lower resource consumption of the DRAFT network was pointed out. However, it is interesting to verify the scalability of the three topologies considering the network performances and the hardware needs. For this comparison, the networks are implemented with the same characteristics as in previous parts. For the network performances, the highest acceptable data rates are chosen in order to place the networks in the worst conditions. So, the data rate is fixed to 800 Mbit/s for each router. Simulations are realized sending 1562 packets of 16 flits each for an injection time of 1ms. Network performances are presented in Figure 14.

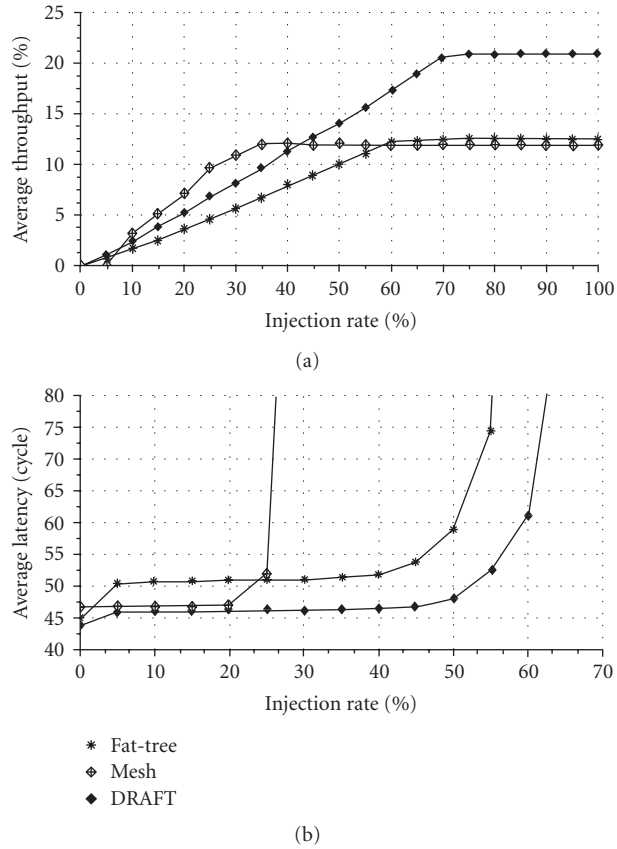


Figure 12: Comparison of the average throughputs (a) and latencies (b) depending on the injection rates for 8 connected CEs.

Thanks to its higher resource consumption, a fat-tree is able to support a higher number of simultaneously connected CEs than other topologies. In these worst transfer conditions, a mesh can manage 9 CEs, DRAFT can handle 10, and the fat-tree supports 13 CEs. Consequently, until 10 simultaneously connected CEs, DRAFT needs less hardware resources and provides a lower latency than other networks. For the connection of more than 10 CEs, the data rates should be restricted or the networks should be adapted for higher network performances at the cost of hardware resources.

The adaptation of the network parameters is now considered in order to improve the network performances. Two parameters are investigated: the flit width and the buffer depth. Results are presented for each parameter considering the required registers and LUTs also with the average latencies. The impact of the flit width is shown Figure 15. The networks are always designed for the connection of 8 CEs, with a buffer depth of 4 flits. The data rate could not be fixed to 800 Mbit/s due to a saturation of the latencies with a flit size of 16 bits. Thus, presented latencies were obtained for a data rate of 400 Mbit/s per CE.

So, the flit size has a great influence over the resource consumption. The impact of this parameter, between 32 and 64 bits, is limited with a data rate of 400 Mbit/s. However, at 800 Mbit/s, it reduces the latencies from 48.05 to 32.31

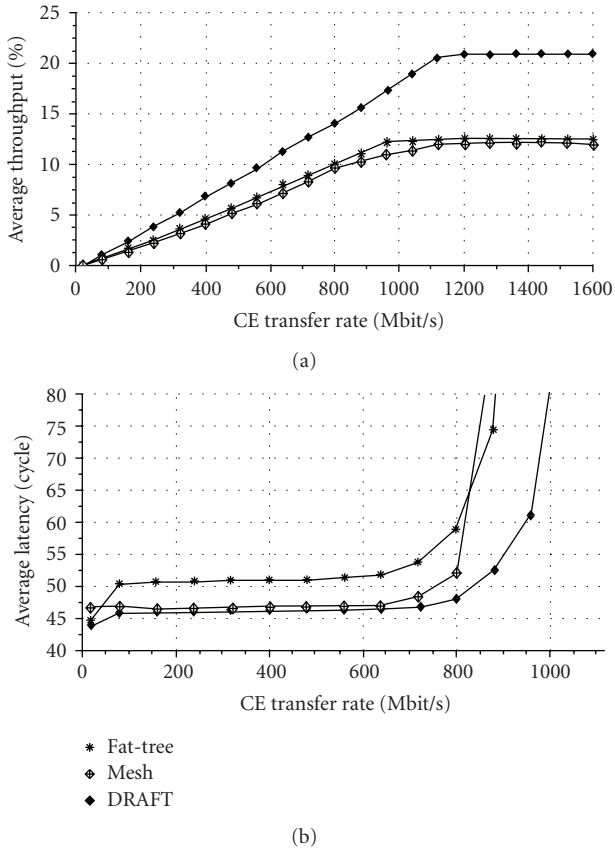


Figure 13: Comparison of the average throughputs (a) and latencies (b) depending on each CE transfer rate.

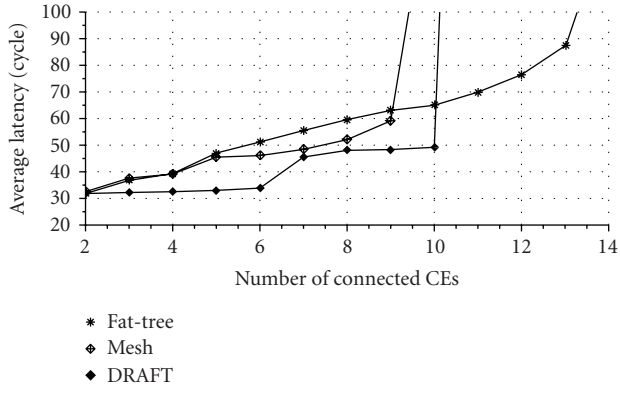


Figure 14: Comparison of the average latencies depending on the number of simultaneously connected CEs.

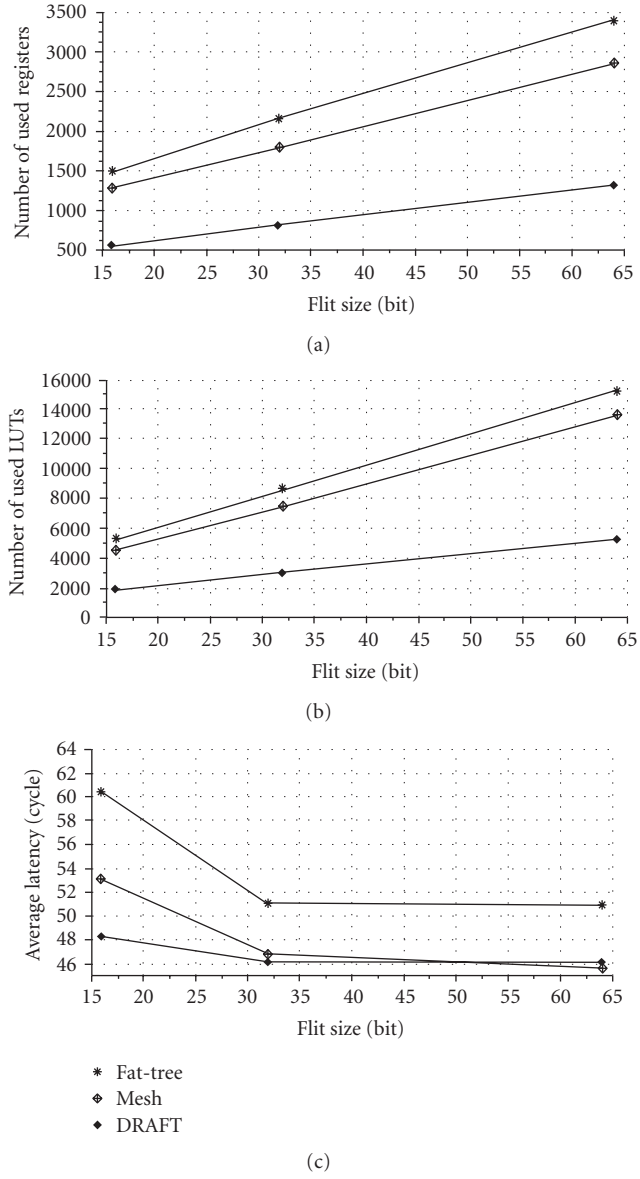


Figure 15: Influence of the flit sizes over the hardware resources (registers (a) and LUTs (b)) and the latencies (c) with a data rate of 400 Mbit/s per CE.

average cycles, respectively, for 32 and 64 bits in DRAFT. This phenomenon can also be observed for the fat-tree and the mesh. Considering the hardware cost of the flit size, this parameter should be minimized according to the desired performances.

Similarly, the impact of the buffer depth is presented in Figure 16. The presented results were obtained with a flit width of 32 bits and with a data rate of 800 Mbit/s per CE.

Concerning the buffer depth, an increase of the depths decreases the average latencies, but its impact over the hardware resources is lower than for the flit size. So, if the flit size is set to a minimum, the buffer depth can be increased in order to reach the network performances and the scalability required by the application.

In conclusion over the scalability, every topology can support a relatively low number of simultaneously connected CEs at full network performances. In order to increase this number, the designer should reduce first the data rate of its CEs. This point is particularly true with the DRAFT topology. If it is not possible, the designer should try to increase the flit size and buffer depth parameters. However, these solutions have an important impact on resource

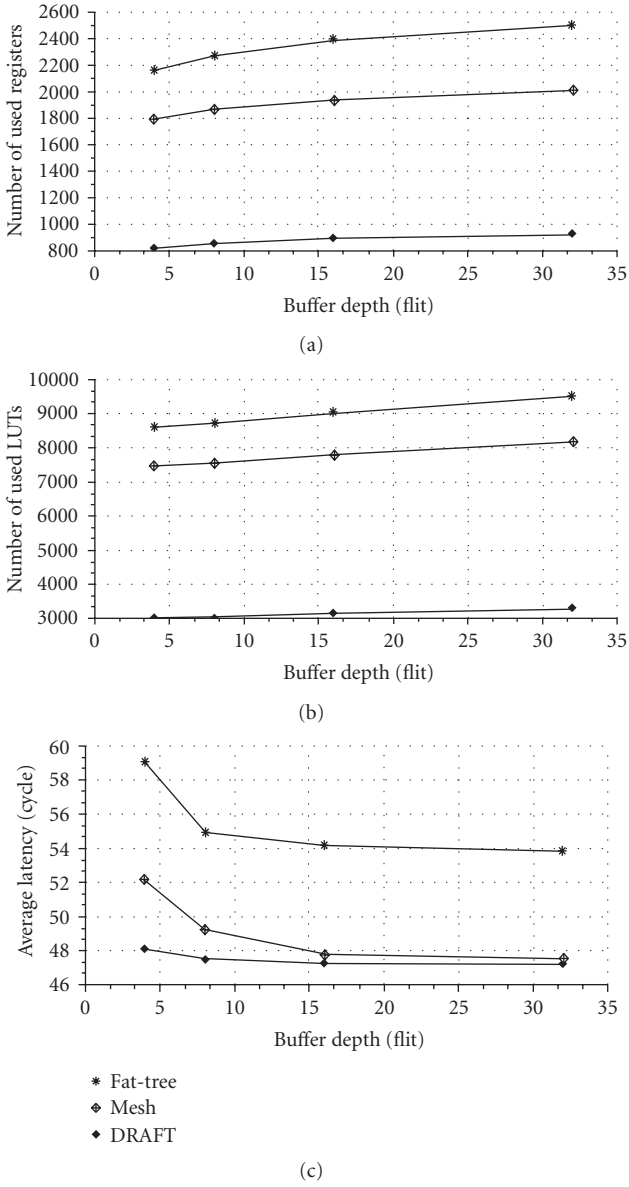


Figure 16: Influence of the buffer depths over the hardware resources (registers (a) and LUTs (b)) and the latencies (c) with a data rate of 800 Mbit/s per CE.

overhead. The use of virtual channels can also increase the scalability of the networks, but the impact over hardware resources is impractical for an implementation in an FPGA.

6.4. Type of Data Traffic. In this part, the influence of the data traffic is presented. Each network is designed to connect 8 CEs with a flit size of 32 bits and a buffer depth of 4 flits. Three types of traffic are studied. Thus, data are produced with a uniform, a normal, or a "pareto on/off" distribution. This latter is a periodic distribution where a period without any emission of data is followed by a period of emission. Results are presented in Table 2.

From these results, DRAFT appears to better support the different types of traffic, even at maximum data rate.

Table 2: Influence of the different data traffics over the latency (average clock cycles).

Type of traffic	DRAFT	Fat-tree	Mesh
Uniform (800 Mbit/s)	48.27	58.22	51.91
Normal (800 Mbit/s)	43.40	56.71	49.21
Pareto (800 Mbit/s)	33.11	42.42	38.40
Uniform (400 Mbit/s)	46.20	51.07	46.87
Normal (400 Mbit/s)	35.58	43.28	38.65
Pareto (400 Mbit/s)	31.43	39.83	35.64

A uniform traffic can be encountered in many applications using an constant flow of data like video processing. The normal repartition of the data rates during computation time is required by applications which depend on the received data like the automatic recognition of a target. The "pareto on/off" corresponds to the traffic between a hardware element and a shared memory using the burst mode, which is the continuous emission of several data during a short period of time. Thus, DRAFT can support all these applications with better performances than other networks.

7. Implementation of an Application Using DRAFT

In this section, the implementation results (hardware resources and network performances) of an application, designed into the framework of the FOSFOR project, are presented. This application is implemented into a middle size Xilinx Virtex5 FPGA: the XC5VSX50T.

A system using DRAFT to interconnect 4 PRRs and 4 shared IOs is implemented with a MicroBlaze processor to control the DPR [31]. The application realizes a target tracking in a video stream. For this purpose, the application is composed of statically implemented tasks, which transform the video stream, and of dynamically implemented ones, which realize the tracking. Dynamically implemented tasks depend on the dynamic number of targets and on the nature of these targets. Thus, these tasks are very heterogeneous in terms of hardware resource requirements. The four shared IOs are located in the central column of the FPGA. This system operates at 100 MHz with a 32 bits data width. In this Virtex 5, DRAFT needs only 2% of the registers and 10% of the LUTs. So, while including the MicroBlaze processor with its memory and peripherals for the DPR management, 92% of the registers and 85% of the LUTs (also with 88% of the BRAMs) remain free for task implementation. Complete implementation results are shown Table 3.

Concerning network performances, the routers are implemented with a critical path of 9,09 ns (110 MHz). DRAFT presents an average latency of 46 clock cycles. In this implementation without virtual channels, it also offers an aggregative bandwidth of 880 MByte/s. A view of the hardware implementation of the system is presented in Figure 17.

Table 3: Complete implementation results of DRAFT interconnecting 4 PRRs and 4 shared IOs.

	Draft alone			Global system		Free space
	Used	Total	%	Used	%	%
Registers	857	32640	2%	2223	6%	92%
LUT	3470	32640	10%	5150	15%	85%
LUT FLIP FLOP	3497	32640	10%	5560	17%	83%
DSP48E	0	288	0%	3	1%	99%
BRAM36	0	132	0%	16	12%	88%

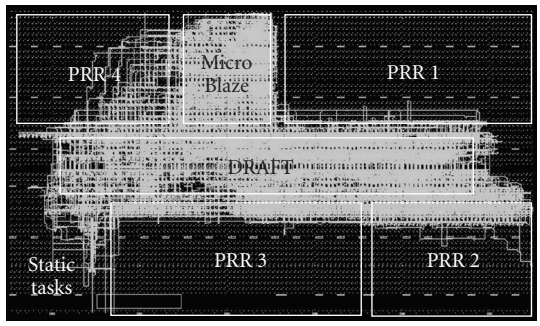


Figure 17: 90-degree rotated view of DRAFT connecting 4 PRRs and 4 shared IOs.

8. Conclusion

In this article, a flexible interconnection network is described. This network is compliant with applications requiring the DPR, and with current FPGA technologies. Thus, from the comparison of current interconnections, even bus-based or NoC based, the fat-tree appeared as a particularly well suited topology for the compliance with the DPR paradigm. Indeed, this structure offers higher network performances than other topologies in terms of bandwidth and latency. A fat-tree is an indirect network that provides high flexibility at the data path level, and supports the parallelization of the communications. Its structure allows interconnecting heterogeneous CEs in heterogeneous FPGAs. The main drawback of this topology is the hardware resource requirements. Hence, the DRAFT flexible network is proposed. DRAFT is indeed the sum of several concepts concerning the structure and the implementation of a fat-tree. These concepts are proposed in order to significantly reduce the resource consumption, and to obtain a network fully compliant with the DPR paradigm. The main idea of DRAFT consists in connecting half of the CEs directly to the root of a fat-tree. The connection of the static elements and the unshared resources is also presented in order to reduce the number of routers, and so the resource consumption. Then, the way to implement DRAFT as a central column into an FPGA is proposed for taking advantage of current FPGA structures.

The DRAGOON generator is designed to parameterize and to automatically generate the DRAFT topology. DRAGOON also supports the simulation of the network allowing to characterize its performances. According with these concepts and thanks to DRAGOON, the DRAFT viability is demonstrated by the comparison with a fat-tree and a mesh network. DRAFT needs fewer resources and fewer communication links than a mesh and a fat-tree. DRAFT presents a lower average latency than every other topology, and supports higher transfer and injection rates (until 1000 Mbit/s). DRAFT is also less sensitive to the flit sizes and the buffer depths than the others so that it can be implemented minimizing its hardware requirements according with the application. Consequently, DRAFT is very well adapted for an implementation into the framework of applications using DPR where there are around 10 simultaneously connected CEs. Finally, the DRAFT viability in terms of compliance with current applications using DPR, and with current technologies, is demonstrated by the implementation of a target tracking application in a Xilinx Virtex5.

Acknowledgment

This research was supported by the ANR (French National Research Agency) within the framework of the FOSFOR project (Flexible OS FOr Reconfigurable devices), http://www.polytech.unice.fr/~fmuller/fosfor/.

References

[1] FOSFOR, http://users.polytech.unice.fr/~fmuller/fosfor/.

[2] D. Cozzi, C. Far, A. Meroni, V. Rana, M. D. Santambrogio, and D. Sciuto, "Reconfigurable NoC design flow for multiple applications run-time mapping on FPGA devices," in *Proceedings of the 19th ACM Great Lakes Symposium on VLSI (GLSVLSI '09)*, pp. 421–424, Boston, Mass, USA, May 2009.

[3] Altera, "Stratix IV Device Handbook—Volume 1," ver 4.0, November 2009.

[4] Altera, "Stratix IV Device Handbook—Volume 2," ver 4.0, November 2009.

[5] ATMEL, "AT40K05/10/20/40AL. 5K–50K Gate FPGA with DSP Optimized Core Cell and Distributed FreeRam, Enhanced Performance Improvement and Bi-directional I/Os (3.3 V)," revision F, 2006.

[6] Xilinx, "Virtex-5 FPGA Configuration User Guide," v3.5, 2008.

[7] "PlanAhead User Guide—version 1.1," Xilinx, 2008.

[8] Xilinx, "Difference-Based Partial Reconfiguration, Application Note XAPP290," 2007.

[9] F. Moraes, N. Calazans, L. Mller, E. Brio, and E. Carvalho, "Dynamic and partial reconfiguration in FPGA SoCs: requirements tools and a case study," in *New Algorithms, Architectures and Applications for Reconfigurable Computing*, pp. 157–168, Springer, New York, NY, USA, 2005.

[10] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 438–452, 2007.

[11] D. Koch, C. Beckhoff, and J. Teich, "Recobus-builder—a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 119–124, Heidelberg, Germany, September 2008.

[12] A. Thomas and J. Becker, "Dynamic adaptive runtime routing techniques in multigrain reconfigurable hardware architectures," in *Field Programmable Logic and Application*, vol. 3203 of *Lecture Notes in Computer Science*, pp. 115–124, Springer, Berlin, Germany, 2004.

[13] S. Winegarden, "Bus architecture of a system on a chip with user-configurable system logic," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 3, pp. 425–433, 2000.

[14] T. Seceleanu, J. Plosila, and P. Liljeberg, "On-chip segmented bus: a self timed approach," in *Proceedings of the Annual IEEE International ASIC/SOC Conference—System-on-Chip in a Networked World*, pp. 216–220, September 2002.

[15] A. Ahmadinia, C. Bobda, J. Ding, et al., "A practical approach for circuit routing on dynamic reconfigurable devices," in *Proceedings of the International Workshop on Rapid System Prototyping (RSP '05)*, pp. 84–90, Montreal, Canada, June 2005.

[16] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[17] E. Salminen, A. Kulmala, and T. D. Hamalainen, "Survey of network-on-chip proposals," http://www.ocpip.org/white_papers.php.

[18] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.

[19] C. Bobda and A. Ahmadinia, "Dynamic interconnection of reconfigurable modules on reconfigurable devices," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 443–451, 2005.

[20] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. Fekete, and J. van der Veen, "DyNoC: a dynamic infrastructure for communication in dynamically reconfigurable devices," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*, vol. 2005, pp. 153–158, Tampere, Finland, August 2005.

[21] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to networks-on-chips," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 155–160, Madrid, Spain, August 006.

[22] "Cell Broadband Engine Programming Handbook," IBM, version, 1.11, 2008.

[23] C. Neeb and N. Wehn, "Designing efficient irregular networks for heterogeneous systems-on-chip," *Journal of Systems Architecture*, vol. 54, no. 3-4, pp. 384–396, 2008.

[24] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, 2005.

[25] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, pp. 71–121, 2006.

[26] J. L. Hennessy and D. A. Patterson, "Appendix E: interconnection networks," in *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, Calif, USA, 2006.

[27] H. Kariniemi and J. Nurmi, "Reusable XGFT interconnect IP for network-on-chip implementations," in *Proceedings of the International Symposium on System-on-Chip*, pp. 95–102, Tampere, Finland, November 2004.

[28] H. Kariniemi, *On-line reconfigurable extended generalized fat tree network-on-chip for multiprocessor system-on-chip circuits*, Ph.D. dissertation, Tampere University of Technology, Tampere, Finland, 2006.

[29] "Synthesis and Simulation Design Guide—ISE 9.2i," Xilinx, 2008.

[30] "ModelSim LE/PE Users Manual 6.5.c," Mentor graphics, 2009.

[31] K. Park and H. Kim, "Remote FPGA Reconfiguration Using MicroBlaze or PowerPC Processors," Application Note: XAPP441 (v1.1) ed., Xilinx.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 454506, 11 pages
doi:10.1155/2010/454506

Research Article

Parameterized Hardware Design on Reconfigurable Computers: An Image Processing Case Study

Miaoqing Huang,[1] Olivier Serres,[2] Tarek El-Ghazawi,[2] and Gregory Newby[3]

[1] *Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA*
[2] *Department of Electrical and Computer Engineering, The George Washington University, Washington, DC 20052, USA*
[3] *Arctic Region Supercomputing Center, University of Alaska Fairbanks, Fairbanks, AK 99775, USA*

Correspondence should be addressed to Miaoqing Huang, mqhuang@uark.edu

Received 1 July 2009; Accepted 10 February 2010

Academic Editor: Elías Todorovich

Reconfigurable Computers (RCs) with hardware (FPGA) co-processors can achieve significant performance improvement compared with traditional microprocessor (μP)-based computers for many scientific applications. The potential amount of speedup depends on the intrinsic parallelism of the target application as well as the characteristics of the target platform. In this work, we use image processing applications as a case study to demonstrate how hardware designs are parameterized by the co-processor architecture, particularly the data I/O, i.e., the local memory of the FPGA device and the interconnect between the FPGA and the μP. The local memory has to be used by applications that access data randomly. A typical case belonging to this category is image registration. On the other hand, an application such as edge detection can directly read data through the interconnect in a sequential fashion. Two different algorithms of image registration, the exhaustive search algorithm and the Discrete Wavelet Transform (DWT)-based search algorithm, are implemented on hardware, i.e., Xilinx Vertex-IIPro 50 on the Cray XD1 reconfigurable computer. The performance improvements of hardware implementations are 10× and 2×, respectively. Regarding the category of applications that directly access the interconnect, the hardware implementation of Canny edge detection can achieve 544× speedup.

1. Introduction

Reconfigurable Computers (RCs) are traditional computers extended with co-processors based on reconfigurable hardware like FPGAs. Representative RC systems include SGI RC100 [1], SRC-6 [2], and Cray XD1 [3]. These enhanced systems are capable of providing significant performance improvement for scientific and engineering applications [4]. The performance of a hardware design on an FPGA device depends on both the intrinsic parallelism of the design as well as the characteristics of the FPGA co-processor architecture, which consists of the FPGA device itself and the surrounding data interface. Due to the limited size of the internal Block RAM memory, it is not applicable to store large amounts of data inside the FPGA device. Therefore external, however, local SRAM modules are generally connected to the hardware co-processor for data storage, such as the example shown in Figure 1. Furthermore, an FPGA co-processor can directly access the host memory through the interconnect, which generally provides a sustained bandwidth up to several GB/s. The available number and data width of local memory banks and the interconnect channels play important roles in the hardware implementation on an FPGA device and decide the parallelism a design can achieve in many cases. In this work, image processing algorithms are adopted as a case study to demonstrate how hardware designs can be parameterized by the data I/O of the FPGA co-processor in order to achieve the best performance.

Image processing applications are capable of gaining a significant performance improvement with hardware design [5, 6]. Two categories of image processing applications are selected to represent hardware designs that present different data I/O requirements. The first category of applications is image registration, which requires the use of local memory

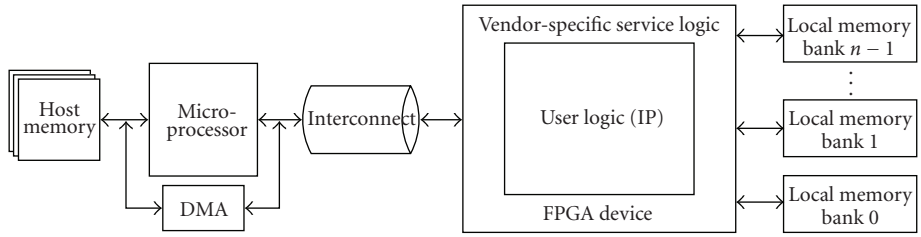


Figure 1: General architecture of a reconfigurable computer.

for data access. The second category of applications is edge detection, which directly reads from or writes to the interconnect in a streaming fashion. Image registration is a very important image processing task. It is used to align or match pictures taken in different conditions (at a different time, angle, or from different sensors). A vast majority of automatic image processing systems require the use of image registration processes. Common image registration applications include target recognition (identification of a small target inside a much bigger image), satellite image alignment in order to detect changes such as land usage or forest fires [7], matching stereo images to recover depth information, and superposing medical images taken at different moments for diagnosis [8, 9]. As an example of the applications in the second category, edge detectors encompass image processing algorithms that identify the positions of edges in an image. Edges are discontinuities in terms of intensity or orientation or both and generally represent meaningful characteristics of the image (boundaries of objects, e.g.). Commonly, edge detectors are used to filter relevant information in an image. Thus, they greatly reduce the amount of processing needed for interpreting the information contents of an image. One of the most important edge detection algorithms is the Canny edge detection [10]. The Canny edge detection operator was developed by Canny in 1986 and uses a multistage algorithm to detect a wide range of edges in images. It remains until now, as a state-of-the-art edge detector used in many applications.

The implementation of image registration and edge detection on reconfigurable computers has been previously reported in [11, 12], respectively. In this paper, we not only present the detail of hardware design itself, but also exploit the role of data I/O of the FPGA co-processor in the design process. More precisely, we demonstrate how the design of image processing applications is parameterized by local memory architecture and DMA interface on reconfigurable computers. Furthermore, the hardware processing time of both applications are formalized in terms of clock cycles. The remaining text is organized as follows. In Section 2, we discuss the related work based on literature survey. In Section 3, two related image registration algorithms, the exhaustive search algorithm and the DWT-based search algorithm, and their hardware implementations are discussed. Section 4 focuses on the design of Canny edge detection in hardware. The implementation of all applications on the Cray XD1 reconfigurable computer is presented in Section 5. Finally, Section 6 concludes this work.

2. Related Work

Since image registration is a computation-demanding process in general, hardware (e.g., FPGA device) is leveraged to improve the processing performance. In [8], Dandekar and Shekhar introduced an FPGA-based architecture to accelerate mutual information-(MI)-based deformable registration during computed tomography-(CT)-guided interventions. Their reported implementation was able to reduce the execution time of MI-based deformable registration from hours to a few minutes. Puranik and Gharpure presented a multilayer feedforward neural network (MFNN) implementation in Xilinx XL4085 for template search in standard sequential scan and detect (SSDA) image registration [13]. In [14], Liu et al. proposed a PC-FPGA geological image processing system in which the FPGA was used to implement Fast Fourier Transform-(FFT)-based automatic image registration. In [15], El-Araby et al. prototyped an automatic image registration methodology for remote sensing using a reconfigurable computer. However, these previous work only emphasized the image registration algorithm itself. The FPGA data I/O as a factor in the design was not discussed in detail.

Low-level image processing operators, such as digital filters, edge detectors and digital transforms are good candidates for hardware implementation. In [16], a generic architectural model for implementing image processing algorithms of real-time applications was proposed and evaluated. In [17], a Canny edge detection application written in Handel-C and implemented in the FPGA device was discussed. The proposed architecture is capable of producing one edge-pixel every clock cycle. The work in [18] illustrated how to use design patterns in the mapping process to overcome image processing constraints on FPGAs. However, most of the previous works, for example, [16–18], only focused on the algorithms alone and did not consider the platform characteristics as a factor in the design.

3. Image Registration

In this section, we discuss how to implement image registration algorithms in hardware to exploit the processing parallelism, which is bounded by the local memory architecture.

3.1. Background. Image registration can be defined as a mapping between two images, the reference image R and the test image T, both spatially and with respect to the intensity

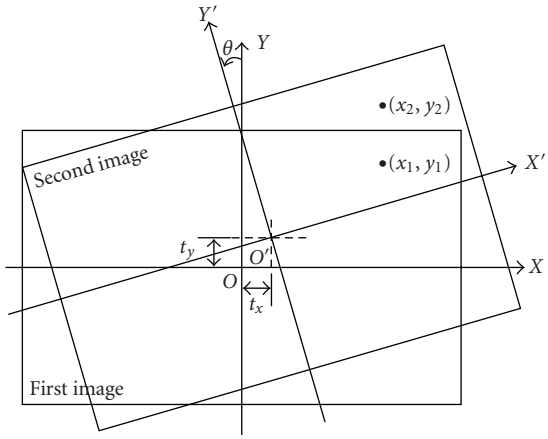


Figure 2: Rigid-body transformation (scale = 1).

[19]. If these images are defined as two 2D arrays of a given size denoted by I_1 and I_2 where $I_1(x, y)$ and $I_2(x, y)$ each map to their respective intensity values, then the mapping between images can be expressed as

$$I_2(x, y) = g(I_1(f(x, y))), \tag{1}$$

where f is a 2D spatial-coordinate transformation and g is a 1D intensity or radiometric transformation. More precisely, f is a transformation that maps two spatial coordinates, x and y, to new spatial coordinates x' and y':

$$(x', y') = f(x, y). \tag{2}$$

g is used to compensate gray value differences caused by different illuminations or sensor conditions.

According to [19], image registration can be viewed as the combination of four components:

(1) *a feature space*, that is, the set of characteristics used to perform the matching and which are extracted from the reference and test images;

(2) *a search space*, that is, the class of potential transformations that establish the correspondence between the reference and test images;

(3) *a search strategy*, which is used to decide how to choose the next transformation from the search space;

(4) *a similarity metric*, which evaluates the match between the reference image and the transformed test image for a given transformation chosen in the search space.

The fundamental characteristic of any image registration technique is the type of spatial transformation or mapping used to properly overlay two images. The most common transformations are rigid-body, affine, projective, perspective, and global polynomial. Rigid-body transformation is composed of a combination of a rotation (θ), a translation (t_x, t_y), and a scale change (s). An example is shown in Figure 2. It typically has four parameters, t_x, t_y, s, θ, which map a point (x_1, y_1) of the first image to a point (x_2, y_2) of the second image as follows:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \quad \text{or} \quad \overline{p}_2 = \overline{t} + s\mathcal{R}\overline{p}_1, \tag{3}$$

where $\overline{p}_1$ and $\overline{p}_2$ are the coordinate vectors of the two images; $\overline{t}$ is the translation vector; s is a scalar scale factor; $\mathcal{R}$ is the rotation matrix. Since the rotation matrix $\mathcal{R}$ is orthogonal, the angles and lengths in the original image are preserved after the registration. Because of the scalar scale factor s, rigid-body transformation allows changes in length relative to the original image, but they are the same in both x and y axes. (Please note both (x_1, y_1) and (x_2, y_2) are coordinates in the same Cartesian coordinate system with the origin O.)

Computing the correlation coefficient is the basic statistical approach to registration and is often used for template matching or pattern recognition. A correlation coefficient is a similarity measure or match metric, that is, it gives a measure of the degree of similarity between a template (the reference image) and an image (the transformed test image). The correlation coefficient between the reference image R and the image T', which is the test image after rigid-body transformation, is given as

$$\frac{\sum_{x,y}(R(x, y) - \mu_R)(T'(x, y) - \mu_{T'})}{\sqrt{\sum_{x,y}(R(x, y) - \mu_R)^2 \sum_{x,y}(T'(x, y) - \mu_{T'})^2}}, \tag{4}$$

where μ_R and $\mu_{T'}$ are mean of the image R and T'. If the image R matches T', the correlation coefficient will have its peak with the corresponding transformation. Therefore, by computing correlation coefficients over all possible transformations, it is possible to find the transformation that yields the peak value of the correlation coefficient.

In this work, rigid-body transformation is selected for the registration between two images and the correlation coefficient is used to measure the similarity. Further, we assume that both the reference image and the test image are 8-bit grayscale and share the same size.

Given a search space, ($\Delta\Theta$,ΔX,ΔY) theoretically all tuples of (θ,t_x,t_y) are to be tested to find the tuple that generates the maximum correlation coefficient between the reference image and the transformed test image (In this work, the scale factor s is fixed at 1.). Figure 3 shows the two steps to test each tuple. The first step is to apply a rigid-body transformation on the test image T to get T'. The second step is to calculate the correlation coefficient between T' and the reference image R. As shown in Figure 3(b), only the pixels of both images within the shaded region are used during the calculation. Since the test image T is rotated and translated to obtain T', some parts of T' are beyond the shaded region. In other words, some portions of the shaded region (shown as crossed regions in the left Cartesian coordinate system of Figure 3(b)) do not belong to T'. For the pixels belonging to these crossed regions, their values are treated as zeros in the calculation.

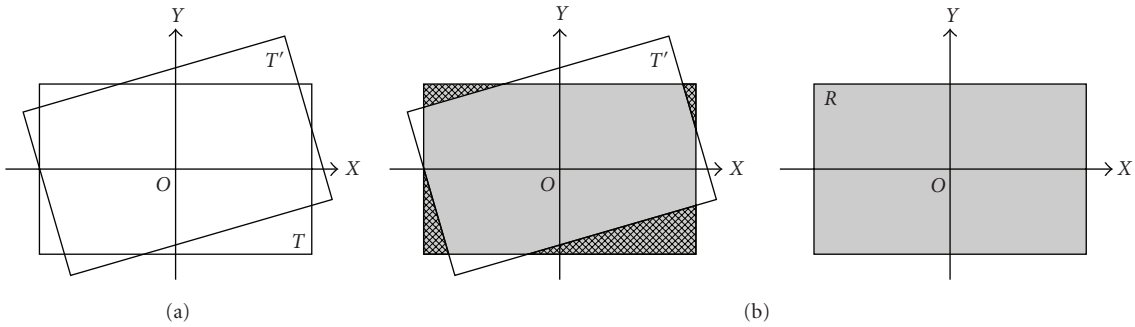


Figure 3: Two steps in image registration: (a) rigid-body transformation on the test image T, (b) calculate correlation coefficient between the transformed test image T' and the reference image R.

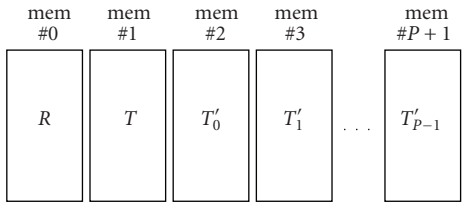


Figure 4: Local memory data storage layout in the exhaustive search algorithm for image registration.

In the remaining part of this section, two different approaches based on rigid-body transformation are discussed in details. The first approach literally tests the whole search space to find the best tuple of (θ,t_x,t_y). The second approach applies DWT on both the reference image and the test image to reduce the search resolutions in order to improve the search efficiency.

3.2. Exhaustive Search Algorithm. As its name implies, the exhaustive search algorithm tests all possible tuples of (θ,t_x,t_y) with a fixed search resolution, δ_θ, δ_x, and δ_y, on each dimension, respectively, in order to find the tuple that produces the highest correlation coefficient between the transformed test image and the reference image. If this algorithm is implemented on a scalar microprocessor, these tuples have to be tested in sequence. However, if the same algorithm is implemented in hardware, multiple tuples can be tested in parallel to improve the performance.

Since the size of an image is normally bigger than the amount of available Block RAM inside an FPGA device, the local external memory is used to store images. Assuming there are $P + 2$ individual local memory banks connected directly to the FPGA device, and one bank keeps the reference image R, one bank keeps the test image T. The other remaining P banks are used to store P transformed test images $T's$ using different tuples of (θ,t_x,t_y), as shown in Figure 4. If we further assume that each memory bank has its own independent read and write ports, P transformations of the test image can be carried out concurrently. The calculation of correlation coefficients between the image R and P different $T's$ can be performed in parallel as well.

Given the coordinate of one pixel in the original image, (3) is used to calculate the coordinate of the corresponding pixel in the transformed image. Then, the intensity of the pixel (x_1, y_1) in T can be written into T' at the coordinate of (x_2, y_2). If we assume that there are S pixels in the original image and the hardware implementation is fully pipelined, the transformation step would take approximately S clock cycles. Furthermore, some extra clock cycles are needed to initialize the intensities of all pixels within the shaded region to zero due to two reasons. First, there are several regions within which the pixels do not belong to T', as shown in Figure 3(b). Second, there may exist artifacts whose coordinates are within both the shaded region and T', but are not calculated due to discretization, as shown in Figure 5. If the intensities of these pixels are left randomly, it may affect the accuracy of the correlation coefficient. Therefore, it is necessary to initialize the intensities of all pixels in the shaded region to zero in the first place. Since the data width of the memory bank's access ports is multiple-byte, multiple pixels can be initialized in one clock cycle. If we assume that the data width is D-byte, then the initialization process would take roughly S/D clock cycles. Overall, the transformation step of P tuples would take $((D + 1)/D)S$ clock cycles. The mean intensity $\mu_{T'}$ of each T' can be calculated during the transformation step, hence it takes no extra time. The mean intensity μ_R can be precalculated by the microprocessor and forwarded to the FPGA device later since it remains unchanged during the whole image registration process.

Although the calculation of the correlation coefficient as (4) between R and $T's$ is more complicated than the transformation step, it takes the same time as the initialization process since D pixels can be read and processed in the same clock cycle. Altogether, these three steps, including initialization, transformation and correlation coefficient calculation, would take $((D + 2)/D)S$ clock cycles for testing P tuples of (θ,t_x,t_y). If the entire search space consists of $\Delta\Theta * \Delta X * \Delta Y$ tuples, the whole registration process would take

$$\frac{\Delta\Theta * \Delta X * \Delta Y * (D+2)}{PD} S \tag{5}$$

clock cycles. Apparently, the image registration time in hardware can be significantly reduced by increasing the number of local memory banks. Widening the data width of

(a) (b)

Figure 5: Artifacts due to discretization in rigid-body transformation ((a) the original image; (b) the transformed image).

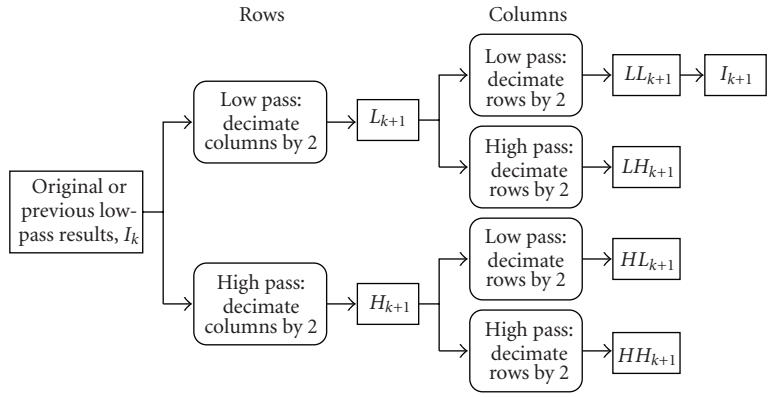


Figure 6: DWT decomposition of an image.

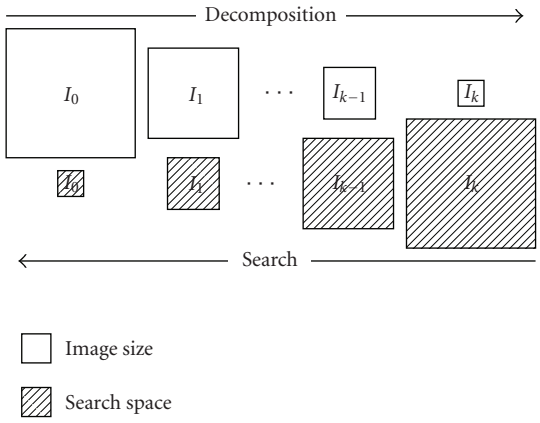


Figure 7: Decrease the search space and increase the search resolution in the search process.

access port of local memory can improve the performance as well; however, it can also hit the upper bound very quickly due to the fact that

$$\lim_{D \to \infty} \frac{D+2}{D} = 1. \tag{6}$$

3.3. DWT-Based Search Algorithm. Although the exhaustive search algorithm is quite straightforward, it is computation-demanding as well. In [20], a DWT-based image registration approach was proposed. As shown in Figure 6, both the test image and the reference image go through several levels of Discrete Wavelet Transform before applying image registration. After each level of DWT, the image size is shrunk to 1/4 of the previous level. In the meantime the image resolution is reduced to half. For example, if k levels of DWT are applied on both the test image T_0 and the reference image R_0, two series of images, $T_0, T_1, \ldots, T_k$, and $R_0, R_1, \ldots, R_k$, are obtained. The registration process starts from the exhaustive search between T_k and R_k among the search space of $(\Delta\Theta, \Delta X, \Delta Y)$ with the search resolution, $2^k * \delta_\theta$, $2^k * \delta_x$, and $2^k * \delta_y$, on each dimension. The registration result between T_k and R_k, $(\theta_k, t_{x_k}, t_{y_k})$ becomes the center of the search space of the registration between T_{k-1} and R_{k-1}. In other words, the registration between T_{k-1} and R_{k-1} is among the search space of $(\theta_k \pm 2^k * \delta_\theta,\ t_{x_k} \pm 2^k * \delta_x,\ t_{y_k} \pm 2^k * \delta_y)$. However, the search resolution is increased to $2^{k-1} * \delta_\theta$, $2^{k-1} * \delta_x$, and $2^{k-1} * \delta_y$, on each dimension. In general, when the registration process traces back one level, the search scope is reduced to half on

Table 1: Search strategy summary for rotation.

Decomposition Level	Search Space	Search Resolution	Result
k	$\Delta\Theta$	$2^k * \delta_\theta$	θ_k
$k-1$	$[\theta_k - 2^k * \delta_\theta; \theta_k + 2^k * \delta_\theta]$	$2^{k-1} * \delta_\theta$	θ_{k-1}
$k-2$	$[\theta_{k-1} - 2^{k-1} * \delta_\theta; \theta_{k-1} + 2^{k-1} * \delta_\theta]$	$2^{k-2} * \delta_\theta$	θ_{k-2}
$\cdots$	$\cdots$	$\cdots$	$\cdots$
2	$[\theta_3 - 2^3 * \delta_\theta; \theta_3 + 2^3 * \delta_\theta]$	$2^2 * \delta_\theta$	θ_2
1	$[\theta_2 - 2^2 * \delta_\theta; \theta_2 + 2^2 * \delta_\theta]$	$2 * \delta_\theta$	θ_1
0	$[\theta_1 - 2 * \delta_\theta; \theta_1 + 2 * \delta_\theta]$	δ_θ	θ_0^*

* The final output.

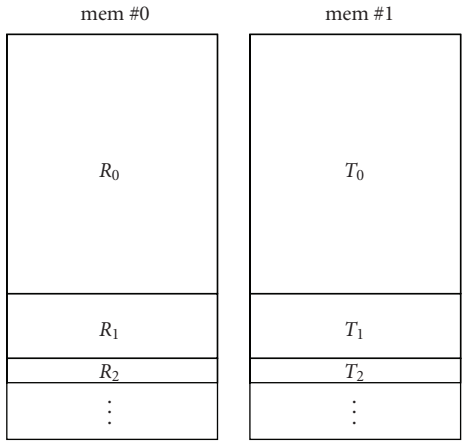


Figure 8: Store the original and decomposed images in the same memory bank in the DWT-based image registration.

each dimension, and the search resolution is increased two times on each dimension, respectively. This search strategy is illustrated in Figure 7. Table 1 details the search space and search resolution at each step in which rotation is taken as an example.

Different DWT decomposition processes have to be carried out in a sequence. Similarly the search processes at different levels need to be performed one after the other. Due to these two reasons, the original image and the decomposed images can be stored in the same memory bank, as shown in Figure 8 in which R_k or T_k denote the decomposed image at the level k.

If we use the same assumptions as in Section 3.2, that is, both original images consist of S pixels, the data width of the local memory is D-byte, and there are $P+2$ independent local memory banks, then the DWT decomposition step alone will take

$$\frac{S}{D}\left(1+\frac{1}{4}+\cdots+\left(\frac{1}{4}\right)^{k-1}\right)=\frac{4S}{3D}\left(1-\left(\frac{1}{4}\right)^{k}\right) \quad (7)$$

clock cycles.

The search between the decomposed reference image and the test image at each level can use the same method described in Section 3.2. By observing Table 1, it is found that the search space at each level, except the level k, is 125 tuples of (θ,t_x,t_y). Therefore, the search from level 0 to level $k-1$ will take

$$\begin{aligned}&\frac{125S(D+2)}{PD}\left(1+\frac{1}{4}+\cdots+\left(\frac{1}{4}\right)^{k-1}\right)\\&=\frac{500S(D+2)}{3PD}\left(1-\left(\frac{1}{4}\right)^{k}\right)\end{aligned} \quad (8)$$

clock cycles. The search at the level k itself takes

$$\frac{\Delta\Theta * \Delta X * \Delta Y * (D+2)}{32^k \delta_\theta \delta_x \delta_y PD} S \quad (9)$$

clock cycles.

4. Edge Detection

Edge detection aims at identifying pixels in a digital image at which the image brightness changes sharply, that is, having discontinuities. Most edge detection algorithms involve the convolution process between the image and a kernel. Convolution provides a way of "multiplying together" two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values. In an image processing context, one of the input arrays is normally just a grayscale image. The second array is usually much smaller, and is also two dimensional (although it may be just a single coefficient). The second array is always known as the *kernel*, as shown in Figure 9. If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will consist of $M-m+1$ rows and $N-n+1$ columns. Mathematically we can write the convolution between the image I and the kernel K as

$$O(x_1,y_1)=\sum_{x_2=0}^{m-1}\sum_{y_2=0}^{n-1} I(x_1+x_2,y_1+y_2)\times K(x_2,y_2), \quad (10)$$

where x_1 runs from 0 to $M-m$ and y_1 runs from 0 to $N-n$.

From Figure 9 and (10), we can find that the calculation of different pixels in the output image is independent to each other. Therefore, the intensities of multiple output pixels can be computed in parallel in a hardware design. Since the input

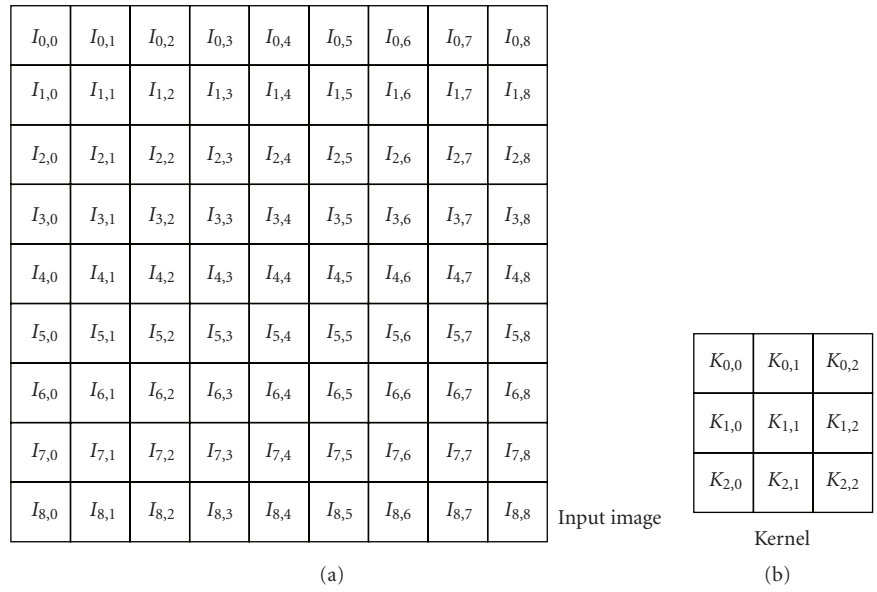


Figure 9: An example small image (a) and kernel (b) to illustrate convolution.

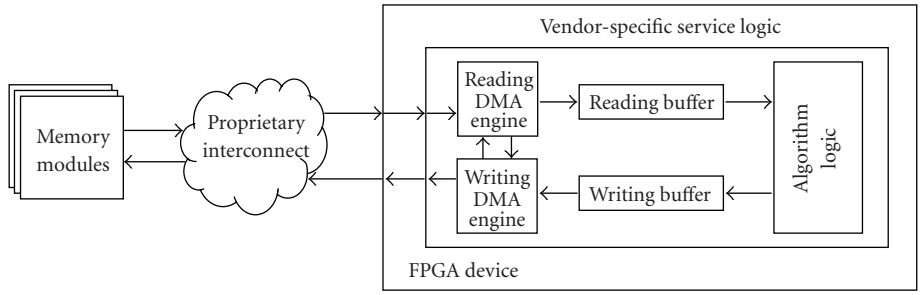


Figure 10: Streaming data transfer mode on reconfigurable computers.

image reading and the output image writing are both in a sequential mode, the data storage in local memory can be avoided. Instead, the user logic can access the interconnect directly for fetching the source image and storing the result image. In order to optimize the hardware performance, the interconnect and the user logic are chained into a pipeline, and the data run through the pipeline as a stream. We call this architecture Streaming Data Transfer Mode, shown in Figure 10. Two DMA engines work in parallel to retrieve raw data blocks from and return result data blocks to the main memory. Under ideal circumstances, the reading DMA engine receives one raw data block from the input channel and the writing DMA engine puts one result data block to the output channel every clock cycle.

Being one stage of the overall pipeline, the design of the algorithm logic is parameterized by the characteristics of other components in the pipeline, that is, the data width of the interconnect between the FPGA device and the μP. Because the data width of the interconnect fabric is multiple-byte wide in general, several pixels are fed into the algorithm logic in the same clock cycle. To maximize the throughput of the overall architecture, the algorithm logic has to be capable of performing the operations of multiple pixels concurrently and taking new data input every clock cycle. In the following discussion, we assume that (1) the image is in 8-bit grayscale, (2) the image size is $M \times N$ and the kernel size is $m \times n$, and (3) the data width of the interconnect is D-byte. Furthermore, pixels in the original image are delivered into the algorithm logic in a stream, starting with the pixel at the top-left corner, ending with the pixel at the bottom-right corner.

The diagram of the algorithm logic is shown in Figure 11. The architecture consists of four components, one Line Buffer, one Data Window, an array of PEs, and the Data Concatenating Block.

The quantity of PEs is D, that is, the data width of the interconnect in byte. Every PE is fully pipelined and is capable of taking a new input, that is, one block of $m \times n$ pixels, every clock cycle. The output of one PE is the intensity of one pixel in the result image.

The Data Window is a two-dimension register array of $m \times (n + D - 1)$ in charge of providing image blocks to the downstream PEs. Analogously, the Data Window scans the original image from left to right and from top to bottom

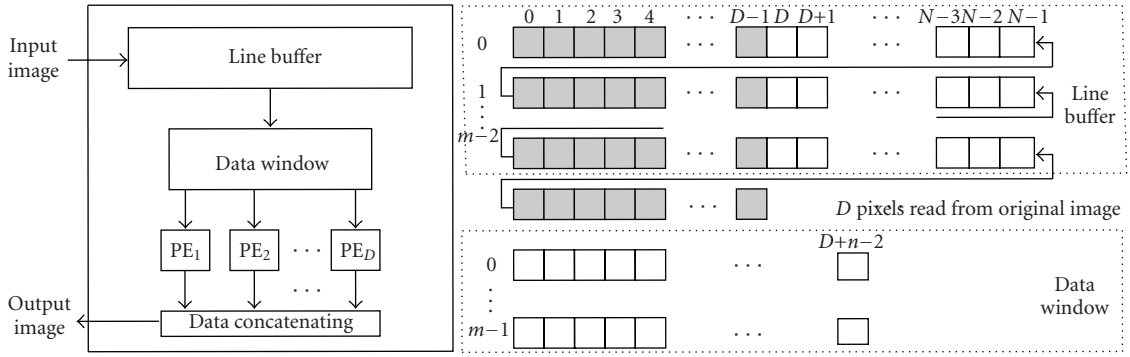


Figure 11: Diagram of the algorithm logic in edge detection.

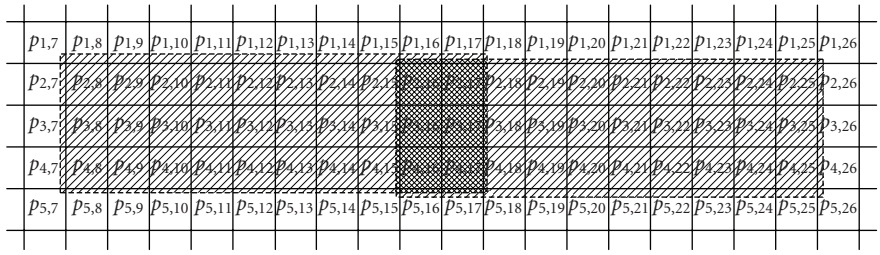


Figure 12: The content of Data Window in two consecutive steps (assume the kernel size is 3×3 and D is 8).

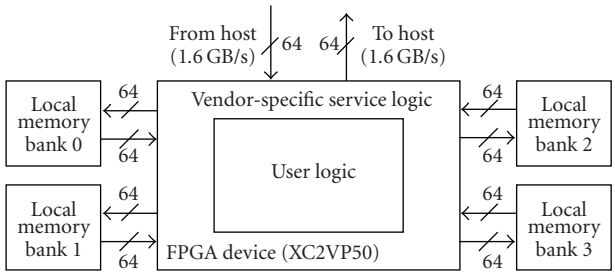


Figure 13: Cray XD1 FPGA local memory architecture.

with a horizontal stride of D pixels and a vertical stride of 1 pixel. Figure 12 demonstrates the scanning from left to right and shows the contents in two consecutive steps of the data window. Once the data window receives a valid input, that is, one block of $m \times D$ pixels, it does a D-byte left shift for all rows with the input tailing the right most $n - 1$ columns.

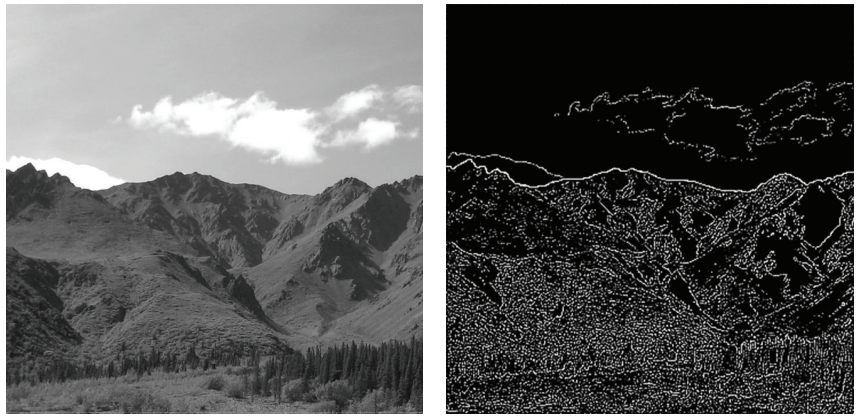
The Line Buffer is a two-dimension register array of $(m-1) \times N$. The original image is transferred into the FPGA as a stream. However, PEs request image blocks that spread different rows. The purpose of the line buffer is to keep all pixels in registers until one $m \times D$ block forms. One $m \times D$ block, shown in gray in Figure 11, comprises the new arrived D pixels and another $(m - 1) \times D$ pixels that reside at the head of every row of the line buffer. Every time the line buffer receives D pixels of the original image, it performs two actions simultaneously. The first action is to deliver the new formed image block to the data window. The second action is to do a D-byte left shift in a zigzag form, in which two neighbor rows are linked together by connecting their tail and head.

The design of the Data Concatenating Block is straightforward because what it does is to concatenate the outputs of the upstream PEs together. Once it receives valid output from the PEs, that is, D consecutive pixels in the output image, it sends them into the output channel.

In general, the four components in Figure 11 form a pipeline chain and each of them is fully pipelined as well. This architecture is able to accept D pixels of the original image every clock cycle and output D pixels of the result image every clock cycle at the same time. In case of a multiple-stage algorithm, such as the four-stage Canny edge detector, various stages can be chained together and each stage consists of these components with different parameters and functionalities. Under ideal scenario, it would take $(M \times N)/D$ clock cycles to perform an edge detection operation on an input image. However, the real performance is upper-bounded by the sustained bandwidth of the interconnect in general.

5. Implementation and Results

These two image registration algorithms and a Canny edge detection algorithm have been implemented on the Cray XD1 reconfigurable computer with Xilinx XC2VP50 FPGA devices. On the Cray XD1 platform, each FPGA device is connected to four local SRAM modules, 4 MB each, as shown in Figure 13. Every local memory module has separate reading and writing ports connected to the FPGA device, and is able to accept reading or writing transactions every



(a) (b)

Figure 14: The original image and the image after applying Canny edge detection.

Table 2: Performance improvement of image registration and Canny edge detection on Cray XD1.

Algorithm name		Computing time (s)		Speedup	Resource utilization	
		Opteron 2.4G	Cray XD1		Slices	Built-in multipliers
Image registration*	Exhaustive search	157.347	16.193	9.72	10.766 (45%)	42 (18%)
	DWT-based search†	1.298	0.829	1.57	20.205 (85%)	108 (55%)
		End-to-end throughput (MB/s)				
Canny edge detection		2.20	1.196	543.6	15.015 (63%)	200 (86%)

*The sizes of reference and test image are both 1024×1024. The search spaces of $\Delta\Theta$, ΔX, and ΔY are all from -8 to $+8$.
†Three levels of DWT are performed before the search, which is based on LL coefficients only.

clock cycle. Furthermore, the interconnect has separate input and output channels, both of them are 64-bit wide. The maximum operating clock rate for the user logic is 200 MHz, which is achieved in all our hardware implementations.

For the exhaustive search approach, the reference image and the test image are stored in two separate local memory modules. Every time, two possible tuples of (θ, t_x, t_y) are tested, that is, two different rigid-body transformations are performed on the test image simultaneously and produce two transformed test images, T' and T'', which are stored in two additional local memory modules separately. The calculation of the correlation coefficients between these two images and the reference image is carried concurrently as well. All the hardware modules are fully pipelined in our design in order to realize the full potential of a hardware implementation. Compared with the performance of the software implementation running on a single microprocessor, an AMD Opteron 2.4 GHz, the performance of the hardware implementation on a single FPGA device is approximately $10\times$ better, as shown in Table 2. The performance improvement of hardware implementation compared with software implementation is mainly due to the fully pipelined hardware design. For example, during the correlation coefficient calculation step, it only takes one clock cycle for hardware to calculate 8 pixels. On the other hand, the software implementation has to calculate one pixel each time. The measured time on the Cray XD1 is the end-to-end time including data transfer time between the μP and the FPGA and the hardware processing time on the FPGA. In the 16.193 s, data transfer time is merely 6 ms. Therefore, the performance of hardware implementation in this case is almost upper-bounded by the available number of local memory banks since only 45% of the FPGA slices are used. As shown in (5), the hardware processing time is reciprocal to the number of memory banks that are used to store transformed test images. Therefore the speedup of hardware implementation compared with software version is linearly proportional to the number of local memory banks.

For the DWT-based approach, three levels of DWT are applied on both the test image and the reference image before the registration process starts. All the original image and decomposed images of different levels are stored in local memory modules, as depicted in Figure 8. Since the whole process consists of two steps, the DWT decomposition and the search, the hardware utilization almost doubles in this case. More built-in multipliers are used in the DWT decomposition as well. Compared with the software implementation, the hardware is barely $2\times$ faster for the DWT-based search approach. The comparatively low speedup achieved by hardware in this case is due to the fact that the amount of computation is significantly reduced because of the DWT decomposition.

For both cases, the hardware implementation is mainly characterized by the local memory architecture, and the performance can be improved if more processing

concurrency is allowed, given that more independent local memory modules are available.

For the category of applications that use the interconnect for data access, a Canny edge detection algorithm is implemented following the architecture in Figure 11. The Canny edge detector comprises four stages in which each stage takes the output from the preceding stage and feeds its output to the following stage as follows:

$$\begin{aligned} \text{Input Image} &\longrightarrow \text{Gaussian Smoothing} \\ &\longrightarrow \text{Compass Edge Detecting} \\ &\longrightarrow \text{Non-maximum Suppression} \\ &\longrightarrow \text{Edge Tracingand Thresholding.} \end{aligned} \tag{11}$$

Because the interconnect is 8-byte wide, 8 edge detection operators are implemented and execute in parallel. Figure 14 shows the original image and the corresponding output after applying the Canny edge detection algorithm. In Canny edge detection algorithm, the processing of each pixel in the output image involves neighbor pixels in the input image. Furthermore, this processing consists of 4 stages and takes hundreds of clock cycles as latency. Due to the fully pipelined design in hardware implementation, the FPGA device is capable of computing 8 pixels in the output image every clock cycle no matter how complicated the computation of each pixel is. On the other hand, the pixels in the output image are computed one by one in the software implementation. Further, it would take thousands of cpu cycles to compute one pixel. Therefore hardware implementation of the Canny edge detection algorithm achieves 544× speedup compared with the corresponding software implementation. Higher speedup can be achieved if multiple images can be processed simultaneously given several interconnect channels are connected to the same FPGA co-processor and there are enough hardware resources to implement multiple instances of edge detection operators.

6. Conclusions

In this paper, we demonstrate how the parallelism of a hardware design on reconfigurable computers is parameterized by the co-processor architecture, particularly the number and the data width of local memory banks and the interconnect. Image registration algorithms based on rigid-body transformation are adopted as a case study to represent applications that use local memory. Two related; however, different algorithms, the exhaustive search algorithm and the DWT-based search algorithm, are described in detail. For the exhaustive search algorithm, the performance is linearly proportional to the available number of local memory banks. On the other hand, the DWT-based search algorithm improves the efficiency by applying DWT decomposition on both the reference and test images before the search in order to reduce the search scope. Compared with software implementations, hardware implementations of exhaustive search and DWT-based search achieve 10× and 2× speedup, respectively. For the category of applications that directly access the host interconnect, edge detection is selected as a case study. A streaming data transfer mode in which the user logic and the interconnect are chained into a pipeline is proposed. A user logic hardware architecture whose parallelism is decided by the data width of the interconnect is discussed in detail. A Canny edge detection application following the proposed architecture is capable of achieving 544× speedup compared with the corresponding software design.

As a future work, we will extend our work to the Tile 64 platform [21]. Parameters such as the external memory bandwidth and the intertile bandwidth will be taken into account to implement image processing algorithms. We expect to take advantages from Tile 64's capability of creating multicore pipelines internally.

Acknowledgment

This work was supported in part by Arctic Region Supercomputing Center (ARSC) at the University of Alaska Fairbanks.

References

[1] *Reconfigurable Application-Specific Computing User's Guide (007-4718-007)*, Silicon Graphics, 2008.

[2] *SRC Carte C Programming Environment v2.2 Guide (SRC-007-18)*, SRC Computers, 2006.

[3] *Cray XD1 FPGA Development (S-6400-14)*, Cray, 2006.

[4] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," *Computer*, vol. 41, no. 2, pp. 69–76, 2008.

[5] C. Torres-Huitzil and M. Arias-Estrada, "FPGA-based configurable systolic architecture for window-based image processing," *Eurasip Journal on Applied Signal Processing*, vol. 2005, no. 7, pp. 1024–1034, 2005.

[6] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "FPGA-Based face detection system haar classifiers," in *Proceedings of the 7th ACM SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '09)*, pp. 103–111, Monterey, Calif, USA, February 2009.

[7] J. Le Moigne, W. J. Campbell, and R. F. Cromp, "An automated parallel image registration technique based on the correlation of wavelet features," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 8, pp. 1849–1864, 2002.

[8] O. Dandekar and R. Shekhar, "FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 2, pp. 116–127, 2007.

[9] B. Zitová and J. Flusser, "Image registration methods: a survey," *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[10] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[11] M. Huang, O. Serres, T. El-Ghazawi, and G. Newby, "Implementing image registration algorithms on reconfigurable computer," in *Proceedings of the 10th Military and Aerospace Programmable Logic Devices Conference (MAPLD '08)*, September 2008.

[12] M. Huang, O. Serres, S. Lopez-Buedo, T. El-Ghazawi, and G. Newby, "An image processing architecture to exploit I/O

bandwidth on reconfigurable computers," in *Proceedings of the 4th Southern Conference on Programmable Logic (SPL '08)*, pp. 257–260, San Carlos de Bariloche, Argentina, March 2008.

[13] M. S. Puranik and D. C. Gharpure, "FPGA implementation of MFNN for image registration," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (ICFPT '02)*, pp. 364–367, December 2002.

[14] S. Liu, N. Li, and W. Wang, "Efficient codesign for geology image processing," in *Proceedings of the 49th Midwest Symposium on Circuits and Systems (MWSCAS '06)*, vol. 1, pp. 272–275, San Juan, Puerto Rico, USA, August 2006.

[15] E. El-Araby, M. Taher, T. El-Ghazawi, and J. L. Moigne, "Automatic image registration for remote sensing on reconfigurable computers," in *Proceedings of the 9th Military and Aerospace Programmable Logic Devices Conference (MAPLD '06)*, 2006.

[16] M. A. de Barros and M. Akil, "Low level image processing operators on FPGA: implementation examples and performance evaluation," in *Proceedings of the 12th IAPR Intarnational Conference on Pattern Recognition*, vol. 3, pp. 262–267, October 1994.

[17] V. Muthukumar and D. V. Rao, "Image processing algorithms on reconfigurable architecture using HandelC," in *Proceedings of the EUROMICRO Systems on Digital System Design (DSD '04)*, pp. 218–226, Rennes, France, August-September 2004.

[18] K. T. Gribbon, D. G. Bailey, and C. T. Johnston, "Using design patterns to overcome image processing constraints on FPGAs," in *Proceedings of the 3rd IEEE International Workshop on Electronic Design, Test and Applications (DELTA '06)*, pp. 47–53, Kuala Lumpur, Malaysia, January 2006.

[19] L. G. Brown, "Survey of image registration techniques," *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, 1992.

[20] T. A. El-Ghazawi, P. Chalermwat, and J. L. Moigne, "Wavelet-based image registration on parallel computers," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC '97)*, pp. 20–28, November 1997.

[21] http://www.tilera.com/.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 475620, 10 pages
doi:10.1155/2010/475620

Research Article

Multiloop Parallelisation Using Unrolling and Fission

Yuet Ming Lam,[1] José Gabriel F. Coutinho,[2] Chun Hok Ho,[2] Philip Heng Wai Leong,[3] and Wayne Luk[2]

[1] *Faculty of Information Technology, Macau University of Science and Technology, Taipa, Macau, China*
[2] *Department of Computing, Imperial College London, London, UK*
[3] *School of Electrical and Information Engineering, University of Sydney, Sydney, NSW, Australia*

Correspondence should be addressed to Yuet Ming Lam, ymlam@must.edu.mo

Received 1 July 2009; Accepted 19 October 2009

Academic Editor: Valentin Obac Roda

A technique for parallelising multiple loops in a heterogeneous computing system is presented. Loops are first unrolled and then broken up into multiple tasks which are mapped to reconfigurable hardware. A performance-driven optimisation is applied to find the best unrolling factor for each loop under hardware size constraints. The approach is demonstrated using three applications: speech recognition, image processing, and the N-Body problem. Experimental results show that a maximum speedup of 34 is achieved on a 274 MHz FPGA for the N-Body over a 2.6 GHz microprocessor, which is 4.1 times higher than that of an approach without unrolling.

1. Introduction

Microprocessors are commonly used to implement computing systems as they have the advantages of low cost and fast development time. In performance-critical applications, performance can be improved by introducing larger degrees of spatial parallelism via reconfigurable hardware implemented on field programmable gate arrays (FPGAs). Heterogeneous computing systems using both microprocessors and FPGA-based custom function units can combine advantages of both for many applications.

Computational intensive tasks in digital signal processing algorithms are usually iterative operations. Scheduling such loops in a heterogeneous computing system to fully utilise the available resources is difficult due to their complex nature. Techniques which have been previously proposed tend to address single loop only and are summarised as follows.

(i) Control flow based [1, 2]. This approach divides a control flow graph into various subgraphs based on control edges, and each subgraph is scheduled independently, typically list scheduling technique is used. A complete scheduling is generated by combining all the schedulings of subgraphs. This approach only analyses one iteration of the loop body, it does not target generating higher parallelism implementation for multiprocessor systems.

(ii) Modulo scheduling [3]. Generates a schedule for one iteration of a loop so that all iterations repeat at a fixed interval, that is, a software pipelined design. Since only a single iteration is analysed, limited parallelism is achieved.

(iii) Graph conversion [4]. An application with loop can be characterised as a cyclic graph, this approach attempts to find a better scheduling of the loop body by using a graph traversal algorithm to convert the cyclic graph to an acyclic one with minimised critical path. Depth-first search technique is used to traverse the cyclic graph and remove the feedback edge; an acyclic graph scheduling technique is then used to form a scheduling of the loop body. This approach does not analyze task dependency in different iterations which may result in reduced parallelism.

Table 1: Some approaches to address mapping/scheduling.

References	Approach	Examples of applications	Comments
[1, 2]	Control flow based	GCD, counter, Filtering	Multiprocessors system not addressed
[3]	Modulo scheduling	DCT, FFT	Analyze one iteration, single loop
[4]	Graph conversion	Random graphs	Less parallelism, single loop
[5–7]	Loop unrolling	Random graphs, FFT, solver equalizer	Single loop unrolling
[8]	Dynamic scheduling	Fractal generation	Loop unrolling not addressed, single loop
[9, 10]	Loop fission	JPEG compression, DCT, BPIC	Loop unrolling not addressed, single loop
This work	Multiloop unrolling	Speech system image processing, N-Body	Global unrolling factors determining, coarse-grained, heterogeneous systems

(iv) Loop unrolling [5–7]. This is a common technique to generate an implementation with greater parallelism. It involves unrolling a loop and extracting parallel tasks from different loop iterations. These references have only been applied to parallelise a single loop.

(v) Dynamic scheduling [8]. This approach schedules tasks at run-time making use of both online and offline parameters. The loop condition is checked dynamically at runtime. Loop parallelisation is not addressed in this approach.

(vi) Loop fission [9, 10]. This approach breaks a loop into multiple tasks and maps each individual one to FPGA. Implementing applications which exceed the size constraint on FPGA thus becomes feasible. Since loop unrolling is not involved, this approach results in limited parallelism.

A comparison between this work and different approaches is shown in Table 1. Previous work has focused on parallelising a single loop [3–7], and multiloop optimisation has not been adequately addressed. Since reconfigurable hardware in a heterogeneous system is capable of supporting parallel execution of tasks, a major challenge is to develop techniques which can effectively exploit this capability.

This work explores techniques to optimise applications with multiple loops in a heterogeneous computing system. Our recent work has shown that an integrated mapping and scheduling scheme with multiple neighborhood functions [11], and combining mapping and scheduling with loop unrolling [12] can achieve considerable performance gains. This work complements those results through a method for optimising the unrolling factors in multiple loops. The novel aspects of this work are as follows:

(i) a performance-driven strategy, combined with an integrated mapping/scheduling system with multiple neighborhood functions, to find the best unrolling factor for each loop (Section 2.4),

(ii) a static mapping and scheduling technique capable of handling cyclic task graphs for which the number of iterations is not known until run-time (Sections 3.1 and 3.3),

(iii) The introduction of additional management tasks for dynamic data synchronisation while maintaining near optimal performance when an accurate compile-time prediction of the run-time condition is made (Section 3.2).

The remainder of this paper is organised as follows. The proposed multiloop parallelisation scheme is presented in Section 2. Section 3 introduces the loop unrolling technique and provides an overview of the multiple neighborhood function based mapping/scheduling system. Experimental results are given in Section 4, and finally, concluding remarks are given in Section 5.

2. Multi-Loop Parallelisation

2.1. Reference Architecture. The reference heterogeneous computing system contains two processing elements (PEs): one microprocessor and one FPGA. Each processing element has a local memory for data storage during task execution, and the communication channel between these two processing elements is being assigned a weight which specifies the data transfer rate. Results of a task's predecessors must be transferred to the local memory before this task starts execution.

2.2. Notations. Given an application containing a loop (Figure 1(a)), the followings are various notations used in this paper:

(i) Loop Unrolling and Unrolling Factor. Loop unrolling is a process to duplicate the body of a loop multiple times and use them to replace the original body, where the loop-control code is adjusted accordingly. The number of copies being duplicated is called unrolling factor. For example, Figure 1(b) shows an unrolled loop with an unrolling factor of N.

(ii) Loop Fission and Sub-Loop. Loop fission is a process to split a loop that contains multiple instructions into a number of loops with the same loop control. Each splitted loop is called a sub-loop which contains a portion of instructions of the original loop body. For instance, Figure 1(c) shows multiple sub-loops after fission.

(iii) Task. A task is a block of consecutive instructions derived from task partitioning stage for a given application [13], for example, a loop in Figure 1(a) is a task.

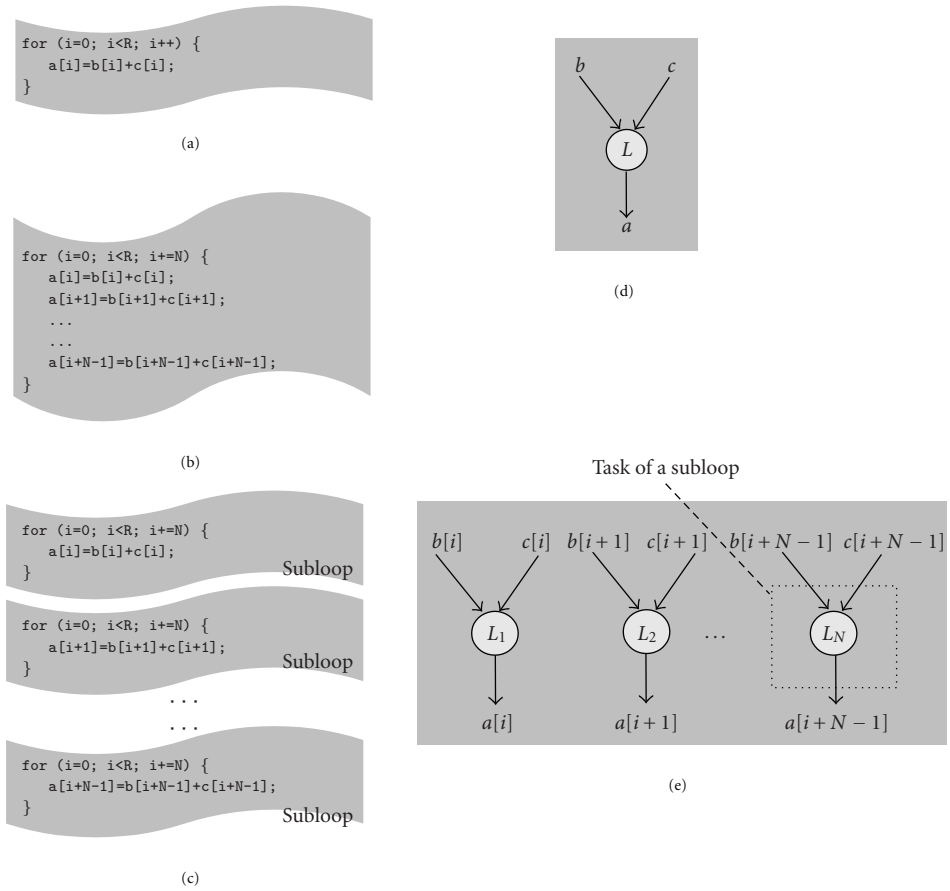


Figure 1: Examples showing various notations: (a) Loop. (b) Unrolled loop. (c) Obtained loop after fission. (d) Task graph representing the original loop, where the loop is a node L in the graph. (e) Task graph representing the loops after fission.

(iv) Task Graph. A task graph is an acyclic graph representing the data flow dependencies of tasks, where a task in the task graph is only executed once and it cannot be executed prior to its predecessors due to the data dependency. For instance, Figures 1(d) and 1(e) are two task graphs of Figures 1(a) and 1(c), respectively, where each loop is a node in the graph.

2.3. Overview. Figure 2 gives an overview of the proposed multi-loop parallelisation strategy. A search strategy is employed where the goal is to find an optimal unrolling factor for each loop so that the overall performance is maximised. This section focuses on the search of unrolling factors; the calculation of quality score will be introduced in Section 3.

Given an application containing a set of loops $LP = \{lp_1, lp_2, \ldots, lp_n\}$, let $UC = \{uc_1, uc_2, \ldots, uc_m\}$ be a set of unrolling configurations with each $uc_i = \{uf_1, uf_2, \ldots, uf_n\}$ designating an instance of the unrolling factors of all loops, where uf_j is the unrolling factor of loop j. Each unrolling configuration uc_i thus contains all unrolling factors for all loops in this application. In each iteration of the search, a set of unrolling configurations UC is firstly generated, and a quality score is then calculated for each configuration after loop unrolling and fission, task graph generation, and mapping/scheduling processes have been applied. The best unrolling configuration uc_i with highest quality score is selected and used for the next iteration. This process is repeated iteratively until a termination condition is reached, the goal being to find a solution with the maximum quality score.

The advantage of considering unrolling and fission of all loops globally is that unrolled sub-loops from various loops can be potentially executed in parallel. This allows for a better mapping/scheduling solution to be found after unrolling and fission. Figure 3 shows an example of unrolling two loops which have no data dependencies between iterations. In the original graph, B and Q represent two loops, $B1$, $B2$, and $B3$ are the three unrolled sub-loops of B; Q is unrolled as $Q1$, $Q2$, and $Q3$. Before unrolling and fission, B and Q are mapped to two processing elements PE1 and PE2. Hardware resources are not fully utilised and the processing time for three iterations using this mapping is 90 time units (Figure 3(c)). After unrolling and fission, the first two sub-loops ($Q1$ and $Q2$) are mapped to PE2 and PE3, respectively, and other unrolled sub-loops are mapped to PE1. Processing time is reduced to 50 time units (Figure 3(d)). Tasks SB and SQ are two generated management tasks to synchronise results produced by different sub-loops which will be introduced in Section 3.2.

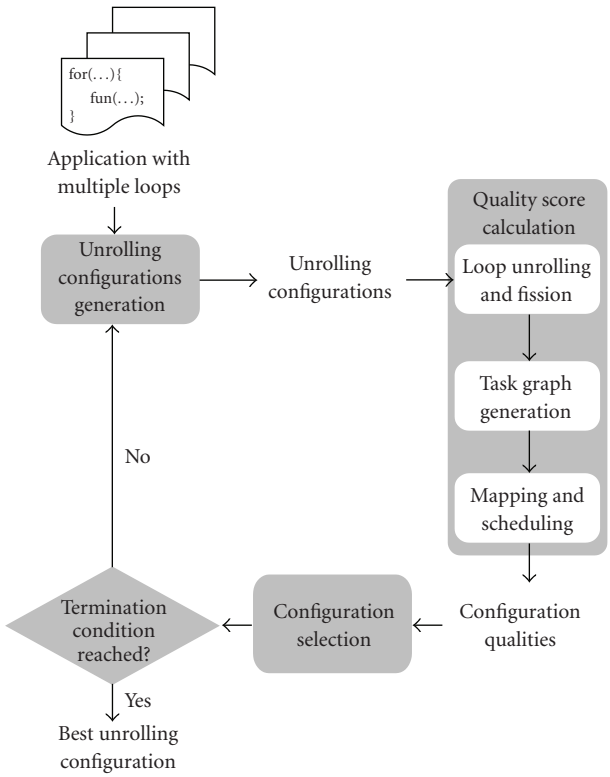


Figure 2: Overview of the proposed multi-loop parallelisation strategy.

Unrolling and fission can still achieve higher parallelism for loops with data dependency between iterations. As a loop may be executed in parallel with other tasks in an application, after unrolling and fission, execution sequence of unrolled sub-loops can be better combined with other tasks. Figure 4 shows the unrolling/fission of two loops with data dependencies between iterations. Before unrolling and fission, B is mapped to PE1 and Q is mapped to PE2. The overall processing time for three iterations is 90 time units (Figure 4(c)). After unrolling, the first sub-loop of Q (i.e., $Q1$) is mapped to PE2, and the remaining sub-loops ($Q2$ and $Q3$) can be executed in PE1. The overall processing time becomes 70 time units (Figure 4(d)). A better mapping/scheduling solution with higher inter-loop parallelism is thus obtained.

2.4. Generation and Selection of Unrolling Configuration. If an application contains only one loop it obviously should be selected for unrolling. For the multiple loop case, the number of loops to unroll and the corresponding unrolling factors need to be determined. Since unrolling a loop without data dependencies between iterations is likely to achieve more performance gain than unrolling a loop with data dependencies, a performance-driven strategy (Algorithm 1) is proposed in this work.

Given an application containing a set of loops $LP = \{lp_1, lp_2, \ldots, lp_n\}$, an initial unrolling configuration uc_{best} is generated with all unrolling factors uf_i being set to 1, that is, $uc = \{uf_i\}$, where $uf_i = 1$ for $1 \le i \le n$. A new set of unrolling configurations $new\_uc = \{uc_1, uc_2, \ldots, uc_n\}$ is generated by incrementing each uf_i in turn, for example, $uc_1 = \{2, 1, \ldots, 1\}$ and $uc_2 = \{1, 2, \ldots, 1\}$. For each unrolling configuration uc_i, a quality score qs_i is calculated by first applying the unrolling factors specified in uc_i followed by fission to break the new loop into sub-loops over the same loop count with each loop having the same loop body as the original loop. A task graph is then generated with each sub-loop being treated as a task. The task graph is then passed to the mapping and scheduling process, where a complete mapping/scheduling solution is generated and a quality score is calculated (Section 3). As a result, a set of quality scores $QS = \{qs_1, qs_2, \ldots, qs_n\}$ is produced. Afterward the corresponding quality improvement qi_i is calculated as:

$$qi_i = qs_i - qs_{\text{best}}, \tag{1}$$

where qs_{best} is the best quality score to date. The best unrolling configuration uc_i with highest quality improvement qi_i is chosen, the best quality score qs_{best} is updated as qs_i, and the unrolling configuration uc_{best} is replaced by uc_i. This process is repeated until the resources on the FPGA are exhausted, causing termination of the algorithm.

3. Quality Score Calculation

3.1. Unrolling, Fission, and Task Graph Generation. Given a set of loops $LP = \{lp_1, lp_2, \ldots, lp_n\}$ and an unrolling configuration $uc = \{uf_1, uf_2, \ldots, uf_n\}$. The following steps are used to generate a task graph:

(i) Unroll each loop lp_i according to uf_i.

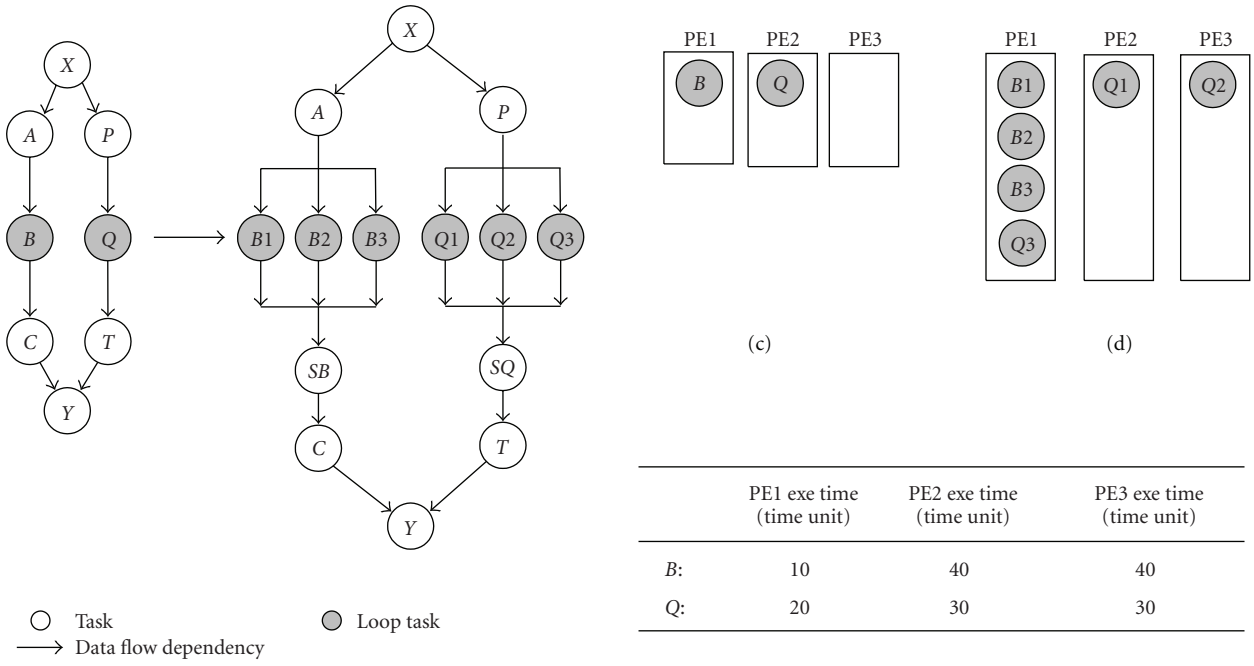
(ii) Break each unrolled loop lp_i into uf_i subloops by fission, each subloop performs the same operations as the original loop body before unrolling.

(iii) Construct a new task graph by treating each subloop as a task, each having the same parent and child tasks as the original task before unrolling.

(iv) Generate a management task to synchronise results produced by different sub-loops (Section 3.2), and insert this task to the tails of all unrolled sub-loops in the task graph (tasks SB and SQ in Figure 3(b)), that is, predecessors of the management task are the unrolled sub-loops, and successors of the management task are the successors of the original loop.

The produced task graph is then presented to the mapping and scheduling tool to generate a quality score (Section 3.3), which guides the search.

3.2. Management Task. One of the problems introduced after unrolling is data synchronisation: since results are produced by unrolled iterations in parallel, they need to be reorganised in the correct sequence (Figure 5). Another problem is loop count uncertainty, for example, a loop may be unrolled n times but the actual loop count at run-time may not be a multiple of n. In this case some results must be discarded.



(e)

	PE1 exe time (time unit)	PE2 exe time (time unit)	PE3 exe time (time unit)
B:	10	40	40
Q:	20	30	30

Figure 3: Mapping/scheduling example for loop unrolling and fission without data dependencies between iterations: (a) Task graph containing two parallel loops. (b) Task graph after unrolling the two loops for three iterations. (c) Mapping and scheduling solution before unrolling and fission, overall processing time for 3 iterations is 90 time units. (d) Mapping and scheduling solution after unrolling and fission, overall processing time is 50 time units. (e) Execution time of one iteration of the loop body for different processing elements. Higher interloop and intraloop parallelism are achieved by unrolling two loops.

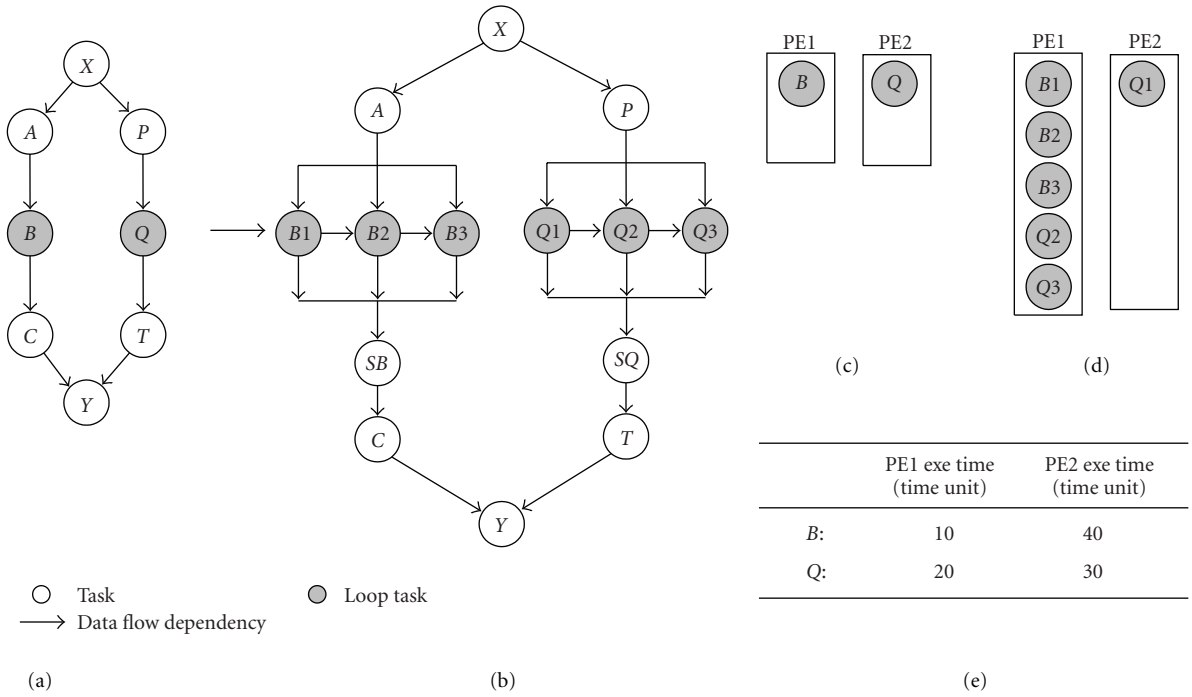
To handle these problems, a management task which collects data from different unrolled tasks, keeps track of the actual loop count at run-time, organises the collected data into the correct sequence, and discards unneeded data is introduced. The management task is treated as a normal task, inserted into the task graph and presented to the mapping/scheduling tool. For loops without data dependencies, the following pseudo-code shows the data synchronisation process:

```
for (i = 0; i < (M-1); i++)  {
    for  (j = 0; j < N; j++)  {
      rst[i∗N+j] = d[j][i];
    }
}
tc = R − (M−1) ∗ N;
for (i = 0; i < tc; i++) {
    rst[(M−1)∗N+i] = d[i][M−1];
}
```

where `M` is the actual count of the unrolled loop being executed, `R` is the required loop count for the loop before unrolling, and `N` is the number of iterations being unrolled. *d* is the result produced by different unrolled iterations, for example, *d*[0] is the result produced by the first iteration. rst is the original array to store results. The second loop is used to collect the results of the last iteration and discard unneeded data, where *tc* is the number of data remaining.

If there are data dependencies between iterations, the management task must select the correct result from the unrolled iterations:

```
tc = M ∗ N − R;
switch(tc) {
    case 0:
      rst = d[N-1];
      break;
    case 1:
      rst = d[N-2];
      break;
    ...
    ...
    case N-1:
      rst = d[0];
      break;
}
```



	PE1 exe time (time unit)	PE2 exe time (time unit)
B:	10	40
Q:	20	30

Figure 4: Mapping/scheduling example for loop unrolling and fission with data dependencies between iterations: (a) Task graph containing two parallel loops having data dependency between iterations. (b) Task graph after unrolling the two loops for three iterations. (c) Mapping and scheduling solution before unrolling and fission, overall processing time for 3 iterations is 90 time units. (d) Mapping and scheduling solution after unrolling and fission, overall processing time is 70 time units. (e) Execution time of one iteration of the loop body for different processing elements. Higher inter-loop parallelism is achieved by unrolling two loops.

(1) $used\_fpga\_area \Leftarrow 0$
(2) $uc_{\text{best}} = \{uf_i\}$, where $uf_i = 1$ for $1 \le i \le n$
(3) $qs_{\text{best}} \Leftarrow 0$
(4) **while** $used\_fpga\_area < total\_fpga\_area$ **do**
(5) **for all** loops lp_i **do**
(6) $uc_i[uf_i] = uc[uf_i] + 1$
(7) **end for**
(8) **for all** unrolling configurations uc_i **do**
(9) **for all** loops lp_i **do**
(10) unroll lp_i for uf_i iterations, where $uf_i \in uc_i$
(11) loop fission
(12) **end for**
(13) generate new task graph
(14) generate complete mapping/scheduling ms_i
(15) calculate quality score qs_i for ms_i
(16) $qi_i \Leftarrow qs_i - qs_{\text{best}}$
(17) **end for**
(18) find loop i with maximum qi
(19) $qi_{\text{best}} \Leftarrow qi_i$
(20) $qs_{\text{best}} \Leftarrow qs_i$
(21) $ms_{\text{best}} \Leftarrow ms_i$
(22) $uc_{\text{best}} \Leftarrow uc_i$
(23) update $used\_fpga\_area$
(24) **end while**
(25) **return** uc_{best} and ms_{best}

Algorithm 1: Search the best unrolling configuration.

The generated mapping/scheduling solution does not require the designer to know the exact loop termination conditions using these management tasks. However, users can specify an estimated loop count at compile time. Loops are unrolled using this information and a mapping/scheduling solution is generated. If the estimated loop count matches the actual value at run-time, maximum performance can be achieved. However, if the loop count is different, the data management task can handle data synchronisation dynamically, which means that the generated mapping/scheduling solution is still feasible. These management tasks can easily be implemented in software or in hardware state machines.

3.3. Mapping and Scheduling Overview. A heuristic search-based approach is used to find the best mapping/scheduling solution for an input task graph as shown in Figure 6. Given a task graph and a target architecture specification which includes information concerning the processing elements and communications channel, a tabu search is used to iteratively generate different mapping/scheduling solutions (neighbors). For each solution, a speedup coefficient is calculated and used to guide the search with the goal being to find a solution with maximum speedup.

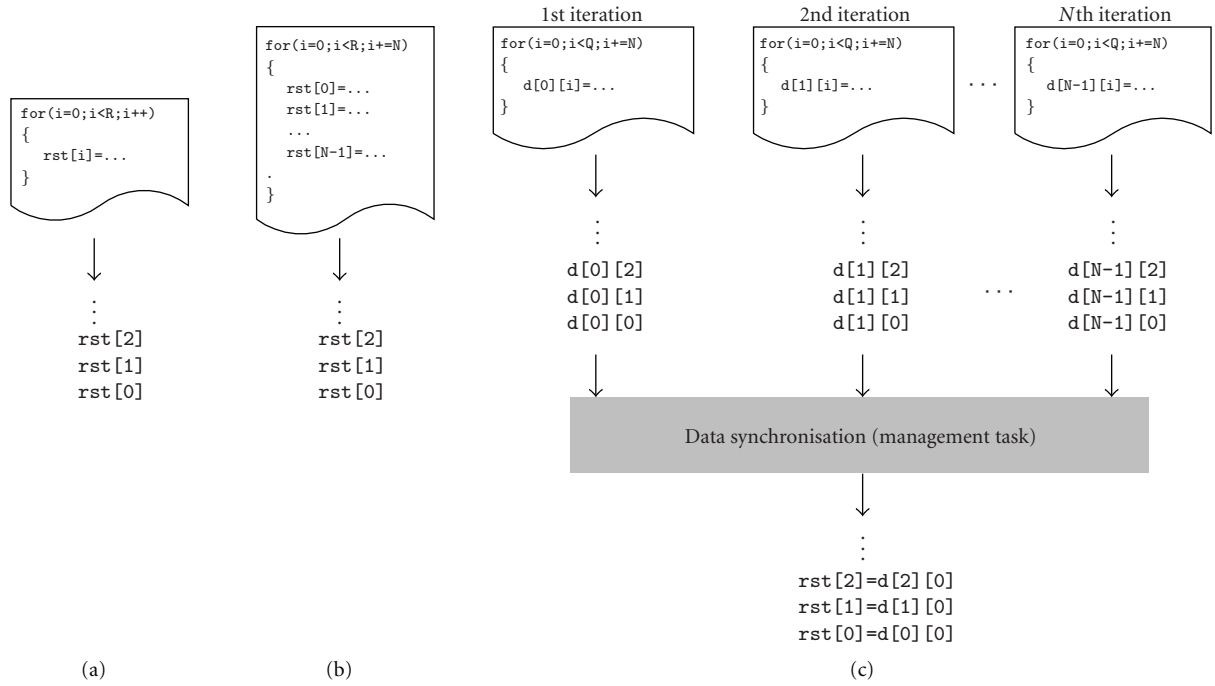


Figure 5: Data synchronisation after unrolling and fission. (a) Original loop. (b) Unrolled loop. (c) Generated sub-loops after fission.

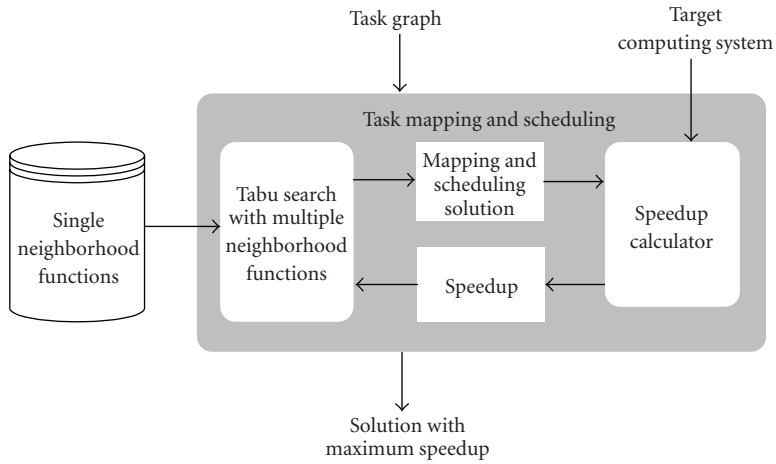


Figure 6: Overview of the mapping and scheduling system.

3.4. Integrated Scheduling Technique. Given a set of tasks $TK = \{tk_1, tk_2, \ldots, tk_n\}$ and a set of task lists $PL = \{pl_1, pl_2, \ldots, pl_m\}$, where each task list $pl_j = (as_{j1}, as_{j2}, \ldots, as_{jq})$ is an ordered task sequence to be executed by processing element pe_j, each task in pl_j will be processed by pe_j in sequence when it is ready for execution, that is, when all of its predecessors are finished. Task mapping and scheduling is thus integrated in a single step that deals with assigning tasks to task lists. A task assignment function is defined as A: $TK \rightarrow PL$, for example, $A(tk_i) = as_{rq}$ denotes task tk_i being assigned to as_{rq} of list pl_r. This means that tk_i is the qth task to be executed by processing element pe_r. A mapping/scheduling solution is characterized by assignments of all tasks to processing elements, that is, for every task $tk_i \in TK$, $A(tk_i) = as_{rq}$ for a $pl_r \in PL$.

3.5. Multiple Neighborhood Functions. Tabu search is used to find the best mapping/scheduling solution. It is based on neighborhood search, which starts with a feasible solution and attempts to improve it by searching its neighbors, that is, solutions that can be reached directly from the current solution by an operation called a move. Tabu search keeps a list of the searched space and uses it to guide the future search direction; it can forbid the search moving to some neighbors. In the proposed tabu search technique with multiple neighborhood functions, after an initial solution is generated, two neighborhood functions are designed to move tasks between task lists and used to generate various neighbors simultaneously [11]. If there exists a neighbor better than the best solution so far and it cannot be found in the tabu list, this neighbor is recorded. Otherwise, a neighbor

Table 2: Profiling results for major processes of the isolated word recognition system.

Process	% of exe time
vq	71.19
autocc	15.4
hmmdec	6.11
windowing	4.39
lpc_analysis	0.95
lpc2cep	0.93
find_max	0.39
others	0.64

that cannot be found in the tabu list is recorded. If all the above conditions cannot be fulfilled, a solution in the tabu list with the least degree, that is, a solution being resident in the tabu list for the longest time, is recorded. If the recorded solution has a smaller cost than the best solution so far, it is recorded as the best solution. The searched neighbors are added to tabu list and solutions with the least degrees are removed. This process is repeated until the search cannot find a better solution for a given number of iterations.

3.6. Quality Score. For each mapping/scheduling solution, an overall execution time is calculated, which is the time to process all tasks using the reference heterogeneous computing system and includes data transfer time. The processing time of a task tk_i on processing element pe_k is calculated as the execution time of tk_i on pe_k plus the time to retrieve results from all of its predecessors. The data transfer time between a task and a predecessor is assumed to be zero if they are assigned in the same processing element.

A speedup coefficient is defined and used to measure the quality of a mapping/scheduling solution, it is calculated as the processing time using a single microprocessor divided by the processing time using the heterogeneous computing system:

$$\text{speedup} = \frac{\text{processing time}_{\text{single CPU}}}{\text{processing time}_{\text{Reference system}}}. \quad (2)$$

A higher speedup means that a mapping/scheduling solution is better as the application can be finished using less time. This score is used to guide the tabu search and the goal is finding a solution with maximum speedup. This maximum speedup is used as the final output and defined as the quality score to measure the quality of the input unrolling configuration.

4. Results

4.1. Experimental Setup. The reference heterogeneous computing system used in work has one 2.6 GHz AMD Opteron(tm) Processor 2218 and one Celoxica RCHTX-XV4 FPGA board with a Xilinx Virtex-4 XC4VLX160 FPGA. The FPGA board and microprocessor are connected via an HTX interface with maximum data transfer rate of 3.2 GB/s.

Table 3: FPGA resources of different speech process, the total area is calculated by counting two "hmmdec12".

Process	Area (slice)
vq3	21819
autocc12	10272
hmmdec12	15948
Total	63987

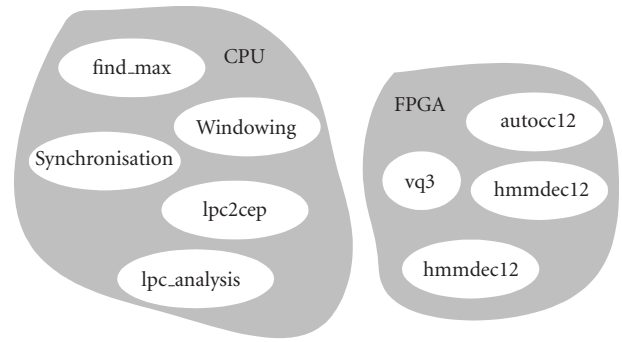


Figure 7: Mapping the speech recognition system to the heterogeneous computing system. "vq3" are the unrolled 3 iterations of vector quantisation. "autocc12" are the unrolled 12 iterations for autocorrelation. "hmmdec12" are the unrolled 12 iterations for HMM decoding. Two "hmmdec12" mean the outer loop of HMM decoding is unrolled for 2 iterations, that is, decoding of two words is executed in parallel.

An isolated word recognition (IWR) system [14] is used as an application. It uses 12th order linear predictive coding coefficients (LPCCs), a codebook with 64 code vectors, and 20 hidden Markov models (HMMs), each with 12 states. One set of utterances from the TIMIT TI 46-word database [15] containing 5082 words from 8 males and 8 females is used for recognition. Table 2 shows the profiling results of major processes of the isolated word recognition system on the AMD processor. It is found that loops in vector quantisation (vq), autocorrelation (autocc), and hidden Markov model decoding (hmmdec) consumed the most CPU resource, which are 71.19%, 15.4%, and 6.11%, respectively.

4.2. Multi-Loop Unrolling and Fission. In this experiment, the proposed unrolling strategy is applied. Figure 7 shows the mapping of different processes in the speech system. It is found that vector quantisation is unrolled 3 times (vq3) and mapped to the FPGA; all 12 iterations of the autocorrelation process are unrolled (autocc12); inner loop of hidden Markov model decoding is unrolled for 12 iterations (hmmdec12) which is equal to the number of HMM states; the outer loop is further unrolled for 2 iterations which means two HMM decoding are executed in parallel. The corresponding FPGA resource usage is shown in Table 3 and the operating frequency is 318.7 MHz, a speedup (quality score) of 10 is obtained for this configuration. In contrast, the speedup obtained without unrolling is 4.7, where vector

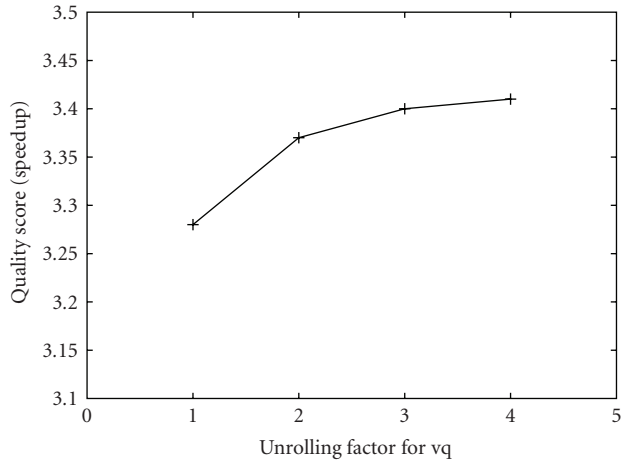


Figure 8: Quality scores (speedups) for different unrolling factors of vector quantisation.

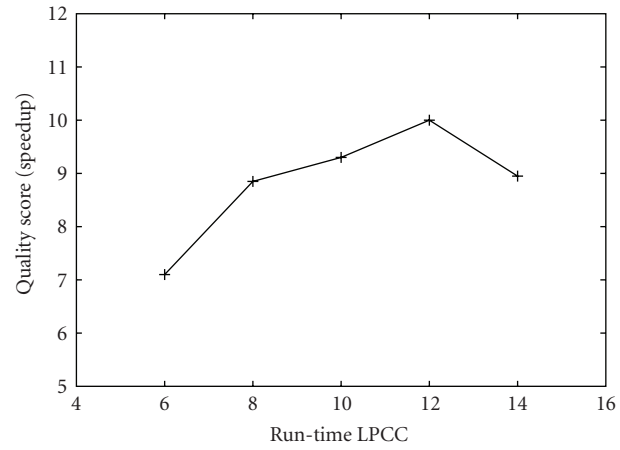


Figure 9: Quality scores (speedups) for different run-time LPCCs, the estimated LPCC order during compile-time is 12.

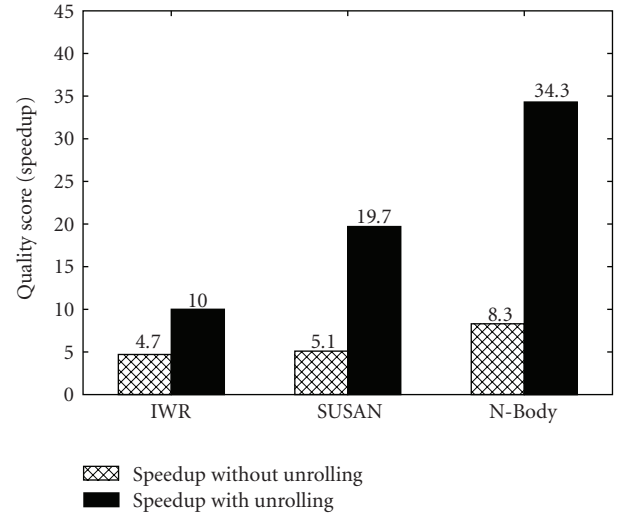


Figure 10: Quality score (speedup) comparison between unrolling and without unrolling for different applications.

Table 4: FPGA resources and operating frequency for IWR, SUSAN, and N-Body.

Application	Area (slice)	Frequency (MHz)
IWR	63987 (94.7%)	318.7
SUSAN	60106 (88.9%)	274
N-Body	62139 (91.9%)	274

quantisation, autocorrelation, and HMM decoding are executed in FPGA without unrolling. An improvement of 2.1 times is hence obtained using the proposed strategy.

Figure 8 shows the speedups for different vector quantisation unrolling factors, where all other processes are executed on the CPU. It is found that the speedup increases with unrolling factor and saturates. This figure explains why only three iterations of vector quantisation are unrolled in the final mapping/scheduling solution.

4.3. Run-Time versus Compile-Time Parameters. In the above experiment, mapping/scheduling solutions are generated by assuming that the LPCC order is 12 at compile-time. However, this value may be modified to cope with different circumstances at run-time. Using a mapping/scheduling solution generated with 12 LPCCs, Figure 9 shows the performance of this system for different run-time LPCC orders. It is found that maximum performance is achieved at 12 LPCCs, and the performance drops when the run-time LPCC order is different from compile-time value, for example, a 5% drop at 10 LPCCs.

4.4. Quality Score Comparison. In addition to the IWR example, two other applications are employed to evaluate the proposed approach: the SUSAN corner detection image processing algorithm [16] and the N-Body problem [17]. Figure 10 shows the quality score comparison between strategies with and without unrolling. The FPGA resource usage and operating frequency are shown in Table 4. The proposed strategy can achieve 10, 19.7, and 34.3 times speedup for IWR, SUSAN, and N-Body, respectively, the corresponding improvements are factors of 2.1, 3.9, and 4.1 over the approach without unrolling. The improvements for SUSAN and N-Body are much higher than the 2.1 times improvement obtained using the IWR application because there is a critical loop in each of these two applications: in SUSAN, the loop to compute the similarity of pixels and for N-body, the loop to compute velocity. Unrolling these loops significantly improves the performance of those cases.

5. Conclusions

A multi-loop parallelisation technique involving fission and unrolling is proposed to improve intra-loop and inter-loop parallelism in heterogeneous computing systems. The utility of this approach is demonstrated in three practical applications and a maximum speedup of 34.3 times is obtained using a computing system containing an FPGA and a microprocessor. It is 4.1 times higher than the case where

unrolling is not applied. The generated system is tolerant to run-time conditions, and its performance is closer to optimum when there is a more accurate prediction of run-time condition during compile-time.

Acknowledgment

The support from FP6 hArtes (Holistic Approach to Reconfigurable Real Time Embedded Systems) Project, the UK Engineering and Physical Sciences Research Council, Celoxica, and Xilinx is gratefully acknowledged.

References

[1] R. Camposano, "Path-based scheduling for synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 85–93, 1991.

[2] M. Rahmouni and A. A. Jerraya, "Formulation and evaluation of scheduling techniques for control flow graphs," in *Proceedings of the European Design Automation Conference (EURO-DAC '95)*, pp. 386–391, Brighton, UK, September 1995.

[3] A. Hatanaka and N. Bagherzadeh, "A modulo scheduling algorithm for a coarse-grain reconfigurable array template," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–8, 2007.

[4] F. E. Sandnes and O. Sinnen, "A new strategy for multiprocessor scheduling of cyclic task graphs," *International Journal of High Performance Computing and Networking*, vol. 3, no. 1, pp. 62–71, 2005.

[5] T. Yang and C. Fu, "Heuristic algorithms for scheduling iterative task computations on distributed memory machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 6, pp. 608–622, 1997.

[6] M. Weinhardt and W. Luk, "Pipeline vectorization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 234–248, 2001.

[7] P. Šůcha, Z. Hanzálek, A. Heřmánek, and J. Schier, "Efficient FPGA implementation of equalizer for finite interval constant modulus algorithm," in *Proceedings of the International Symposium on Industrial Embedded Systems (IES '06)*, pp. 1–10, Antibes Juan-les-Pins, France, October 2006.

[8] H. Styles, D. B. Thomas, and W. Luk, "Pipelining designs with loop-carried dependencies," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '04)*, pp. 255–262, 2004.

[9] M. Kaul, R. Vemuri, S. Govindarajan, and I. Ouaiss, "An automated temporal partitioning and loop fission approach for FPGA based reconfigurable synthesis of DSP applications," in *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, pp. 616–622, 1999.

[10] J. M. P. Cardoso, "Loop dissevering: a technique for temporally partitioning loops in dynamically reconfigurable computing platforms," in *Proceedings of the International Parallel and Distributed Processing Symposium*, pp. 22–26, April 2003.

[11] Y. M. Lam, J. G. F. Coutinho, W. Luk, and P. H. W. Leong, "Mapping and scheduling with task clustering for heterogeneous computing systems," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 275–280, Berlin, Germany, September 2008.

[12] Y. M. Lam, J. G. F. Coutinho, W. Luk, and P. H. W. Leong, "Unrolling-based loop mapping and scheduling," in *Proceedings of the International Conference on Field-Programmable Technology (ICFPT '08)*, pp. 321–324, Taipei, Taiwan, December 2008.

[13] W. Luk, J. G. F. Coutinho, T. Todman, et al., "A high-level compilation toolchain for heterogeneous systems," in *Proceedings of the IEEE International SOC Conference*, pp. 9–18, Belfast, Ireland, September 2009.

[14] L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1993.

[15] LDC, http://www.ldc.upenn.edu/.

[16] S. M. Smith and J. M. Brady, "SUSAN—a new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[17] S. J. Aarseth, "Direct methods for N-body simulation," in *Multiple Time Scales*, Academic Press, New York, NY, USA, 2001.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 787405, 9 pages
doi:10.1155/2010/787405

Research Article

Power Characterisation for Fine-Grain Reconfigurable Fabrics

Tobias Becker,[1] Peter Jamieson,[1] Wayne Luk,[1] Peter Y. K. Cheung,[2] and Tero Rissa[3]

[1] *Department of Computing, Imperial College London, London SW7 2AZ, UK*
[2] *Department of EEE, Imperial College London, London SW7 2AZ, UK*
[3] *Nokia Devices R&D, Tampere, Finland*

Correspondence should be addressed to Tobias Becker, tbecker@doc.ic.ac.uk

Received 1 July 2009; Accepted 22 October 2009

Academic Editor: Elías Todorovich

This paper proposes a benchmarking methodology for characterising the power consumption of the fine-grain fabric in reconfigurable architectures. This methodology is part of the GroundHog 2009 power benchmarking suite. It covers active and inactive power as well as advanced low-power modes. A method based on random number generators is adopted for comparing activity modes. We illustrate our approach using five field-programmable gate arrays (FPGAs) that span a range of process technologies: Xilinx Virtex-II Pro, Spartan-3E, Spartan-3AN, Virtex-5, and Silicon Blue iCE65. We find that, despite improvements through process technology and low-power modes, current devices need further improvements to be sufficiently power efficient for mobile applications. The Silicon Blue device demonstrates that performance can be traded off to achieve lower leakage.

1. Introduction

Rapidly evolving standards, convergence of increasingly complex features, and growing time to market pressure are pushing manufacturers of mobile consumer devices to consider alternatives to ASICs and microprocessors. There is a clear demand for power efficient circuits that are flexible, while capable of delivering performance through parallelism. Reconfigurable architectures, such as FPGAs, have good potential in meeting the demand for flexibility and performance, but they often miss the power requirements by up to several orders of magnitude. They can consume roughly ten times more active power than ASICs [1]. Moreover, they can draw inactive power that is one hundred to a thousand times higher than what is permitted for mobile devices in standby mode [2]. Devices can also cause thermal problems when they heat up during intense processing.

One of the problems with optimising for low power in FPGAs is that a designer has to map a particular application onto a range of target devices and evaluate their power consumption by using the power estimation tools provided with some FPGA CAD flows. But these estimation tools have often limited accuracy. Moreover, synthesising, implementing, and simulating a design on a range of devices can be very time consuming. The result of such an evaluation is only meaningful for this particular design. Instead, it would be desirable to have a set of test cases that can be used to benchmark the power characteristics of reconfigurable devices. A list of test results would allow users to choose suitable devices for a low-power design and the procedure can also be used to evaluate architectural improvements.

In order to address these issues we have developed the GroundHog 2009 power benchmarking suite for reconfigurable architectures [3]. GroundHog 2009 includes one application independent benchmarking technique, which is presented in this paper. In addition, GroundHog 2009 also provides six application specific test cases which are representative for mobile devices. These benchmarks are described in [4]. In this paper, we focus on the application independent method that allows a fast and easy evaluation and classification of the power consumption in fine-grain FPGAs. The classification is based on several activity modes which reflect realistic high-power or low-power scenarios a device could be operated in. Another important aspect is thermal characteristics such as the temperature dependency of static power and heating up of the device under different processing scenarios. The proposed methodology is intended to provide a fair comparison of existing devices, and should also be able to capture improvements in future devices with new low-power features.

The goal of this benchmark is to provide a simple technique for evaluating and comparing the power efficiency of devices. Such an evaluation will always be influenced by external factors such as implementation tools and measuring environment, although our benchmarking technique minimises these influences as much as possible. It is also important to point out that this paper describes an example of the benchmark implementation with regard to some of its parameters such as clock frequency or logic utilisation. But a benchmark user may find that a variation of these parameters might be more appropriate for their purposes. This will come at the expense of having limited comparability with other benchmark users but might be more appropriate for evaluating devices in a particular domain.

The remainder of this paper is organised as follows. Section 2 describes previous work. Section 3 proposes a method for power characterisation of fine-grain reconfigurable fabrics. Section 4 describes how this method can be implemented on commercial FPGAs, and Section 5 shows some results of our power characterisation. Finally, Section 6 concludes the paper.

2. Background

In this section we consider some of the general aspects of power consumption as well as their dependencies and trends. We further review previous research on power consumption in FPGAs. Additionally, we review existing FPGA architectures and techniques that are designed for low power.

2.1. Power Consumption in CMOS Devices. The total power consumption in CMOS devices is a combination of static and dynamic power. Static power is caused by leakage currents inside the device while dynamic power is caused by switching activity.

Static power has two main components: gate leakage and subthreshold leakage, with the latter being the most significant component [5]. Gate leakage is a leakage current from the transistor gate through the gate oxide into the substrate. Subthreshold leakage or source-to-drain leakage is a leakage current from the transistor source to drain when the transistor is turned off and the gate voltage is below the threshold voltage. Lowering the threshold voltage, which increases performance, causes subthreshold leakage to grow exponentially. The subthreshold leakage also increases exponentially with the junction temperature. Static power used to be a minor component of the total power consumption. But lower threshold voltages in modern CMOS devices have led to a growth of this component. Static power can be addressed by using thicker gate oxides for noncritical path transistors, back-biasing the substrate or providing standby modes that turn off circuits during periods of inactivity. Static power also has a strong dependency on the variation of process parameters. Hence, the static power profile of a device can vary from die-to-die, wafer-to-wafer, and lot-to-lot [6].

Dynamic power on the other hand is caused by a combination of charging and discharging load capacitances as well as short-circuit currents when transistors switch. Charging and discharging capacitances are usually the dominant effect [5]. Dynamic power is given by (1). It has linear dependency on the clock frequency f and the capacitance C, and a quadratic dependency on the supply voltage V. In general, the capacitance is defined by all transistors in the device. In an FPGA however, it depends on the number of logic and routing elements used in a particular configuration. The factor α is the activity or toggle rate and depends on the design and its input stimuli. Dynamic power is independent of temperature and also does not depend as strongly on process variation as static power

$$P_{\text{dynamic}} = \alpha \cdot C \cdot V^2 \cdot f. \tag{1}$$

Dynamic power is improved by reducing the transistor capacitances through feature scaling and reducing the voltage through voltage scaling. Additional improvements can be made by reducing the switching activity in the device. For example, the clock frequency can be scaled according to processing requirements or the clock can be stop completely during inactive periods.

2.2. FPGA Power Research. One of the earliest comparative studies for power consumption on FPGAs is done by George et al. [7]. The authors create a low-power FPGA through architecture and low-level circuit design, and compare their FPGA to Xilinx and Altera devices. The comparison is based on three test circuits that are evaluated with Synopsis Powermill. The three circuits consist of a single flip-flop driving 9 routing segments, a 1 K array of 16 bit counters, and a toggle circuit.

Shang et al. [8] measure the dynamic power consumption of a Xilinx Virtex-II FPGA [9] using one Xilinx internal benchmark that represents a large industrial circuit. Using this internal benchmark and input stimuli, they calculate the switching activity of the design. They estimate power based on the calculated switching activity and the effective capacitance of each resource on the FPGA. This is possible since they have access to low-level models of the FPGA, but such models are usually not accessible.

Gayasen et al. propose an FPGA architecture with two supply voltages where the lower voltage is used for all noncritical path components [10]. The efficiency of their architecture is evaluated with MCNC benchmarks which provide a range of simple circuits and state machines.

Tessier et al. show that the power efficiency of an application can be improved by reconfiguring between a more powerful, less efficient core and a less powerful, more efficient core on demand [11]. Becker et al. analyse the power consumption during reconfiguration [12].

Recently, Kuon and Rose [1] assess the gap between FPGAs and ASICs. This work includes an attempt to measure the dynamic and static power consumption gap between the two technologies. They estimate the static and dynamic power consumption of an FPGA using the power estimation tools provided by the FPGA vendor, and use either included testbenches or estimates of net activity.

Table 1: Comparison of devices and low-power features.

Device manufacturer	Xilinx	Lattice	Actel	Silicon Blue
Device Name	Spartan-3A, Spartan-3AN	Mach XO	IGLOO	iCE65
Device size [LUTs]	1.6 k–53 k	256–2.3 k	192–36 k (*)	1.8 k–17 k
Low-power feature	Suspend mode	Sleep mode	Flash freeze	iCEgate
Wake-up time [μs]	100–500	1000	1	Instant
Retain state	Yes	No	Yes	Yes

(*) device does not provide LUT/FF pairs like all other devices, reconfigurable tiles can be used as LUT or FF.

FPGA fabrics have also been characterised in terms of their thermal characteristics and die variation. Lopez-Buedo et al. [13] use ring oscillators programmed onto an FPGA. These oscillators are placed around an existing mapped design to measure the temperature of the fabric when the design is operating. The local temperature is determined by measuring the frequency of the ring oscillator.

FPGA clock networks can be responsible of a significant amount of power consumption. Lamoureux and Wilton examine clock-aware placement techniques, and investigate the tradeoff between the power consumption of the clock network and the impact of the constraints imposed by the design automation tools [14].

2.3. Low-Power FPGAs. Tuan et al. present an experimental low-power FPGA with several power optimisations such as voltage scaling, leakage reduction of configuration memory cells, and power gating of tiles with preservation of state and configuration [2]. An implementation on a 90 nm process technology demonstrates a reduction of 46% in active power and 99% in standby power. These values are not specific to a particular FPGA configuration.

Most current commercial FPGAs have limited low-power capabilities. Traditionally, FPGA power optimisations target the operational power in order to reduce the complexity of the power supply and eliminate extra cooling systems. Xilinx introduced triple-oxide technology, which uses less leaky transistors with thicker gate oxide for configuration memory cells and interconnect pass gates [15]. Altera developed a programmable transistor back-biasing feature that allows selecting high-performance logic with high power consumption only for timing-critical components in the design [16]. But these improvements alone are insufficient to enable the use of FPGAs in mobile, battery-based applications. Such a scenario would require dedicated low-power modes that reduce the total power consumption by several orders of magnitude during periods of inactivity. With most current devices, the only methods of saving power during inactivity are to employ clock gating or to turn off the entire device. The first method has limited potential whereas in the latter case, state and configuration are lost.

Recently, some devices with additional low-power capabilities were introduced. Xilinx Spartan-3A and Spartan-3AN FPGAs support a suspend mode in which auxiliary clock circuitry can be stopped and powered down [17]. Lattice Mach XO devices provide a sleep mode [18] that can reduce the standby power by a factor of 100. However, the application state is lost when using this sleep mode. Another example is Actel IGLOO devices which feature a sleep state that does retain the application state [19]. Silicon Blue provides iCE65 FPGAs [20] that are optimised for low static power. Table 1 shows a comparison of these devices and their low-power features.

3. Fabric Characterisation Method

When developing a general characterisation method one faces a number of challenges. The method should:

(i) be applicable to a wide range of devices,

(ii) be a fair comparison and results free from implementation tool influences or hand optimisations,

(iii) allow to capture different power modes and possible future techniques that are currently not available.

The basic idea of our method is to implement a highly active circuit on the FPGA and measure the power in several activity modes. These activity modes represent active processing, inactivity and dedicated low-power states. This method allows us to characterise power consumption quickly in best-case and worst-case scenarios and it outlines the suitability of a device for low-power design. Furthermore, we consider the thermal aspects of the device such as heating up under active processing.

The benefit of this characterisation is that it allows us to assess the adequacy of a device for a low-power design. Moreover, it can be used to compare and optimise devices for lower power. The key aspects of our method can be summarised as follows.

(i) Use random number generators (RNGs) as test circuit with high activity.

(ii) Use 90% of the logic resources in the device.

(iii) Run the test circuit at a fixed clock rate of 100 MHz when active.

(iv) Specify the behaviour of activity modes and switch between these with various duty cycles.

(v) Measure power and temperature in these modes.

To create a worst case active processing scenario, we use pseudo random number generators as test circuits [21]. These random number generators are based on binary linear recurrence where each bit of the next state is generated based on a linear combination of the current state. Compared to linear feedback shift registers (LFSRs), the most common type of random number generators, this improves quality of the random numbers. But for our purposes, the main advantage is the lack of optimisation potential in the circuit. This will minimise the influence of the implementation tools on the result of our characterisation. Using binary linear recurrence yields a circuit where each RNG state flip-flop is fed by a LUT which has its inputs connected to some other state flip-flops. An n-bit RNG will therefore map to exactly n LUTs and n flip-flops. This circuit does not provide any potential for logic optimisation and thus, eliminates the influence of the synthesis tools. The circuit is also characterised by LUTs that are heavily interconnected to seemingly random points. Hence, it does not provide any opportunity for optimised placement and routing other than concentrating the RNG circuit to the smallest possible area. It will also result in an implementation that will exercise all different kinds of short and long wires of the routing fabric.

The random number generator circuit is also characterised by a high and uniformly distributed toggle rate, and therefore suitable to act as worst case scenario of maximum activity in the fine grain fabric. The statistical chance of toggling on the rising clock edge is 50% for all flip-flops in the circuit. Hence, the total toggle rate of the circuit is 50%.

Currently, we use a 512-bit random number generator core that maps to 512 LUTs and 512 flip-flops. Since current FPGAs provide tens to hundreds of thousands of LUTs and flip-flops, we scale the size of the test circuit with the size of the device. To achieve high logic utilisation while still allowing routability, we implement multiple instances of the random number generators so that 90% of all logic resources are used. The resulting power consumption is normalised to the number of LUTs in order to allow a comparison of differently sized chips. If, however, 90% logic utilisation should lead to routing congestion and prevent the implementation tools from completing the design on other target devices, then the utilisation can be lowered. This should only affect active power and there should be minimal to no influence on inactive power or power in low-power modes.

The cores are driven by a 100 MHz clock when the circuit is active. This frequency simply acts as a reference point for a typical FPGA clock frequency. The power characteristics for different clock frequencies can be estimated by scaling the power consumption linearly to the clock rate. However, a benchmark user can also decide to run the circuit at a different clock frequency if the suggested 100 MHz clock seems unrealistic for their purposes. Varying utilisation or clock frequency limits the comparability with other published benchmark results but may yield in a more realistic evaluation in a particular domain.

To enable a comparison of devices with different low-power capabilities, we define the behaviour of activity modes. These activity modes specify how the device behaves in a certain mode rather than by which means this mode is implemented. The two basic modes that are applicable to all devices are *active* and *inactive*. In *active* mode, the test circuit continuously generates random numbers at 100 MHz clock frequency. The power consumption in this mode is a combination of static and high dynamic power. In *inactive* mode, the circuit does not generate random numbers. However, its state is preserved and it can be instantly brought back into active mode. These are the most basic activity modes and a transition between these two modes can usually be implemented with a simple clock gating approach as illustrated in Figure 1. However, we are only concerned about the power profile of the inactive device with preservation of state and instant wake-up capability, and not the details of its technical implementation. This mode corresponds to static power only, if clock gating is chosen as a method to implement this mode. However, depending on the implementation details, there might be supporting circuitry such as clock managers that are still operating and drawing dynamic power. It is important to point out that in this context, we are not necessarily interested in measuring pure static power but rather the minimal power to implement the *inactive* mode.

Table 2: Examples of activity modes. The first two modes are fixed, further modes can be defined based on the device capabilities.

	Activity mode			
	Standard		Device-specific	
	Active	Inactive	Sleep	Hibernate
Generate output	Yes	No	No	No
Retain state	—	Yes	Yes	No
Wakeup time	—	Instant	500 μs	50 ms

In order to evaluate devices with advanced low-power modes such as the ones in Table 1, we characterise the behaviour of the low-power mode and compare the power consumption with the two basic modes. Table 2 illustrates an example with our two basic activity modes *active* and *inactive*, and two hypothetical advanced modes *sleep* and *hibernate*. The behaviour of the basic modes is fixed, while the behaviour of the advanced modes depends on the device capabilities.

Since device temperature has a feedback effect on static power, we define a specific environment in order to reduce external influences on the measurement. The test environment is specified as a chip mounted on a PC board surrounded by ambient air with a temperature of 25°C. The board is placed in a large open cardboard enclosure which is supposed to reduce airflow to natural convection and reflect infrared radiation. No heatsinks or active cooling systems are to be used.

In the following, we distinguish between *cold* and *hot* devices in our characterisation. Devices which do not exceed a surface temperature of 35°C when being in active mode are characterised as *cold*. In this case, the influence of temperature is small and can be neglected. The power is measured in each activity mode and the results are reported normalised to LUTs.

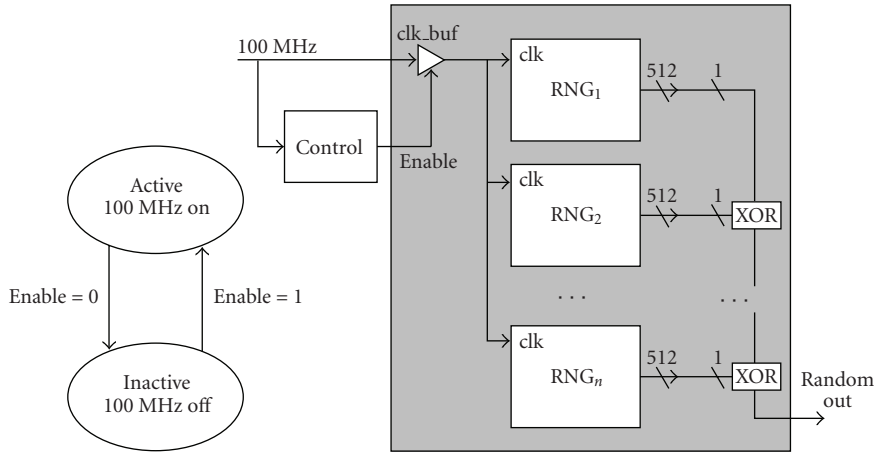


Figure 1: Fabric characterisation circuit implemented with random number generators.

For *hot* devices, we propose a more detailed analysis that takes the heating up of the device and its influence on power into account. Even though mobile applications are characterised by strict low-power constraints, thermal considerations are also important. In addition to power constraints, devices also have a thermal budget. Hence, we want to analyse how a device heats up for a given amount of activity.

Thermal characterisation of a device usually involves determining the thermal resistance of the device between die (also called junction) and case or junction and ambient air. For this purpose, a special test die is usually mounted in a case and heated up to a defined junction temperature T_j by applying power P. The junction-to-case thermal resistance θ_{jc} can then be calculated based on the following equation:

$$\theta_{jc} = \frac{T_j - T_c}{P}. \tag{2}$$

Equation (2) can also be used to calculate the case temperature T_c if θ_{jc} is specified by the device manufacturer. This however requires the knowledge of the junction temperature T_j. A further disadvantage is that (2) is not very accurate since it does not consider heatflow into the PC board or the ambient air. As an alternative, we propose to measure the case temperature as well as power consumption in our well-defined environment over the device activity. The activity in the device is adjusted by periodically switching the device between active and inactive state with various duty cycles. For each duty cycle we measure instantaneous active power, inactive power and temperature. To take these measurements, we first wait until the temperature has converged to its final value as illustrated in Figure 2. We record this temperature and then take a reading of active and inactive power. As mentioned earlier, inactive power does not necessarily have to be equivalent to static power and can have dynamic components as well. Nonetheless, we expect a strong temperature dependency on inactive power because of its close relation to static power. Likewise, we record the instantaneous active power that we expect to be less temperature dependent, since it is largely based on dynamic power. These measurements are taken for duty cycles from 0% to 100% with 5% increments. Results for temperature and normalised active and inactive power are reported over duty cycle. For devices with additional low-power modes, the power in these modes is measured at ambient temperature without any duty cycle.

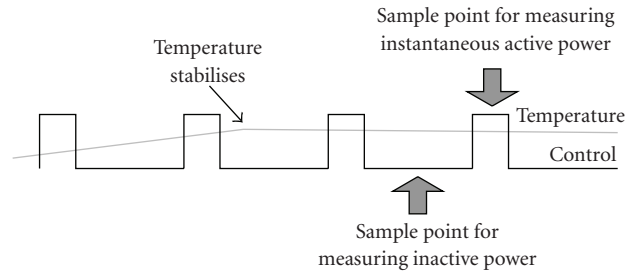


Figure 2: Depiction of power measurement sample points relative to temperature.

4. Implementation of the Fabric Characterisation Method

We implement our fabric characterisation in Xilinx Virtex and Spartan series FPGAs and one Silicon Blue iCE65 FPGA. The synthesis and mapping of the random number generator circuit for these devices is straightforward. Xilinx Virtex series and Spartan-3E FPGAs do not support any low-power modes other than simply stopping the clock. The most efficient way of setting the device into inactive mode as specified in Section 3 is to disable the clock tree using an internal clock buffer. This buffer is located at the root of the clock tree and therefore reduces the dynamic component of inactive power to a minimum. The enable signal of the clock buffer is connected to an external pulse generator with variable duty cycle. Spartan-3A and Spartan-3AN FPGAs also feature an advanced low-power mode called *suspend* [22]. This mode is controlled by an external signal pin and does not require modifications to the logic design itself.

Silicon Blue iCE65 FPGAs provide an iCEgate latch which will stop input signals from propagating into the device. In our case we use the iCEgate as a clock buffer, which is again controlled by an external pulse generator.

Figure 1 illustrates the implementation of our test circuit. The RNG cores only have a clock input and a 512-bit wide output. Each core is initialised with a different seed and one output pin of each core is connected via an XOR chain to avoid logic optimisation of the circuit. The reported LUT usage of the implementation tools should match the expected value of $512 \cdot n$, where n is the number of instantiated RNGs.

The power consumption of an FPGA can be obtained by measuring the current on the supply rails. However, it should be considered that FPGAs are sensitive to core voltage variations and usually have to stay within 50 mV of the nominal value. The voltage drop caused by the internal shunt resistor of a current meter can exceed this value if high currents are measured. It is therefore not recommended to wire a current meter directly into the supply rail. As an alternative, we insert precision current sense resistors directly into the rail and measure the voltage drop over the resistor. Our current sense resistors are in the milliohm range and have 1% tolerance. The resistor value is selected such that the voltage drop at maximum current is less than 50 mV. We measure the voltage drop over the resistor with a Hameg HM8012 digital multimeter and use this value to calculate current and power. Most FPGAs provide different voltage rails for core logic and IOs. For our fabric characterisation we only measure core power because IO power is highly dependent on pin loads of a given design.

We use an Optris MiniSightPlus infrared thermometer to determine the surface temperature of the device. This contactless method minimises the influence of the measurement on the system. Infrared thermometers work very well on matt, black plastic cases. However, they are inaccurate on shiny metal surfaces since these emit less infrared radiation. We therefore apply a thin blackened aluminium plate to devices with a metallic surface.

5. Results and Measurements

We perform a fabric characterisation as described in Section 3 on five commercially available FPGAs. Our measurements cover four Xilinx devices and one Silicon Blue device as listed in Table 3. The tested devices vary notably in their process technology and core voltage and thus have significantly different power profiles. Table 3 also shows the logic capacity of each device, the number of RNG cores used, and the resulting device usage which should be as close as possible to 90%. Both Spartan devices have less than half of the logic capacity than the Virtex devices. The Silicon Blue device is the smallest device with significantly less logic capacity than all others. The Spartan devices have plastic packages which have higher thermal resistances. This leads to increased junction temperatures for a given surface temperature. The Xilinx Virtex-II Pro and the Silicon Blue device have ceramic cases which provide better thermal conductivity than plastic. However the exact thermal resistance is not specified for the Silicon Blue device. Xilinx Virtex-5 FPGAs have metal cases which provide the best thermal conductivity. All boards have current measurement facilities except the ML505 board which was specifically modified to allow equally precise measurements. The designs targeting Xilinx device are implemented with Xilinx ISE 9.2. The design for the Silicon Blue FPGA is implemented with Silicon Blue iCEcube 2008 development software.

Table 4 shows the total and normalised inactive and active power for all devices. The table also lists the minimum temperature the device reaches when constantly being inactive and the maximum temperature the device reaches with full duty cycle of active processing. Inactive power at minimum temperature represents the case when the device is held in inactive state for a longer period of time and hence, is cooled down whereas inactive power at maximum temperature represents the case where the device is just switched off from active processing. Virtex-II Pro is the only device that cannot be run at full duty cycle without overheating. At 60% duty cycle, the device reaches a surface temperature of 74°C. Based on the thermal resistance and the power consumption, we estimate that this surface temperature corresponds to the maximum allowed junction temperature of 85°C. All other devices can be run at full duty cycles without overheating. The minimum and maximum case temperatures are shown in columns two and three. Table 4 also lists the total and normalised active power for maximum temperature, since this is the temperature the device operates under for full duty cycle.

The Silicon Blue iCE65 FPGA does not exceed a temperature of 29°C. The inactive and active power is not influenced by any temperature feedback of activity and the device can therefore be treated as a cold device. The low temperature can be explained with good thermal conductivity of the case and overall low total power. The normalised active power of the iCE65 device is similar to Virtex-5 which is manufactured in the same process technology. The inactive power in the iCE65 device is measured using iCEgate latch feature which can freeze the state of an IO bank. Outputs are kept at their current state and input signals do not propagate into the device. Internal switching activity is therefore stopped completely. This is not a dedicated low-power state by itself, but contributes to good power efficiency when being inactive. The inactive power of the iCE65 FPGA is significantly better than all other devices which can be explained by the prevention of internal switching activity by the iCEgate and the fact that the device is specifically optimised for low leakage. Optimising for low leakage however comes at the expense of performance. It is challenging to achieve timing closure for our test design on the iCE65 FPGA while all other Xilinx devices provided good performance headroom. For all Xilinx devices, we can observe a temperature feedback of the active processing on inactive power. For Virtex-II Pro, we find a ninefold increase of inactive power between minimum and maximum temperature. In Xilinx devices, we can also observe an improvement of normalised active power with modern process technology while at the same time, inactive power deteriorates. In the following we further analyse the four Xilinx devices using our duty cycle measurement method.

Table 3: Details of FPGAs used for our fabric characterisation experiment.

FPGA family	Device	Board	Process technology	Core voltage	Thermal resistance Θ_{JC}	Number of LUTs/FFs	Number of RNGs	Logic utilisation
Xilinx Virtex-II Pro [9]	XC2VP30	XUP	130 nm	1.5 V	0.6°C/W	27,392	48	89.7%
Xilinx Spartan-3E [17]	XC3S500E	Spartan-3E Starter Kit	90 nm	1.2 V	9.8°C/W	9,312	16	88%
Xilinx Spartan-3AN [17]	XC3S700AN	Spartan-3AN Starter Kit	90 nm	1.2 V	5.3°C/W	11,776	21	91.3%
Xilinx Virtex-5 [23]	XC5VLX50T	ML505	65 nm	1.0 V	0.2°C/W	28,800	50	88.9%
Silicon Blue iCE65 [20]	iCE65L04	iCEman Eval Kit	65 nm	1.2 V	n/a	3,520	6	87.3%

Table 4: Power measurement results. Shown is total and normalised inactive and active power.

			Inactive mode				Active mode	
Device	T_{min}	T_{max}	P_{total} (T_{min}) [mW]	P_{norm} (T_{min}) [μW/LUT]	P_{total} (T_{max}) [mW]	P_{norm} (T_{max}) [μW/LUT]	P_{total} (T_{max}) [W]	P_{norm} (T_{max}) [μW/LUT]
V2P30	26°C	74°C	24.0	0.88	216.0	7.89	11,190	455.3
S3E500	26°C	46°C	19.2	2.06	26.2	2.81	1,022	124.8
S3AN700	26°C	47°C	18.7	1.59	28.8	2.45	1,363	126.8
V5LX50T	30°C	54°C	349.0	12.12	412.1	14.31	2,545	99.4
iCE65L04	25°C	29°C	0.13	0.037	0.13	0.037	324	105.5

Figure 3 shows the surface temperature of the Xilinx devices over the duty cycle. The temperature is measured with an infrared thermometer as explained in Section 4. For all devices the temperature increases almost linearly. Virtex-II Pro has the steepest slope and reaches a surface temperature of 74°C at 60% duty cycle. Spartan-3E, Spartan-3AN and Virtex-5 reach a surface temperature of 46°C, 47°C, and 54°C, respectively, when on full duty cycle.

Figure 4 illustrates the inactive power consumption normalised per LUT over duty cycle. Each point on the line is measured when the temperature of the FPGA has stabilized for a given duty cycle as illustrated in Figure 2. We observe that inactive power consumption, which in our implementation is almost equivalent to static power, increases with the duty cycle. This is due to the rising temperature of the device caused by heat dissipated during the active part of the duty cycle. The Virtex-5 device, which is manufactured in 65 nm process, has a 12 times higher inactive power consumption under cold conditions (0% duty cycle) than the 130 nm Virtex-II Pro. This can be explained with worsened static power in the smaller process technology but unused hard IP blocks may also have an influence. The inactive power in Virtex-II Pro deteriorates quickly with higher duty cycles because of the high device temperature as illustrated in Figure 3. Virtex-5 and the Spartan-3 devices develop less heat which leads to a less progressive increase in inactive power. Of all four Xilinx FPGAs, Spartan-3 devices show the overall best efficiency during inactive phases.

Figure 5 shows the normalised active power consumption for all Xilinx devices. This is the instantaneous active power during on-phase of the duty cycle as illustrated in Figure 2. We can observe a notable improvement in active power for newer devices which is due to feature and voltage scaling in the process technology. The improvement from Virtex-II Pro (130 nm) to Spartan-3 (90 nm) is especially noteworthy. Compared to Virtex-5, the active power is reduced by more than a factor of 4. The active power consumption is relatively independent of duty cycle and temperature, although, this could change if static power becomes a more dominant component in active power. In current devices, we find that inactive power is considerably less than active power although the ratio increases from 0.2% to 14% between Virtex-II Pro and Virtex-5 in cold conditions, or from 1.5% to 16% in hot conditions. On the other hand, devices can be optimised for low leakage by trading off performance. The Silicon Blue device, for example, has an inactive-to-active power ratio of 0.04%.

The only device in our test that features a dedicated low-power mode is Spartan-3AN that provides a suspend mode. This mode reduces the power consumption of all auxiliary circuits powered on the VCC_{aux} rail [22]. The logic state is preserved during suspend mode and the wake-up time

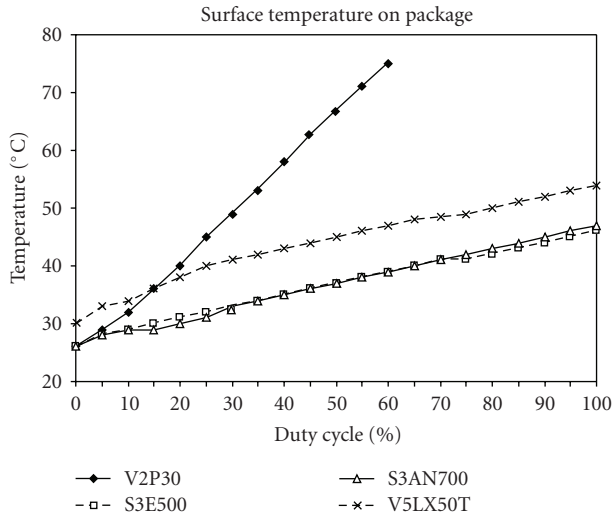


Figure 3: Variation of device surface temperature with duty cycle.

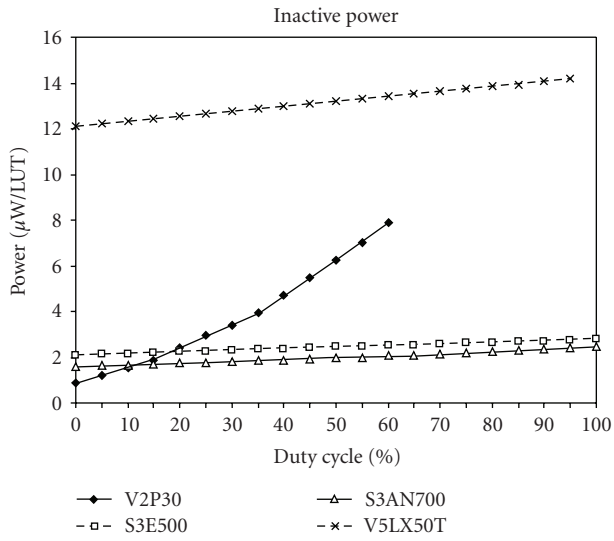


Figure 4: Variation of inactive power consumption with duty cycle.

Table 5: Total core power in a Xilinx Spartan-3AN 700 FPGA for active, inactive, and suspend modes. All values are measured at 25°C.

Power	Active mode	Inactive mode	Suspend mode
P_{int}[mW]	1349	18.7	18.1
P_{aux}[mW]	44	43.6	5.8
P_{total}[mW]	1393	62.3	23.9

ranges between 100 μs and 500 μs. Table 5 illustrates the core power consumption in all modes. Compared to the inactive mode, the suspend mode reduces the power consumption by a factor of 3.

In an overall comparison, the Silicon Blue iCE05 FPGA is the most power efficient device. It consumes slightly more active power than Virtex-5 but significantly less inactive power than all other devices. However, it has the lowest

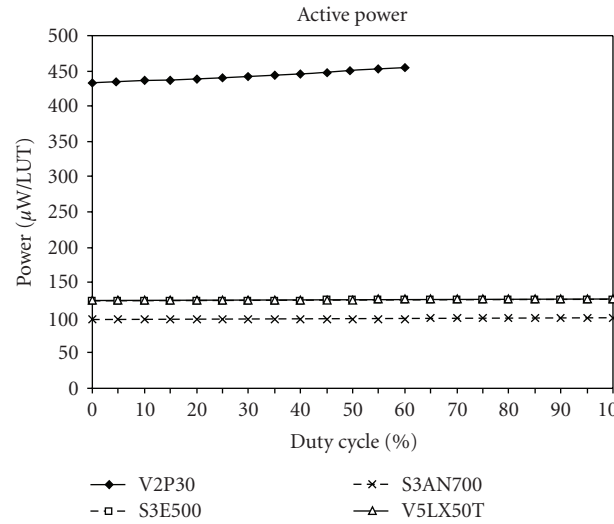


Figure 5: Variation of instantaneous active power consumption with duty cycle.

logic capacity and was specifically optimised for low leakage and has therefore less performance. The device performance is not addressed in this fabric characterisation method but can be evaluated using the additional power benchmarks presented in [4]. From Xilinx devices, we can also observe a general trend of reduction of active power with advancing process technology while inactive power is becoming more challenging. Dedicated low-power modes are needed to address the issue of power consumption during periods of inactivity. Spartan-3AN is the only device in our experiment that features a low-power mode. The inactive power reduction by a factor of 3 represents a good improvement but a power consumption of 23.9 mW in suspend mode is too high for most power budgets in mobile applications.

All our experiments are performed on just one particular device of each type and we therefore have no information on the variability of the results depending on process variation. To make the measurements more representative, the benchmark can be repeated and averaged with several devices of the same type.

6. Conclusions and Future Work

In this paper, we provide a new, application independent methodology for the fabric characterisation of fine-grain FPGAs. This methodology is useful in evaluating the active and inactive power consumption as well as advanced low-power modes. We describe procedures for measuring active and inactive power and temperature on FPGAs using a simple experimental setup. The key to this setup is the use of random number generators as a highly active circuit.

To illustrate our methodology, we perform the fabric characterisation for four Xilinx FPGAs and one Silicon Blue FPGA. Our comparison of Xilinx and Silicon Blue shows that active power is similar for the same technology node but inactive power is significantly improved in the case of Silicon Blue. This is because the Silicon Blue device is specifically

optimised for low leakage trading off device performance. Another optimisation is the reduction of internal switching activity in inactive states. Our measurements also show how advances in process technology reduce the active power by more than a factor of 4. We further observe an increase of inactive power by up to one order of magnitude. However, modern devices generate less heat per activity and suffer less from temperature-based deterioration of inactive power. Additionally, we measure one specific low-power mode in Spartan-3AN that reduces the inactive power by a factor of 3. These improvements are noteworthy, but advances through process technology alone are not enough to meet the strict power constraints in mobile devices. In particular, the power consumption during inactive periods needs to be addressed. To meet mobile power requirements, future devices need flexible and more effective low-power modes as well as techniques addressing the leakage in next-generation process technologies.

Our proposed methodology is part of a power benchmarking framework [3] which also covers further application-specific test cases that are representative of computations in mobile devices. Current and future work includes studying a wide range of devices and characterising hardened IP blocks, and their performance in various low-power applications.

References

[1] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

[2] T. Tuan, A. Rahman, S. Das, S. Trimberger, and S. Kao, "A 90-nm low-power FPGA for battery-powered applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 296–300, 2007.

[3] "Groundhog benchmark suite," Tech. Rep., Imperial College, London, UK, 2009, http://cc.doc.ic.ac.uk/projects/GROUNDHOG/.

[4] P. Jamieson, T. Becker, W. Luk, P. Y. K. Cheung, T. Rissa, and T. Pitkänen, "Benchmarking reconfigurable architectures in the mobile domain," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 1–8, Napa, Calif, USA, April 2009.

[5] C. Piguet, *Low-Power CMOS Circuits*, CRC Press, Boca Raton, Fla, USA, 2005.

[6] O. S. Unsal, J. W. Tschanz, K. Bowman, et al., "Impact of parameter variations on circuits and microarchitecture," *IEEE Micro*, vol. 26, no. 6, pp. 30–39, 2006.

[7] V. George, H. Zhang, and J. Rabaey, "The design of a low energy FPGA," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '99)*, pp. 188–193, San Diego, Calif, USA, August 1999.

[8] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex™-II FPGA family," in *Proceedings of the 10th ACM International Symposium on Field Programmable Gate Arrays (FPGA '02)*, pp. 157–164, Monterey, Calif, USA, February 2002.

[9] "*Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*," Xilinx Inc., May 2007.

[10] A. Gayasen, K. Lee, V. Narayanan, M. Kandemir, M. J. Irwin, and T. Tuan, "A dual-VDD low power FPGA architecture," in *Proceedings of the 14th International Conference on Field Programmable Logic and Its Application (FPL '04)*, vol. 3203, pp. 145–157, Leuven, Belgium, August-September 2004.

[11] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A reconfigurable, power-efficient adaptive Viterbi decoder," *IEEE Transactions on VLSI Systems*, vol. 13, no. 4, pp. 484–488, 2005.

[12] J. Becker, M. Hübner, and M. Ullmann, "Power estimation and power measurement of Xilinx Virtex FPGAs: trade-offs and limitations," in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI '03)*, pp. 283–288, Sao Paulo, Brazil, September 2003.

[13] S. Lopez-Buedo, J. Garrido, and E. Boemo, "Thermal testing on reconfigurable computers," *IEEE Design and Test of Computers*, vol. 17, no. 1, pp. 84–91, 2000.

[14] J. Lamoureux and S. J. E. Wilton, "On the trade-off between power and flexibility of FPGA clock networks," *ACM Transactions Reconfigurable Technology and Systems*, vol. 1, no. 3, pp. 1–33, 2008.

[15] "*Power vs. Performance: The 90 nm Inflection Point*," White Paper, Xilinx Inc., 2006.

[16] "*40-nm FPGA Power Management and Advantages*," White Paper, Altera Inc., 2008.

[17] "*Spartan-3 Generation FPGA User Guide v.1.2*," Xilinx Inc., April 2007.

[18] "*MachXO Family Data Sheet*," Lattice, June 2009.

[19] "*Igloo Handbook*," Actel, April 2009.

[20] "*iCE65 Ultra Low-Power Programmable Logic Family Data Sheet*," SiliconBlue, June 2009.

[21] D. B. Thomas and W. Luk, "High quality uniform random number generation using LUT optimised state-transition matrices," *The Journal of VLSI Signal Processing*, vol. 47, no. 1, pp. 77–92, 2007.

[22] "*Using Suspend Mode in Spartan-3 Generation FPGA*," Xilinx Inc., May 2007.

[23] "*Virtex-5 Family Platfrom Overview LX and LXT Platforms v2.2*," Xilinx Inc., January 2007.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 219764, 14 pages
doi:10.1155/2010/219764

Research Article

High-Speed FPGA 10's Complement Adders-Subtractors

G. Bioul,[1,2] M. Vazquez,[2] J. P. Deschamps,[1,3] and G. Sutter[4]

[1] *Faculty of System Engineering, FASTA University, Mar del Plata, Argentina*
[2] *Faculty of System Engineering, UNCPBA University, Tandil, Argentina*
[3] *School of Engineering, Rovira I Virgili University, Tarragona, Spain*
[4] *School of Engineering, Universidad Autónoma de Madrid, Madrid, Spain*

Correspondence should be addressed to G. Sutter, gustavo.sutter@uam.es

Received 3 June 2009; Accepted 22 October 2009

Academic Editor: Elías Todorovich

This paper first presents a study on the classical BCD adders from which a carry-chain type adder is redesigned to fit within the Xilinx FPGA's platforms. Some new concepts are presented to compute the *P* and *G* functions for carry-chain optimization purposes. Several alternative designs are presented. Then, attention is given to FPGA implementations of add/subtract algorithms for 10's complement BCD numbers. Carry-chain type circuits have been designed on 4-input LUTs (Virtex-4, Spartan-3) and 6-input LUTs (Virtex-5) Xilinx FPGA platforms. All designs are presented with the corresponding time performance and area consumption figures. Results have been compared to straight implementations of a decimal ripple-carry adder and an FPGA 2's complement binary adder-subtractor using the dedicated carry logic, both carried out on the same platform. Better time delays have been registered for decimal numbers within the same range of operands.

1. Introduction

In a number of computer arithmetic applications, decimal systems are preferred to the binary ones. The reasons come not only from the complexity of coding/decoding interfaces but mostly from the lack of precision and clarity in the results of the binary systems.

Decimal arithmetic plays a key role in data processing environments such as commercial, financial, and Internet-based applications [1–3]. Performances required by applications with intensive decimal arithmetic are not met by most of the conventional software-based decimal arithmetic libraries [1]. Hardware implementation embedded in recently commercialized general purpose processors [3, 4] is gaining importance.

Furthermore, *IEEE* has recently published a new standard 754-2008 [5] that supports the floating point representation for decimal numbers.

At the moment, Binary Coded Decimal (BCD) is used for decimal arithmetic algorithm implementations. Although other coding systems may be of interest, BCD seems to be the best choice until now. Issues of hardware realization of decimal arithmetic units appear to be widely open: potential improvements are expected in what refers to algorithm concepts as well as to hardware design. This paper resumes some new concepts about carry-chain type algorithms for adding BCD numbers. Two key ideas have been introduced: (i) the Propagate *P* and generate *G* functions are computed from the input data instead of intermediate BCD sums, and (ii) the functions have been implemented in Xilinx Virtex-4 [6] and Virtex-5 *FPGA* platforms [7], taking advantage of the 6-input LUTs structure of Virtex-5 version.

Signed numbers addition is used as a primitive operation for computing most arithmetic functions, so that it deserves particular attention. It is well known that in classical algorithms the execution time of any program or circuit is proportional to the number *N* of digits of the operands. In order to minimize the computation time, several ideas have been proposed in the literature [8, 9]. Most of them consist in modifying the classical algorithm in such a way as to minimize the computation time of each carry; the time complexity may still be proportional to *N*, but the proportionality constant may be reduced. Moreover, it has to be pointed out that, within the same range, decimal addition involves shorter carry propagation process than for the straight binary code. It will be shown in the practical

implementations that adding BCD digits can not only save coding interfaces but moreover provides time delay reductions. Hardware consumption for BCD will be greater, if coding and decoding processes are not considered; as of today, the dramatic decreasing of hardware cost stimulates work on time saving.

In this paper, decimal carry-chain and ripple-carry adders have been implemented on Virtex-4 Xilinx *FPGA* platforms, for a number of operand sizes; comparative performances are presented for binary and BCD digit operands.

Additionally, three implementations of adders-subtractors have been implemented on FPGA Xilinx Virtex-5 platforms for a number of operand sizes; comparative performances are presented for binary and BCD digit operands, respectively. Adder-subtractor inputs are 10's complement signed BCD numbers; sign-change algorithm is used whenever subtraction is at hand.

2. Base-B Ripple-Carry Adders

Consider the base-B representations of two n-digit numbers:

$$\begin{aligned} x &= x_{n-1}\cdot B^{n-1} + x_{n-2}\cdot B^{n-2} + \cdots + x_0\cdot B^0, \\ y &= y_{n-1}\cdot B^{n-1} + y_{n-2}\cdot B^{n-2} + \cdots + y_0\cdot B^0. \end{aligned} \tag{1}$$

Algorithm 1 (pencil and paper) computes the $(n+1)$-digit representation of the sum $z = x + y + c_{\text{in}}$ where c_{in} is an initial carry equal to 0 or 1.

Algorithm 1. Classic addition (ripple carry):

$c(0) := c\_\text{in}$;
for i **in** $0 \cdots n-1$ **loop**
  if $x(i) + y(i) + c(i) > B - 1$ **then** $c(i+1) := 1$;
  else $c(i+1) := 0$; **end if**;
  $z(i) := (x(i) + y(i) + c(i)) \bmod B$;
end loop;
$z(n) := c(n)$;

As $c(i+1)$ is a function of $c(i)$, the execution time of Algorithm 1 is proportional to n (Figure 1). In order to reduce the execution time of each iteration step, Algorithm 1 can be modified as shown in Section 3.

3. Base-B Carry-Chain Adders

First define two binary functions of two B-valued variables, namely, the *propagate* (P) and *generate* (G) functions:

$$\begin{aligned} P(i) &\equiv \begin{cases} P(x(i), y(i)) = 1 & \text{if } x(i) + y(i) = B - 1, \\ P(x(i), y(i)) = 0 & \text{otherwise;} \end{cases} \\ G(i) &\equiv \begin{cases} G(x(i), y(i)) = 1 & \text{if } x(i) + y(i) > B - 1, \\ G(x(i), y(i)) = 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{2}$$

The next carry c_{i+1} can be calculated as follows:

if $P(i) = 1$ **then** $c(i+1) := c(i)$;
else $c(i+1) := G(i)$; **end if**;

The corresponding modified Algorithm 2 is the following one.

Algorithm 2. Carry-chain addition

– computation of the generation and propagation conditions:
for i **in** $0 \cdots n-1$ **loop**
  $G(i) := G(x(i), y(i))$;

  $P(i) := P(x(i), y(i))$;
end loop;
– carry computation:
$c(0) := c\_\text{in}$;
for i **in** $0 \cdots n-1$ **loop**

  if $P(i) = 1$ **then** $c(i+1) := c(i)$;
  else $c(i+1) := G(i)$; **end if**; (3)

end loop;
– sum computation
for i **in** $0 \cdots n-1$ **loop**
  $z(i) := (x(i) + y(i) + c(i)) \bmod \text{B}$;
end loop;
$z(n) := c(n)$;

Comments.

(1) Instruction sentence (3) is equivalent to the following Boolean equation:

$$c(i+1) = P(i)\cdot c(i) \vee \text{not } (P(i))\cdot G(i). \tag{4}$$

Furthermore, if the preceding relation is used, then the definition of the generate function can be modified:

$$\begin{aligned} G(i) &= 1 \quad \text{if } x(i) + y(i) > B - 1, \\ G(i) &= 0 \quad \text{if } x(i) + y(i) < B - 1, \\ G(i) &= 0 \text{ or } 1 \quad \text{(do not care) otherwise.} \end{aligned} \tag{5}$$

(2) Another Boolean equation equivalent to (4) is

$$c(i+1) = G(i) \vee P(i)\cdot c(i). \tag{6}$$

If the preceding relation is used, then the definition of the propagate function can be modified:

$$\begin{aligned} P(i) &= \begin{cases} 1 & \text{if } x(i) + y(i) = B - 1, \\ 0 & \text{if } x(i) + y(i) < B - 1, \end{cases} \\ P(i) &= 0 \text{ or } 1 \quad \text{(do not care) otherwise.} \end{aligned} \tag{7}$$

The structure of an n-digit adder with separate carry calculation is shown in Figure 2. It is based on Algorithm 2. The *G-P* (*Generate-Propagate*) cell calculates the *Generate* and *Propagate* functions (2).

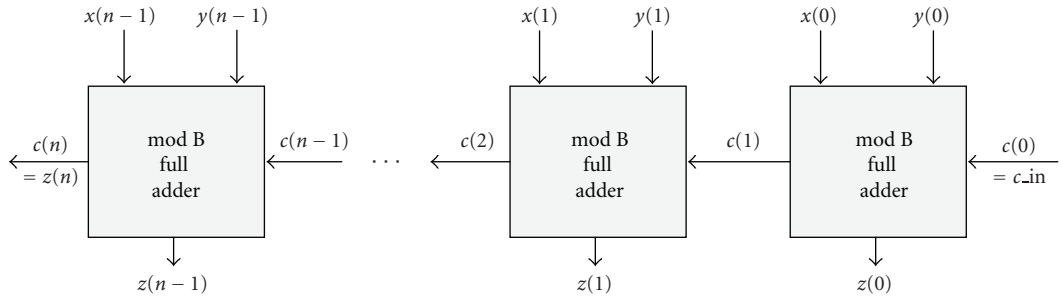


Figure 1: Ripple-Carry adder.

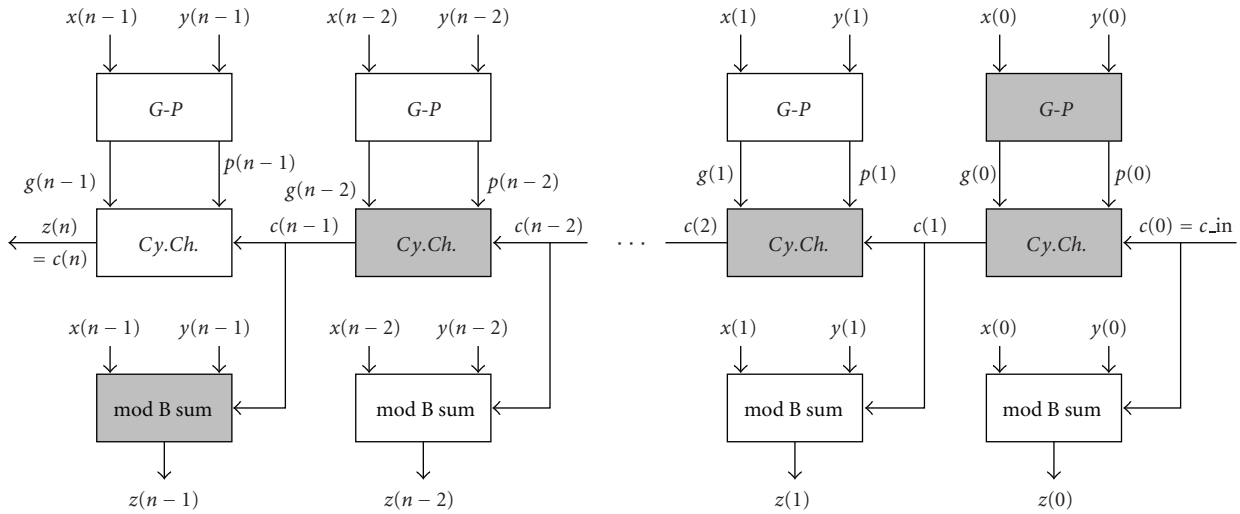


Figure 2: Carry-chain adder.

The *Cy.Ch* (*carry-chain*) cell computes the next carry, that is to say

$$c(i+1) = \begin{cases} c(i) & \text{if } P(i) = 1, \\ G(i) & \text{otherwise,} \end{cases} \tag{8}$$

so that $G(i)$ generates a carry, whatever happens upstream in the carry-chain, and $P(i)$ propagates the carry from level $i-1$. The mod B *sum* cell calculates

$$z(i) = (x(i) + y(i) + c(i)) \bmod B. \tag{9}$$

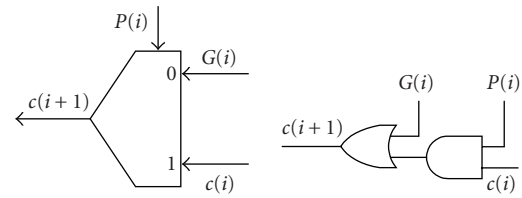
As regards the computation time T, the critical path is shaded in Figure 2. It has been assumed that $T_{\text{sum}} > T_{\text{Cy.Ch}}$.

Another interesting time is the delay $T_{\text{carry}}(n)$ from $c(0)$ to $c(n)$ assuming that all propagate and generate functions have already been calculated:

$$T_{\text{carry}}(n) = n \cdot T_{\text{mux2-1}}. \tag{10}$$

Comments. The carry-chain cells are binary circuits, whereas the *generate-propagate* and the mod B *sum* cells are B-ary ones.

Equation (4) can be implemented by a 2-to-1 binary multiplexer (Figure 3(a)) while (6) by a 2-gate circuit (Figure 3(b)). In the first case, the *per-digit-delay* of a carry-chain adder is equal to the delay $T_{\text{mux2-1}}$ of a 2-to-1 binary multiplexer, whatever the base B is.



(a) Carry multiplexer (b) AND-OR circuit

Figure 3: Carry-chain cells.

If $B = 2$ and the carry-chain cell of Figure 3(a) is used, then $P(i) = x(i) \oplus y(i)$ and $G(i)$ can be chosen equal to, for example, $y(i)$. The corresponding cell for a n-bit binary adder is shown in Figure 4.

4. Base-10 Complement and Addition

4.1. Ten's Complement Numeration System. B's complement representation general principles are available in the literature as, for example, computer arithmetic books such as

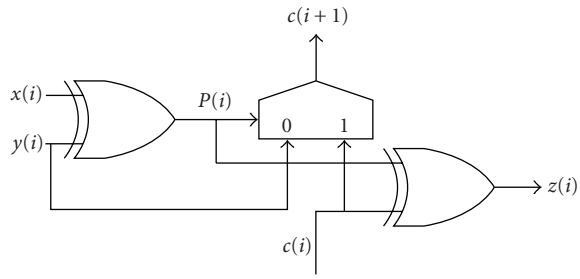


Figure 4: Binary adder cell.

[8, 9]. One restricts to 10's complement system to cope with the needs of this paper. A one-to-one function $R(x)$, associating a natural number to x, is defined as follows.

Every integer x belonging to the range

$$-\frac{10^n}{2} \le x < \frac{10^n}{2} \tag{11}$$

is represented by $R(x) = x \bmod 10^n$, so that the integer represented in the form $x_{n-1}\, x_{n-2} \cdots x_1\, x_0$ is

$$\begin{aligned} &x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0 \\ &\quad \text{if } x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0 < \frac{10^n}{2}, \\ &x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0 - 10^n \\ &\quad \text{if } x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0 \ge \frac{10^n}{2}. \end{aligned} \tag{12}$$

The conditions (12) may be more simply expressed as

$$\begin{aligned} &x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0 \quad \text{if } x_{n-1} < 5, \\ &x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0 - 10^n \quad \text{if } x_{n-1} \ge 5. \end{aligned} \tag{13}$$

Another way to express a 10's complement number is

$$x'_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \cdots + x_0, \tag{14}$$

where $x'_{n-1} = x_{n-1} - 10$ if $x_{n-1} \ge 5$ and

$$x'_{n-1} = x_{n-1} \quad \text{if } x_{n-1} < 5, \tag{15}$$

while the sign definition rule is the following one: if x is negative, then $x_{n-1} \ge 5$; otherwise $x_{n-1} < 5$.

4.2. Ten's Complement Sign Change. Given an n-digit 10's complement integer x, the inverse $z = -x$ of x is an $(n+1)$-digit 10's complement integer. Actually the only case that $-x$ cannot be represented with n digits is when $x = -10^n/2$, so $-x = 10^n/2$, that is to say $-x = 0.10^n + (5) \cdot 10^{n-1} + 0.10^{n-2} + \cdots + 0.10^0$). The computation of the representation of $-x$ is based on the following property.

Assuming x to be represented as an n-digit 10's complement number $R(x)$, $-x$ may be readily computed as

$$-x = 10^{n+1} - R(x). \tag{16}$$

A straightforward inversion algorithm then consists in representing x with $n+1$ digits, complementing every digit to 9, then adding 1. Observe that sign extension is obtained by adding a digit 0 to the left of a positive number or 9 for a negative number, respectively.

5. Base-10 Adders

5.1. Base-10 Ripple-Carry Adders. For $B = 10$, the classic and naïve approach [8] of ripple-carry for a BCD decimal adder cell can be implemented as in Figure 5. Observe that the critical path involves the carry propagation through 7 binary adders plus a 4-bit Boolean circuit (checking if the sum **s** is greater than 9 or not).

5.2. Base-10 Carry-Chain Adders. If $B = 10$, the carry-chain circuit remains unchanged but the P and G functions as well as the modulo-10 sums are somewhat more complex. In base 2, the mod B *sum* cell appears to be a single XOR function, while the mod 10 *sum* cell is more complex as suggested by Figure 5.

In base 2, the P and G cells are, respectively, synthesized by XOR and AND functions, while in base 10, P and G are now defined as follows:

$$\begin{aligned} P(i) &\equiv \begin{cases} P(x(i), y(i)) = 1 & \text{if } x(i) + y(i) = 9, \\ P(x(i), y(i)) = 0 & \text{otherwise;} \end{cases} \\ G(i) &\equiv \begin{cases} G(x(i), y(i)) = 1 & \text{if } x(i) + y(i) > 9, \\ G(x(i), y(i)) = 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{17}$$

A straightforward way to synthesize P and G is shown at Figure 6. Nevertheless, functions P and G may be directly computed from $x(i)$ and $y(i)$ inputs. The following formulas (18) are Boolean expressions of conditions (17),

$$\begin{aligned} P(i) &= p_0 \cdot [k_1 \cdot (p_3 \cdot k_2 \vee k_3 \cdot g_2) \vee g_1 \cdot k_3 \cdot p_2] \\ G(i) &= g_0 \cdot [p_3 \vee g_2 \vee p_2 \cdot g_1] \vee g_3 \vee p_3 \cdot p_2 \vee p_3 \cdot p_1 \\ &\quad \vee g_2 \cdot p_1 \vee g_2 \cdot g_1 \end{aligned} \tag{18}$$

where $p_j = x_j \oplus y_j$, $g_j = x_j \cdot y_j$, and $k_j = x'_j \cdot y'_j$ are the binary propagator, generator, and carry-kill for the jth components of the BCD digits $x(i)$ and $y(i)$.

The BCD carry-chain adder ith cell is shown at Figure 7. It is made of a first mod 16 adder stage, a carry-chain cell driven by the G-P functions, and an output adder stage performing a correction (adding 6) whenever the carry-out is one. Actually, a zero carry-out $c(i+1)$ identifies that the mod 16 sum does not exceed 9 if $c(i) = 0$, respectively, 8 if $c(i) = 1$; so no corrections are needed. Otherwise, the *add*-6 correction applies.

The G-P functions may be computed according to Figure 6, using the outputs of the mod 16 stage, including the carry-out s_4. With more hardware consumption, but saving time delays, formulas (18) may be used.

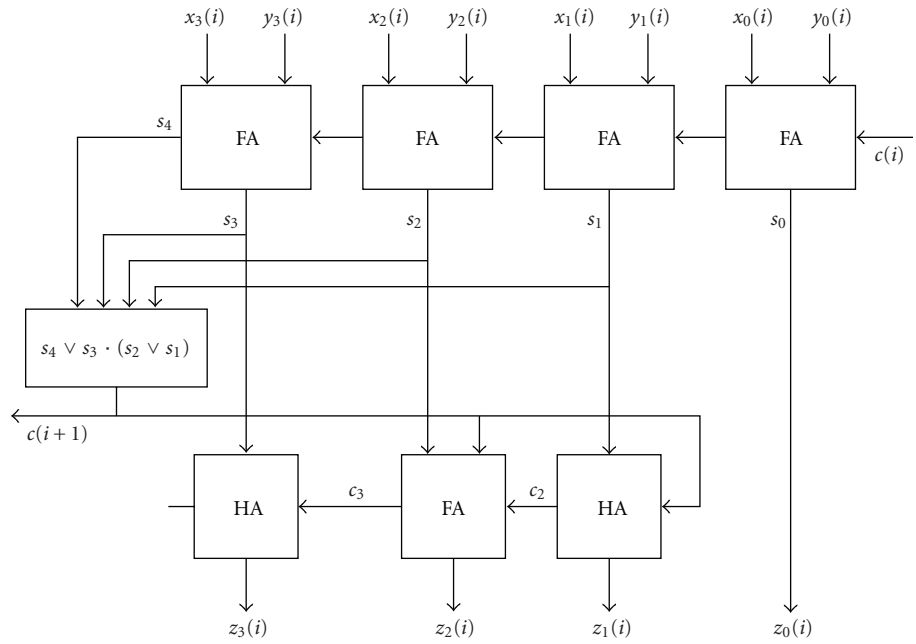


Figure 5: Ripple-Carry BCD adder cell.

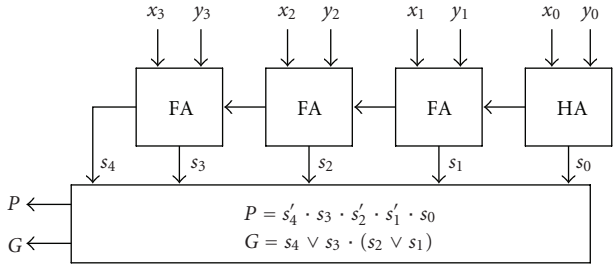


Figure 6: *G-P* cell for BCD adder.

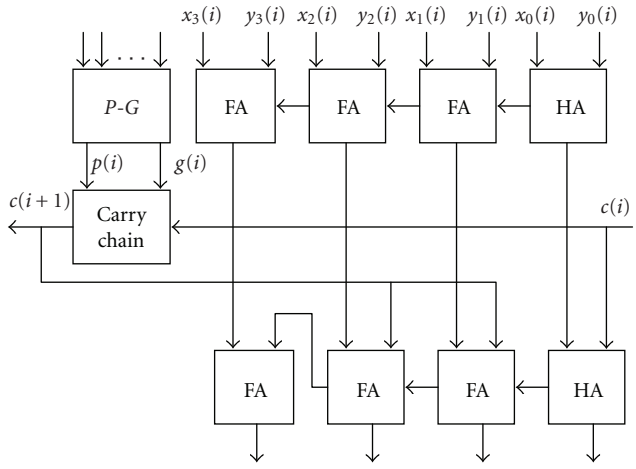


Figure 7: Carry-chain BCD adder *i*th cell.

6. FPGA Implementations of the Base-10 Adders on 4-Input LUTs Xilinx Platforms

The base-10 adders of Figures 5 and 7 have been implemented on 4-LUTs (Look-Up Tables up to 4 inputs) Xilinx devices. Virtex-4, Spartan 3, and the obsolete Virtex-2, Virtex and Spartan 2 are 4-input LUT*s*-based FPGA [6, 10]. In what follows the area is expressed in LUT*s*. In the Xilinx Virtex-4 technology a configurable logic block (*CLB*) involves 4 *slices* and *a slice* is made by two 4-LUT*s* and some additional logic. VHDL models are available at [11].

6.1. Base-10 Ripple-Carry Adder. The classic implementation of the ripple carry adder cell in *FPGA* implies a 4-bit adder, a 4-LUT to detect the carry condition, and a final 3-bit adder. The delay and area consumption of an N-digit ripple carry adder are

$$T_{N\text{-digit-}B10\text{-rc-adder}} = N \cdot \Big[T_{\text{LUT}} + 4 \cdot T_{\text{mux-cy}} + T_{\text{XOR}} + T_{\text{con}} + T_{\text{LUT}} + T_{\text{con}} + T_{\text{LUT}} + 3 \cdot T_{\text{mux-cy}} + T_{\text{XOR}} + T_{\text{con}}\Big], \quad (19)$$

$$C_{N\text{-digit-}B10\text{-rc-adder}} = 8 \cdot N \text{ LUTs}.$$

6.2. FPGA Implementation of the Base-10 Carry-Chain Adder. In order to make the best use of the resources, the design has been achieved using relative location techniques (*RLOC*) [12] with low-level component instantiations. This first architecture is called *GP_a*.

The adding stages are implemented as shown at Figures 8(a) and 8(b) while the carry-chain structure with the *G-P* functions has been implemented as shown at Figure 9 where G is computed according to Figure 6, while P is computed as

$$P = s_3 \cdot s_0 \cdot G' \quad (20)$$

equivalent to the expression of Figure 6. Figure 9 emphasizes that G depends on s_1, s_2, s_3, and s_4 while P is computed from s_0, s_3, and G.

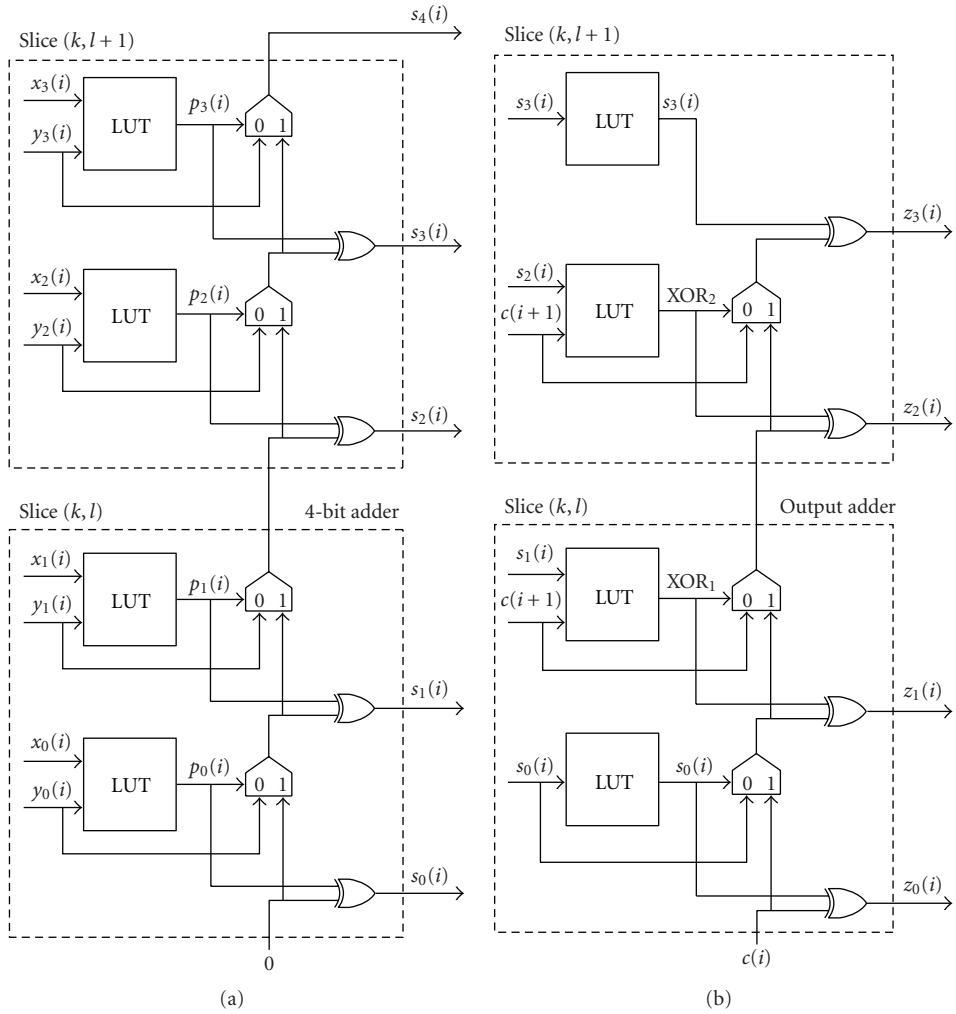


Figure 8: *FPGA* implementation of adders: (a) 4-bit adder stage and (b) output adder stage.

The time delay corresponding to the 4-bit adder stage (Figure 8(a)) and the output adder stage (Figure 8(b)) is given as

$$T_{\text{4-bit adder}} = T_{\text{LUT}} + 4 \cdot T_{\text{mux-cy}}, \tag{21}$$

$$T_{\text{output adder}} = T_{\text{LUT}} + 3 \cdot T_{\text{mux-cy}} + T_{\text{XOR}}. \tag{22}$$

Both adder stages of Figures 8(a) and 8(b) need the same hardware requirement; computed in slices, the area consumption is given as

$$C_{\text{4-bit adder}} = C_{\text{output adder}} = 4\,\text{LUTs}. \tag{23}$$

The complexity figures of the carry-chain circuit for a 4-digit unit, as shown at Figure 9, are given as

$$T_{\text{Cy-Ch-}a} = 2 \cdot T_{\text{LUT}} + T_{\text{con1}} + 4 \cdot T_{\text{mux-cy}}, \tag{24}$$

$$C_{\text{Cy-Ch-}a} = 8\,\text{LUTs}, \tag{25}$$

where T_{con1} stands for the average connection delay between two neighboring *slices* of the same *CLB*.

The overall circuit is represented in Figure 10. The overall time delay is computed from formulas (21), (22) and (24):

$$T_{N\text{-digit adder-}a} = 4 \cdot T_{\text{LUT}} + (N+7) \cdot T_{\text{mux-cy}} + T_{\text{XOR}} + T_{\text{con1}} + 2 \cdot T_{\text{con2}}, \tag{26}$$

where T_{con2} stands for the average connection delays between two *slices* located in neighbor columns. T_{con2} has to be accounted twice to involve both the connection delay between the 4-bit adder and the carry-chain and the one between the carry chain and the output adder.

From (23) and (25), the area requirement may be computed as

$$C_{N\text{-digit adder-}a} = 10 \cdot N\,\text{LUTs}. \tag{27}$$

6.3. Other Implementations of Base-10 Carry-Chain Adders. Functions P and G may be directly computed from $x(i)$ and

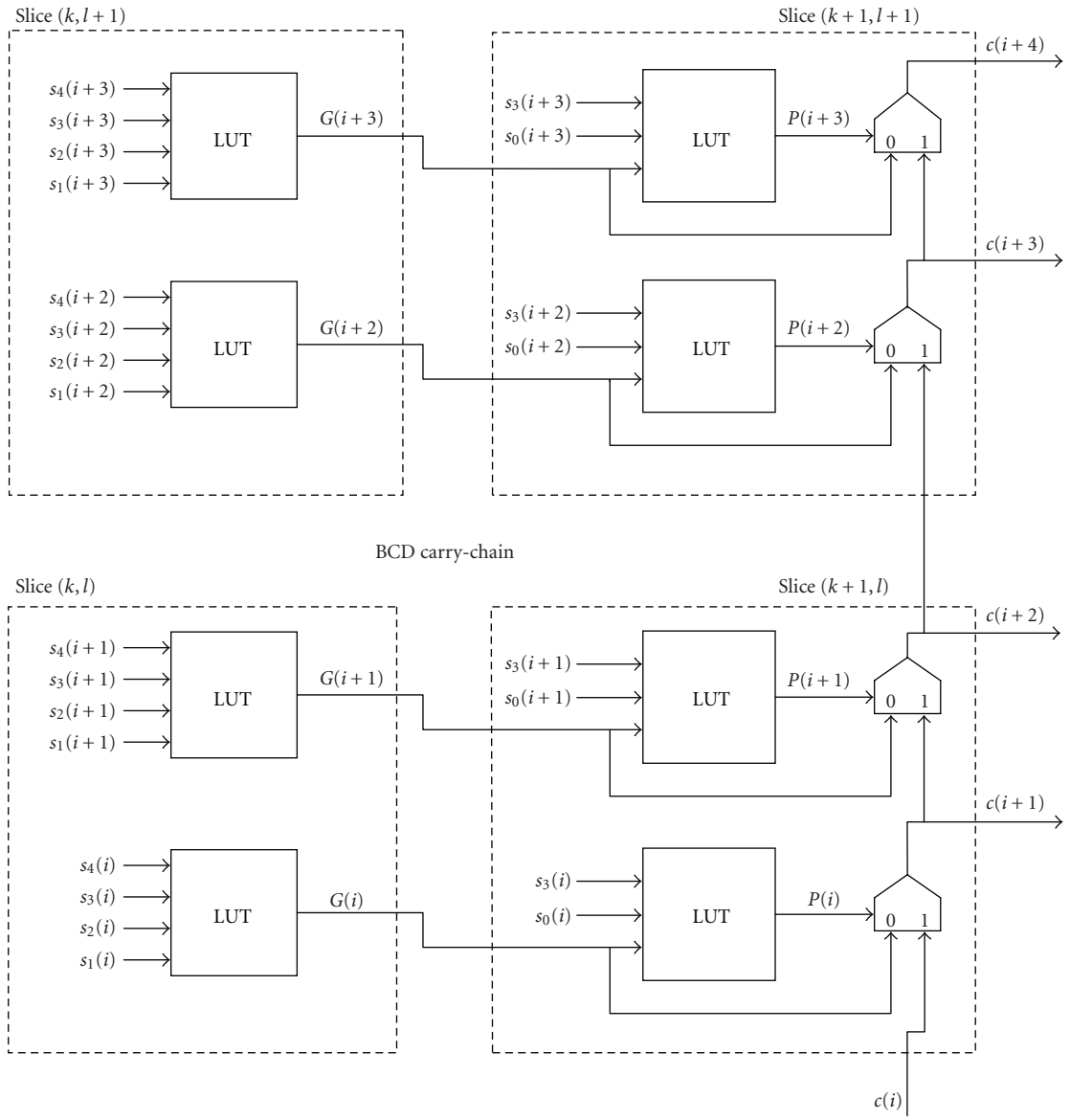


Figure 9: *FPGA* implementation of the carry-chain circuit (*GP_a* architecture).

$y(i)$ inputs using the Boolean expression (18). Using 4-input LUTs (4-LUTs), a first implementation (Figure 11) computes

$$
\begin{aligned}
c &= (x_2 \oplus y_2) \cdot (x_3' \cdot y_3'),\\
b &= (x_2' \cdot y_2') \cdot (x_3 \oplus y_3) \vee (x_2 \cdot y_2) \cdot (x_3' \cdot y_3'),\\
a &= (x_1' \cdot y_1') \cdot b \vee (x_1 \cdot y_1) \cdot c, \quad \text{prop} = (x_0 \oplus y_0) \cdot a,\\
h &= (x_0 \cdot y_0) \cdot (x_1 \cdot y_1), \qquad g = (x_0 \cdot y_0) \vee x_1 \vee y_1,\\
e &= (x_2 \cdot y_2) \cdot g \vee h \cdot (x_2 \oplus y_2),\\
f &= (x_1 \oplus y_1) \vee (x_2 \oplus y_2),\\
d &= (x_0 \cdot y_0) \vee f, \qquad \text{gen} = (x_3 \cdot y_3) \vee (x_3 \oplus y_3) \cdot d \vee e.
\end{aligned}
\tag{28}
$$

This architecture called *GP_b* is shown in Figure 11. The corresponding time and area of a carry-chain cell using this architecture is

$$
\begin{aligned}
T_{\text{Cy-Ch-}b} &= 4 \cdot T_{\text{LUT}} + 3 \cdot T_{\text{con1}} + T_{\text{mux-cy}},\\
C_{\text{Cy-Ch-}b} &= 10\,\text{LUTs}.
\end{aligned}
\tag{29}
$$

The complete cell includes a 4-bit adder and a conditional 3-bit output adder adding 6 whenever necessary (similar to Figure 5). The overall time delay and area consumption using this carry-computation cell is:

$$
\begin{aligned}
T_{N\text{-digit adder-}b} &= 4 \cdot T_{\text{LUT}} + (N+3) \cdot T_{\text{mux-cy}} + T_{\text{XOR}}\\
&\quad +3 \cdot T_{\text{con1}} + T_{\text{con2}},
\end{aligned}
\tag{30}
$$

$$
C_{N\text{-digit adder-}b} = 18 \cdot N\,\text{LUTs}. \tag{31}
$$

The results in area and speed are poor compared to the *GP_a* implementation (obtaining *G-P* from the results of the 4-bit adder).

Another alternative is based on the use of dedicated multiplexers. Xilinx Spartan 3, Virtex-2, and Virtex-4 devices have Look-Up Table multiplexers (muxf5, muxf6, muxf7, muxf8) in order to construct functions of 5, 6, 7, and 8 variables without using the general purpose routing fabric.

Using this feature the circuit of Figure 12 (*GP_c*) can be implemented using the following relations:

$$\begin{gathered} b = y_3' \cdot y_2' \cdot y_1'; \qquad d = x_2' \cdot x_1'; \\ c = y_3 \cdot d \vee y_3' \cdot e; \qquad a = x_3 \cdot b \vee x_3' \cdot c; \\ p1 = (x_0' \vee y_0') \cdot a, \\ e = (x_2 \cdot x_1 \cdot y_2' \cdot y_1) \vee (x_2' \cdot x_1 \cdot y_2 \cdot y_1) \vee (x_2 \cdot x_1' \cdot y_2 \cdot y_1'), \\ k = (x_2 \cdot y_2) \vee (x_1 \cdot y_1) \cdot (x_2 \vee y_2); \qquad h = j = 1; \\ i = y_3 \cdot j \vee y_3' \cdot k; \qquad g1 = x_3 \cdot h \vee x_3' \cdot i; \\ p2 = (x_0 \vee y_0). \end{gathered} \tag{32}$$

The corresponding time and area of a carry-chain cell *GP_c* is

$$\begin{gathered} T_{\text{Cy-Ch-}c} = T_{\text{6-LUT}} + T_{\text{LUT}} + T_{\text{con1}} + T_{\text{mux-cy}}, \\ C_{\text{Cy-Ch-}c} = 8\,\text{LUTs}, \end{gathered} \tag{33}$$

where $T_{\text{6-LUT}}$ stands for the delay from an LUT input to a muxf6 output. The complete cell also includes 4-bit adder and a conditional 3-bit adder. The overall delay-area for *GP_c* cell is

$$\begin{aligned} T_{N\text{-digit adder-}c} &= T_{\text{6-LUT}} + T_{\text{LUT}} + (2 \cdot N + 3) \cdot T_{\text{mux-cy}} \\ &\quad + T_{\text{XOR}} + T_{\text{con1}} + T_{\text{con2}}, \end{aligned} \tag{34}$$

$$C_{N\text{-digit adder-}c} = 16 \cdot N\,\text{LUTs}. \tag{35}$$

7. FPGA Implementations of Base-10 Adders and Adders-Subtractors on 6-Input LUTs Xilinx Platforms

7.1. Base-10 BCD Carry-Chain Adder. In a first version, *Ad*-I, the adding stage and correction stage are implemented as shown at Figures 8(a) and 8(b), respectively, while the carry-chain structure with the *G-P* functions is computed according to Figure 6.

Xilinx Virtex-5 6-input/2-output LUT is built as two 5-input functions while the sixth input controls a 2-1 multiplexor allowing to implement either two 5-input functions or a single 6-input one; so *G* and *P* functions fit in a single LUT as shown at Figure 13.

In a second version, *Ad*-II, the carry-chain is speeded up thanks to a direct computation of the *G-P*, namely, using inputs $x(i)$ and $y(i)$, instead of the intermediate sum bits s_k. For this purpose one could use formulas (18); nevertheless, in order to minimize time and hardware consumption the implementation of $P(i)$ and $G(i)$ is revisited as follows. Remembering that $P(i) = 1$ whenever the arithmetic sum $x(i) + y(i) = 9$, one defines a 6-input function $pp(i)$ set to be 1 whenever the arithmetic sum of the first 3 bits of $x(i)$ and $y(i)$ is 4. Then $P(i)$ may be computed as

$$P(i) = (x_0(i) \oplus y_0(i)) \cdot pp(i). \tag{36}$$

On the other hand, $gg(i)$ is defined as a 6-input function set to be 1 whenever the arithmetic sum of the first 3 bits of $x(i)$ and $y(i)$ is 5 or more. So, remembering that $G(i) = 1$ whenever the arithmetic sum $x(i) + y(i) > 9$, $G(i)$ may be computed as

$$G(i) = gg(i) \vee (pp(i) \cdot x_0(i) \cdot y_0(i)). \tag{37}$$

As Xilinx Virtex-5 LUTs may compute 6-variable functions, then $gg(i)$ and $pp(i)$ may be synthesized using 2 LUTs in parallel while $G(i)$ and $P(i)$ are computed through an additional single LUT as shown at Figure 14.

7.2. 10*'s Complement BCD Carry-Chain Adder-Subtractor.* To compute $X + Y$ similar algorithm as in Section 7.1 is used. In order to compute X–Y, 10's complement subtraction algorithm actually adds $(-Y)$ to X.

7.2.1. 10*'s Complement (AS-I).* 10's complement sign change algorithm may be implemented through a digitwise 9's complement stage followed by an add-1 operation. It can be shown that the 9's complement binary components z_3, z_2, z_1, and z_0 of a given BCD digit y_3, y_2, y_1, and y_0 are expressed as

$$z_3 = y_3' \cdot y_2' \cdot y_1';\ z_2 = y_2 \oplus y_1;\ z_1 = y_1;\ z_0 = y_0'. \tag{38}$$

To compute X–Y, 10's complement subtraction algorithm actually adds $(-Y)$ to X. So for a first implementation, *AS*-I, Figure 15 presents a 9's complement implementation using 6-input/2-output LUTs, available in the Virtex-5 Xilinx technology. A'/S is the add/subtract control signal; if $A'/S = 1$ (subtract), formulas in (38) apply; otherwise $A'/S = 0$ and $z_j(i) = y_j(i)$ for all i, j.

The *AS*-I circuit is similar to the *Ad*-I (Figures 8 and 13) using, instead of input Y, the input Z as produced by the circuit of Figure 15.

7.2.2. Improving the Adder Stage. To avoid the delay produced by the 9's complement step, this operation may be carried out within the first binary adder stage, as depicted in Figure 16, where $p(i)$ and $g(i)$ are computed as

$$\begin{aligned} p_0(i) &= x_0(i) \oplus y_0(i) \oplus (A'/S), \\ p_1(i) &= x_1(i) \oplus y_1(i), \end{aligned}$$

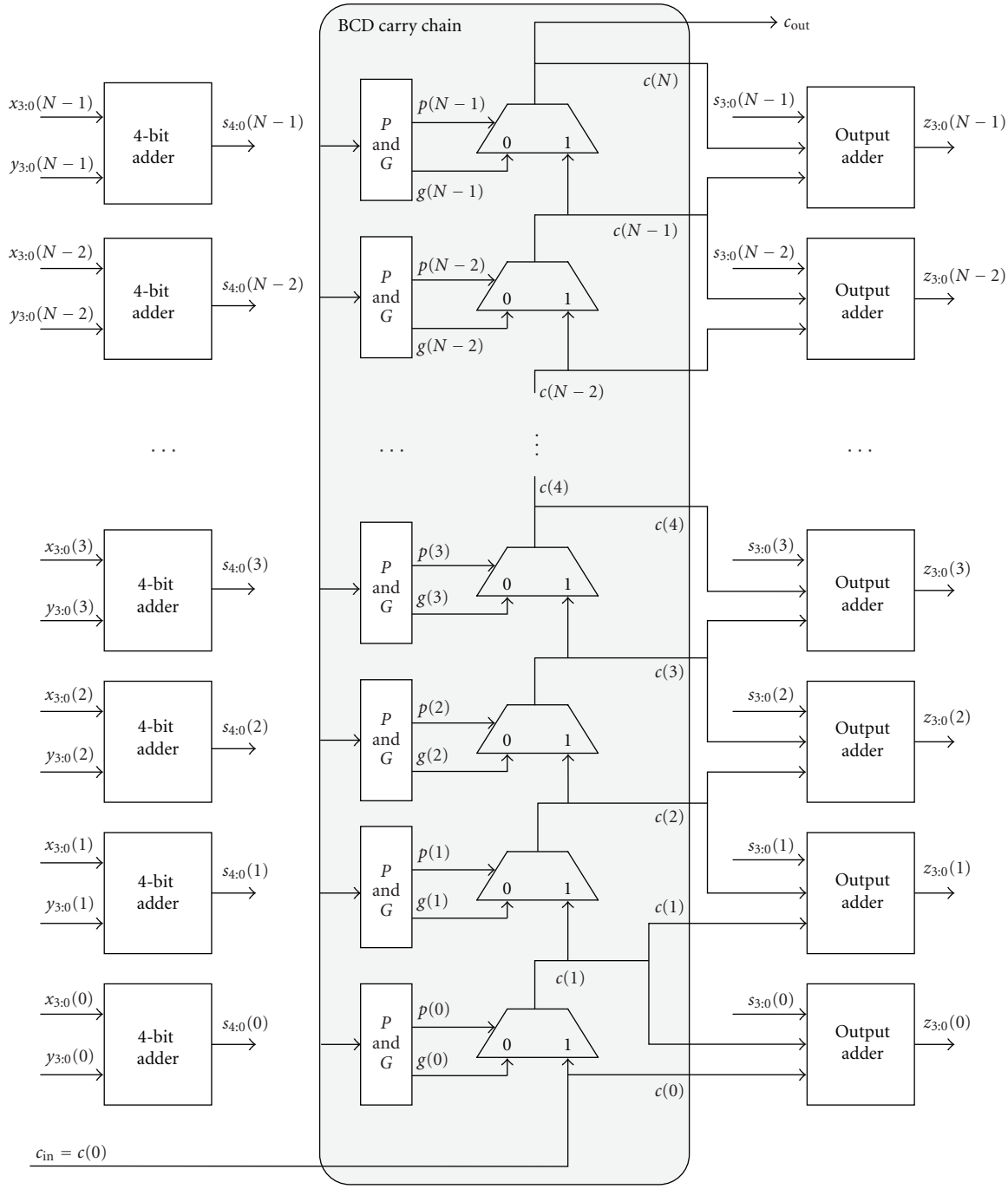


Figure 10: *FPGA* implementation of an N-digit BCD Adder.

$$
\begin{aligned}
p_2(i) &= x_2(i) \oplus y_2(i) \oplus y_1(i) \cdot (A'/S),\\
p_3(i) &= x_3(i) \oplus \left(y_3(i)' \cdot y_2(i)' \cdot y_1(i)'\right) \cdot (A'/S)\\
&\quad \oplus y_3(i) \cdot (A'/S)',\\
g_k(i) &= x_k(i), \quad \forall k.
\end{aligned} \tag{39}
$$

7.2.3. Carry-Chain Stage Computing G and P Directly from the Input Data (AS-II). As far as addition is concerned, the P and G functions may be implemented according to formulas (36) and (37). The idea of the *AS*-II is computing the corresponding functions in the subtract mode and then multiplexing according to the add/subtract control signal A'/S. For this reason, assuming that the operation at hand is $X + (\pm Y)$, one defines on one hand $ppa(i)$ and $gga(i)$

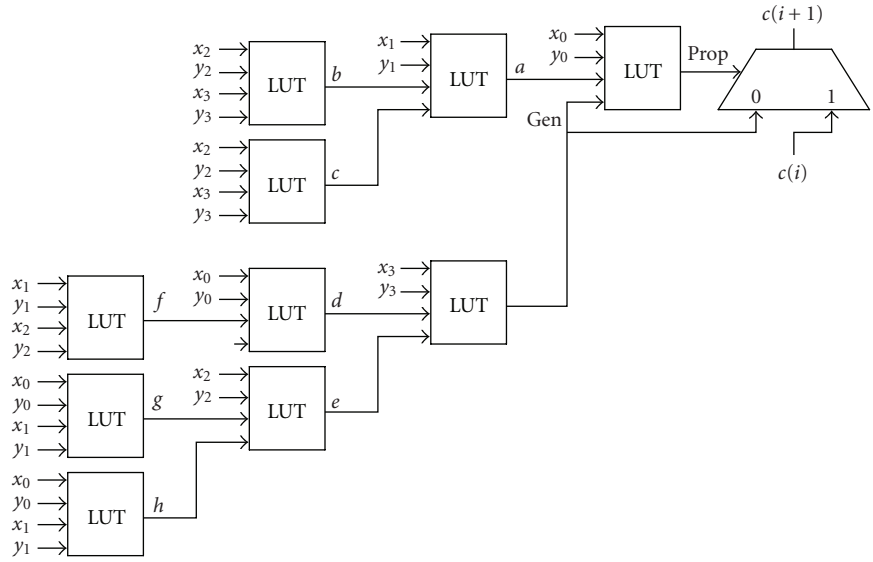


Figure 11: Carry generation (*GP_b* architecture).

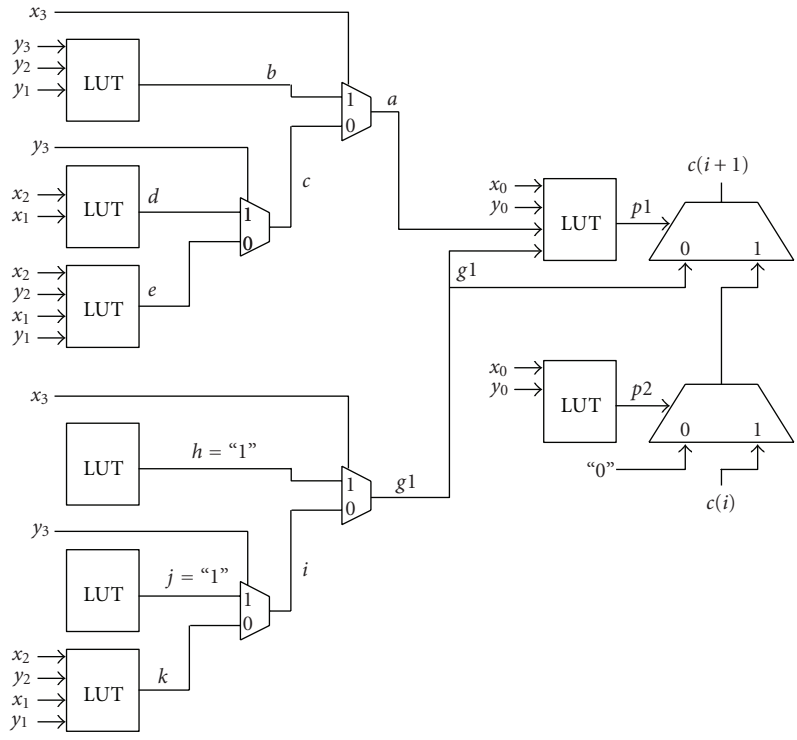


Figure 12: Carry generation (*GP_c* architecture).

according to Section 7.1, that is, using the straight values of Y's BCD components.

On the other hand, $pps(i)$ and $ggs(i)$ are defined according to the same Section 7.1 but using $z_k(i)$ as computed by the 9's complement circuit shown at Figure 15. As $z_k(i)$ are expressed from the $y_k(i)$ (38), both $pps(i)$ and $ggs(i)$ may be computed directly from $x_k(i)$ and $y_k(i)$ as shown in Figure 17. Nevertheless, for subtraction, the computation of $z_0(k) = y_0'(k)$ is carried out at the output LUT level. So formulas (36) and (37) are then expressed as

$$\begin{aligned} P(i) &= \left(x_0(i) \oplus y_0(i) \oplus (A'/S)\right) \cdot pp(i), \\ G(i) &= gg(i) \vee \left(pp(i) \cdot x_0(i) \cdot \left(y_0(i) \oplus (A'/S)\right)\right). \end{aligned} \tag{40}$$

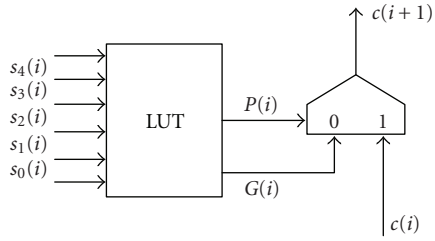


Figure 13: *FPGA* carry-chain circuit for *Ad*-I.

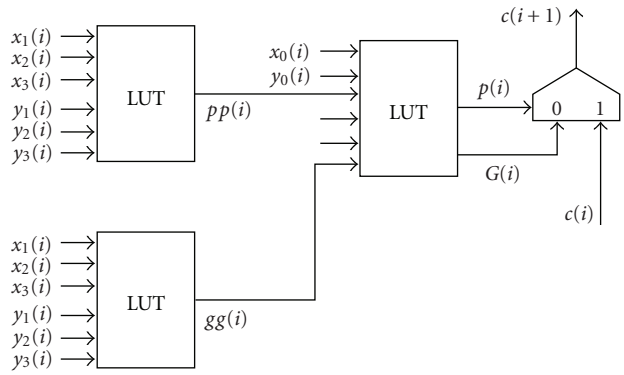


Figure 14: *FPGA* carry-chain circuit for *Ad*-II.

8. Experimental Results

8.1. Xilinx Virtex-4 Adder Implementations. The base-10 adders have been implemented on Xilinx Virtex-4 *FPGA* family speed grade-11 [6]. The Synthesis and implementation have been carried out on *XST* (Xilinx Synthesis Technology) [13] and Xilinx *ISE* (Integrated System environment) version 10.1 [14].

Performances of different N-digit BCD adders have been compared to those of an M-bit binary carry chain adder (implemented by *XST* [13] using Xilinx fast carry logic) covering the same range, that is, as

$$M = \left\lceil N \cdot \log_2(10) \right\rceil \cong 3.322 \times N. \tag{41}$$

The time and hardware complexities of an M-bit ripple-carry adder implemented on the same 4-LUT based Xilinx *FPGA* are given by

$$\begin{aligned} T_{M\text{-bit adder}} &= T_{\text{LUT}} + M \cdot T_{\text{mux-cy}}, \\ &= T_{\text{LUT}} + 3.322 \times N \times T_{\text{mux-cy}}. \end{aligned} \tag{42}$$

$$C_{M\text{-bit adder}} = M \text{ LUTs} = 3.322 \times N \text{ LUTs} \tag{43}$$

Formulas (26), (30), (34), and (42) show that, asymptotically, $T_{N\text{-digit adder}}$ should be somewhat inferior to $T_{M\text{-bit adder}}$. Nevertheless, as shown by the experimental results, the additive values appearing in (26), (30), and (34) are not negligible for reasonable values of N; so the saving in time will mainly appear for applications where BCD-to-binary coding and decoding operations play a significant role in the overall delay.

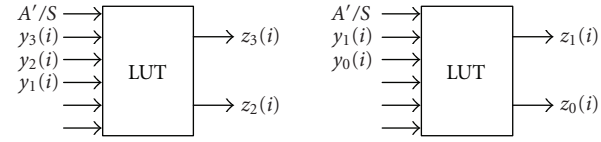


Figure 15: *FPGA* 9's complement circuit for *AS*-I.

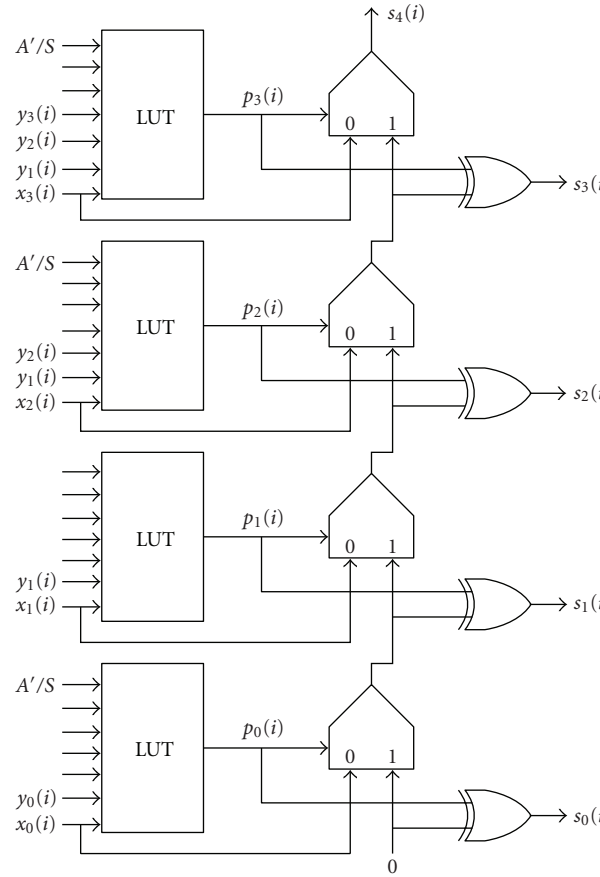


Figure 16: *FPGA* implementation of the adder stage for a 10's complement BCD adder-subtractor.

Post place-and-route time delays and area consumptions are quoted in Tables 1 and 2, respectively, where N stands for the number of BCD digits while M stands for the number of bits required to cover the decimal N-digit range. The results presented in the table are as follows:

(i) *Ripple:* Naïve implementation of base-10 ripple-carry (Section 5.1, Figures 1 and 5),

(ii) *PG_a:* base-10 carry chain using an adder to produce *P-G* values (Section 5.2, Figures 2, 6, 8, 9, and 10),

(iii) *PG_b:* base-10 carry chain computing directly *P-G* values (Section 6.3, Figures 2 and 11),

(iv) *PG_c:* base-10 carry chain computing directly *P-G* values using muxf5 and muxf6 (Section 6.3, Figures 2 and 12),

(v) *M-bit Binary:* base-2 carry chain adder covering the same range as an N-digit adder.

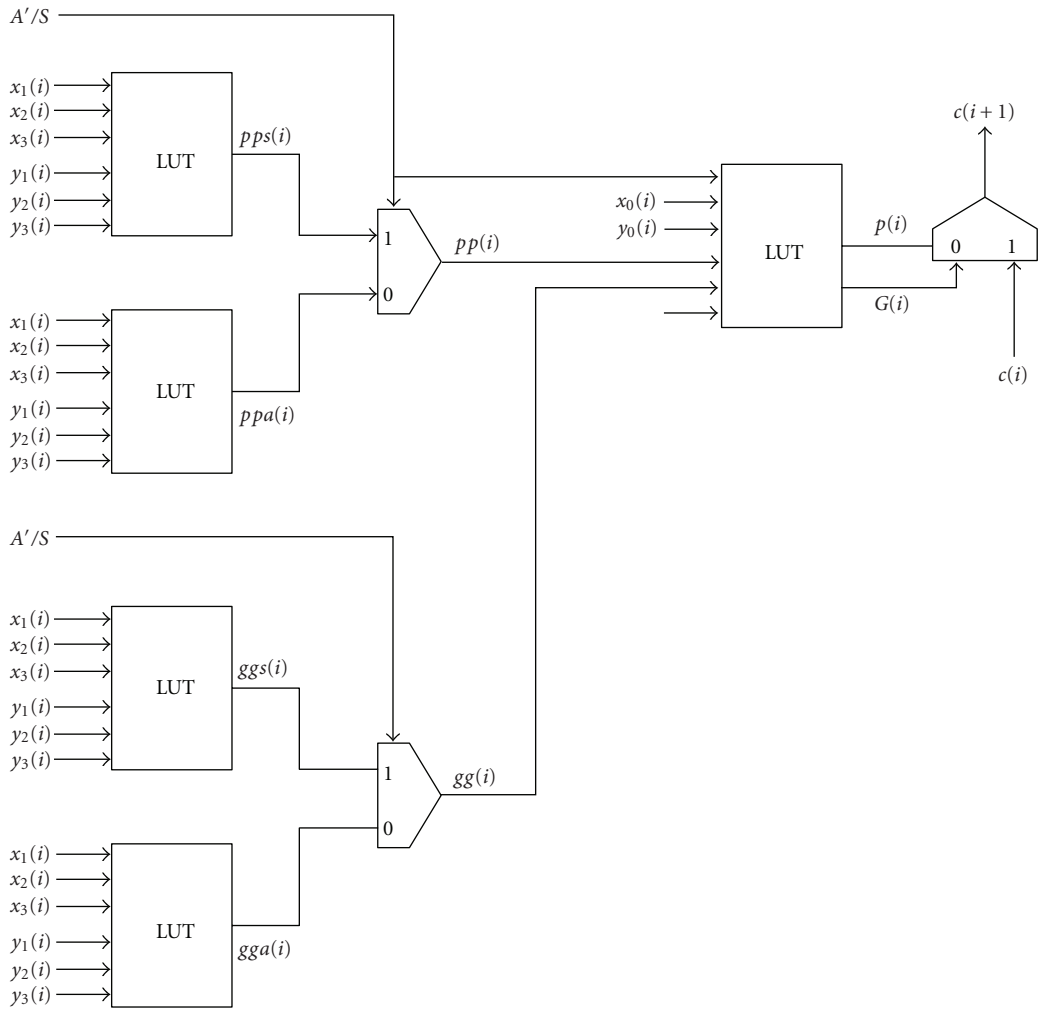


Figure 17: *FPGA* implementation of the carry-chain stage *AS*-II for BCD adder-subtractor.

Figure 18 shows the delays for the compared adders. Observe that, for the technology at hand, Table 1 and Figure 18 suggest that for $N > 48$ the carry-chain decimal implementation of adders is faster than the binary one for the equivalent range. Furthermore for small numbers of digits to add ($N < 40$) the *PG_c* architecture is faster than other decimal implementations.

8.2. Virtex-5 Adder-Subtractor Implementations. The adder-subtractor circuits have been implemented on Xilinx Virtex-5 family with speed grade-2 [7]. The synthesis and implementation have been carried out on *XST* (Xilinx Synthesis Technology) [13] and Xilinx *ISE* (Integrated System environment) version 10.1 [14]. The critical parts were designed using low-level components instantiation (lut6_2, muxcy, xorcy, etc.) in order to obtain the desired behavior. Performances of different N-digit BCD adders have been compared to those of an M-bit binary carry chain adder (implemented by *XST*) covering the same range, that is, such that $M = \lceil N \cdot \log_2(10) \rceil \cong 3.322\,\mathrm{N}$.

Table 1: Time delays (nsec) for different adders in Virtex-4 -11.

N	N-digit BCD adder				M	M-bit Binary
	Ripple	*PG_a*	*PG_b*	*PG_c*		
8	14	5.8	6.7	4.5	27	2.7
12	20	6.0	6.8	4.8	40	3.2
16	27	6.1	7.0	5.1	54	3.7
24	40	6.4	7.3	5.7	80	4.7
32	53	6.7	7.6	6.3	107	5.7
40	66	7.0	7.9	6.9	133	6.6
48	79	7.3	8.2	7.5	160	7.6
64	105	7.9	8.7	8.7	213	9.6
96	—	9.1	9.9	11.0	319	13.5
128	—	10.3	11.1	13.4	426	17.5

Table 3 exhibits the postplacement and routing delays in *ns* for the decimal adder implementations *Ad*-I and *Ad*-II of Section 7.1; Table 4 exhibits the delays in *ns* for

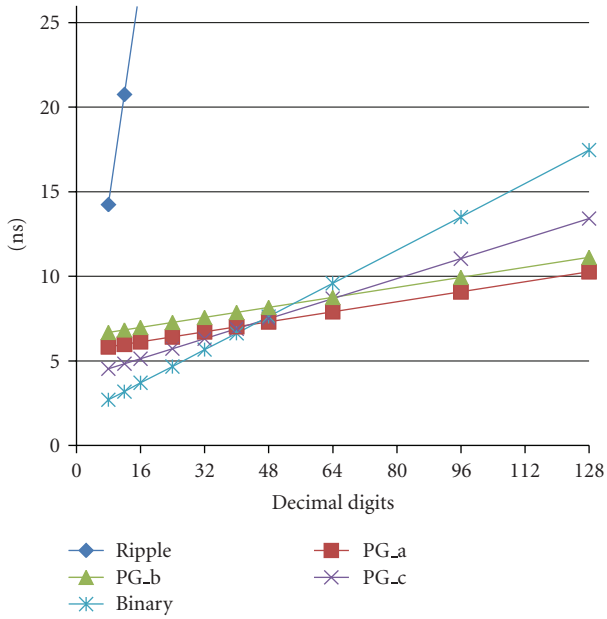


Figure 18: Delay in *ns* for different adders in Virtex-4.

Table 2: Area (in LUTs) for different adder's size in Virtex-4.

Adder Circuit	# LUT's
Ripple	$8 \times N$
PG_a	$10 \times N$
PG_b	$18 \times N$
PG_c	$16 \times N$
Binary	$\lceil 3.32 \times N \rceil$

Table 3: Delays in ns for decimal and binary adders in Virtex-5 -2.

N (BCD digits)	*Ad*-I	*Ad*-II	M (bits)	Binary Adder
8	3.8	3.6	27	2.1
16	4.4	4.0	54	2.6
32	5.0	4.5	107	3.8
48	5.2	5.1	160	5.1
64	5.6	5.2	213	6.6
96	6.1	5.9	319	8.7

the decimal adder-subtractor implementations *AS*-I and *AS*-II of Section 7.2. Table 5 lists the consumed areas expressed in terms of 6-input look-up tables (6-input LUTs). The estimated area presented in Table 5 was empirically confirmed.

Comments. Observe that, for large operands, the decimal operations are faster than the binary ones.

The overall area with respect to binary computation is not negligible. In Virtex-4 the area increases, with respect to an equal range binary adder, in a factor between 2.4 and 5.4. In the 6-input LUT family Virtex-5 an adder-subtractor is between 3.0 and 3.9 times bigger.

Table 4: Delays in ns for decimal and binary adder-subtractor in Virtex-5 -2.

N (BCD digits)	*AS*-I	*AS*-II	M (bits)	Binary Add-Sub
8	3.8	3.8	27	2.1
16	4.1	4.0	54	2.6
32	4.7	5.0	107	3.8
48	5.3	5.2	160	5.2
64	5.7	5.5	213	6.6
96	6.3	6.2	319	8.8

Table 5: Area in 6-input LUTs for different adders and adders-subtractors.

Circuit	# LUTs
Adder *Ad*-I	$8 \times N$
Adder *Ad*-II	$10 \times N$
Binary Adder	$\lceil 3.32 \times N \rceil$
Adder-Subtractor *AS*-I	$10 \times N$
Adder-Subtractor *AS*-II	$13 \times N$
Binary Adder-Subtractor	$\lceil 3.32 \times N \rceil$

9. Conclusions

The present interest in BCD arithmetic systems stimulates further researches at both the algorithmic and design levels. Considering that the hardware costs are everyday more affordable, full hardware BCD units are now very attractive, with moreover a growing potential in the near future.

This paper has developed some implementations of BCD adders and subtractors in *FPGA* platforms. Experimental results emphasize time performances with reasonable costs in terms of area. Matched with the binary system, the decimal implementations are faster as operand sizes are growing (break even around 50 digits).

One of the key points about delays comes from the fact that the carry-propagation computation remains binary; then a faster carry-chain circuit can be designed because, for the same operand range, the number of digits (therefore of carries to propagate) is lower in decimal than in binary. In the carry-chain structures studied in this paper, the propagate P and generate G functions are more complex and therefore more time and area consuming than in the binary ones; therefore, the speed improvements only appear for large enough operands. The breakeven point is obviously technology dependent; so it could be expected to occur for a smaller number of digits in the near future.

The area overhead with respect to binary computation is not negligible; it is around five times in Virtex-4 and nearly four times in Virtex-5. That is mainly due to the more complex definition of the *carry propagate* and *carry generate* functions and to the final mod 10 reduction. The decreasing costs of technology make hardware consumption less central.

For BCD addition, the performance considerations on Xilinx Virtex-5 platform are similar to those of 4-input LUTs-based Virtex-4 technology. That is, the addition time of BCD digits remains faster than the binary counterpart in the same conditions.

Finally, the BCD adder/subtractor, with a relatively small penalty in area, presents time performances quite similar to those of a straight BCD adder.

Acknowledgments

This work is supported by the Universities FASTA, Mar del Plata, Argentina, UNCPBA Tandil, Argentina, UAM, Madrid, Spain, and URV, Tarragona, Spain; it has been partially granted by the CICYT of Spain under contract TEC2007-68074-C02-02/MIC.

References

[1] M. F. Cowlishaw, "Decimal floating-point: algorism for computers," in *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 104–111, June 2003.

[2] G. Jaberipur and A. Kaivani, "Binary-coded decimal digit multipliers," *IET Computers and Digital Techniques*, vol. 1, no. 4, pp. 377–381, 2007.

[3] M. Cowlishaw, "General Decimal Arithmetic," http://speleotrove.com/decimal/.

[4] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, "The IBM Z900 decimal arithmetic unit," in *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1335–1339, November 2001.

[5] *IEEE Standard for Floating-Point Arithmetic (IEEE 754)*, IEEE, 2008.

[6] Xilinx Inc., "Virtex-4 User Guide," April 2007, http://www.xilinx.com/.

[7] Xilinx Inc., "Virtex-5 User Guide," 2008, http://www.xilinx.com/.

[8] J.-P. Deschamps, G. Bioul, and G. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*, John Wiley & Sons, New York, NY, USA, 2006.

[9] B. Parhami, *Computer Aritmethic: Algorithms and Hardware Designs*, Oxford University Press, Oxford, UK, 2000.

[10] Xilinx Inc., Xilinx, http://www.xilinx.com/.

[11] "Decimal Arithmetic in FPGA," http://arithmetic-circuits.org/decimal/.

[12] Xilinx Inc., *Constraints Guide—ISE9.2i*, chapter 2, Relative Location (RLOC), 2008.

[13] Xilinx Inc., "XST User Guide-10.1i," 2008, http://www.xilinx.com/.

[14] Xilinx Inc., "ISE 10.1 Documentation," 2008, http://www.xilinx.com/.

Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 962057, 8 pages
doi:10.1155/2010/962057

Research Article

Concurrent Calculations on Reconfigurable Logic Devices Applied to the Analysis of Video Images

Sergio R. Geninatti,[1] José Ignacio Benavides Benítez,[2] Manuel Hernández Calviño,[3] and Nicolás Guil Mata[4]

[1] *Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Universidad de Rosario, Avenue Pellegrini 250, 2000 Rosario, Argentina*
[2] *Escuela Politécnica Superior, Universidad de Córdoba, Menéndez Pidal s/n, 14001 Córdoba, Spain*
[3] *Departamento de Física General, Facultad de Física, Universidad de La Habana, Colina Universitaria El Vedado, 10300 La Habana, Cuba*
[4] *Departamento de Arquitectura de Computadoras, Universidad de Málaga, C. Teatinos, 29071 Málaga, Spain*

Correspondence should be addressed to Sergio R. Geninatti, foco@fceia.unr.edu.ar

Received 25 May 2009; Accepted 11 November 2009

Academic Editor: Elías Todorovich

This paper presents the design and implementation on FPGA devices of an algorithm for computing similarities between neighboring frames in a video sequence using luminance information. By taking advantage of the well-known flexibility of Reconfigurable Logic Devices, we have designed a hardware implementation of the algorithm used in video segmentation and indexing. The experimental results show the tradeoff between concurrent sequential resources and the functional blocks needed to achieve maximum operational speed while achieving minimum silicon area usage. To evaluate system efficiency, we compare the performance of the hardware solution to that of calculations done via software using general-purpose processors with and without an SIMD instruction set.

1. Introduction

The capacity of Reconfigurable Logic for massive concurrent computation makes it wellsuited to implementing complex algorithms. This feature has increased the potential of digital design, as shown by several papers which have proposed implementing mathematical algorithms implemented on FPGA. In almost all cases the topology and the performance obtained depend on the specific application, such that each case must be analyzed individually. References [1–3] show the design of custom image processing systems on FPGA.

This paper presents the advantages of concurrency and parallelism in implementing video temporal segmentation with Reconfigurable Logic Devices, emphasizing the advantages and limitations of FPGA technology and its development tools. We chose to implement a function that has been thoroughly studied in [4] and that measures the similarity between two frames based on their luminance distributions. After breaking the algorithm down into its component parts, we propose specific solutions for each part, while making it clear that obtaining the optimum hardware solution is an iterative process which benefits from all the features of FPGA devices [5].

In Section 2, we briefly present the design philosophy of dedicated hardware, where both Configurable Logic Blocks, (CLBs in XILINX devices), and special-function blocks have been used. In addition, we describe an interface with external memory, an essential feature when dealing with huge volumes of data.

Section 3 offers an introduction to video temporal segmentation through the artificial synthesis of visual human perception and presents the similarity-calculation algorithm on which this paper is based.

Section 4 describes in detail the implementation and optimization of the proposed algorithm, emphasizing the design and evaluation criteria.

Finally, the experimental results section shows the quantitative results obtained by comparing computing time using the proposed hardware solution to that of pure software running on a PC.

2. Design Criteria

The combination of CLB logic with embedded Blocks and data supply are critical aspects of the design to be taken into account.

2.1. CLB Logic versus Embedded Blocks. The main drawback of using a reconfigurable hardware device is the loss of performance imposed by the reconfiguration circuit itself [6]. For this reason, device manufacturers often include embedded Blocks able to perform frequent and specialized tasks, including RAM blocks, multipliers, timers, and communication devices, whose performance is far superior to that of equivalent functions implemented using CLB logic.

Another important issue is the time delay introduced by routing. This factor justifies the inclusion on the FPGA of different-quality routes for different purposes (clocks, near, long, etc.).

The present study will show the many possible alternatives for combining all these elements in a specific design. Those selected in each case will depend on the specific implementation being conducted.

2.2. Data Supply. Video processing is a task characterized by a very high demand for data, and when using an FPGA there is only a limited storage capacity available on chip. Thus, an efficient interface is required with external memory capable of feeding and receiving data at the necessary rate. In this case, we used a Xilinx Spartan-3 development board provided with a static external memory bank having 32-bit width and 10-ns access time. This bus width allows us to read and process four bytes at a time. The reading and modification of data storage into the Spartan BlockRAM requires two clock cycles per operation. This memory is synchronous and can operate up to a 200 MHz clock frequency, perfectly matching the 10-ns access time of the external memory.

3. Application: Temporal Video Segmentation

Our aim is the development of a specific architecture for video segmentation. This video analysis technique enables the detection of low-level semantic units in the video, which are called shots. Many applications are based on shot information to carry out higher level analysis, such as video classification, summarization, visual index generation, or sequence comparison [7]. The technique is based on giving a quantitative value to the human perception of similarity between frames. In this theory, it is assumed that the properties of a certain stimulus—in this case an image—can be represented as a vector in a space of characteristics and, as a consequence, the similarity between two images is reduced to the appropriate measurement of a distance using a metric on a psychological space [4].

One of the main features of an image is its luminance histogram, defined as the frequency of occurrence of the pixels' luminance values in each frame. The similarity between two frames is inversely related to the distance between vectors representing its characteristics. In the case of a histogram of luminance, this can be defined through the following normalized equation [4]:

$$f_s(i-1,i) = \frac{\sum_b H_{i-1}[b] \cdot W_i[b]}{\sqrt{\sum_b H_{i-1}[b] \cdot W_{i-1}[b]} \cdot \sqrt{\sum_b H_i[b] \cdot W_i[b]}}, \tag{1}$$

where $H_i[b]$ is the histogram of the bin "b" from frame "i", $\mathrm{W_i}[b]$ is the windowed histogram at level "b" from frame "i", Notice that "$i-1$" and "i" subindexes refer to consecutive frames.

4. Algorithm Segmentation

In order to implement and optimize expression (1), we divide the procedure into the following five sequential stages:

(1) calculation of the histogram,

(2) 1D windowing using a window size of three to calculate $W_i[b]$,

(3) sum of products,

(4) square root,

(5) division.

Multipliers needed for stage 3 were not implemented in VHDL because the Spartan 3 device has dedicated 18-bit multiplier blocks providing much better performance than that attainable using CLB logic.

4.1. Calculating the Histogram. The real bottleneck occurs at the stage responsible for obtaining the histogram, and its interface with external memory is the limiting factor for the overall circuit performance. Therefore, it is advisable to try to reduce the amount of data to be processed. As shown in [8], the use of DC coefficients of compressed frames instead of frame pixels values has no influence on the reliability of the video segmentation technique. Thus, we have reduced the number of data to be read from each video frame by using these DC coefficients. This allows us to achieve a reduction ratio of 64 to 1.

Considering the results obtained in [9], which proved that a circuit for obtaining the histogram using BlockRAMs as accumulators performs better than those using CLB logic, we propose the hardware shown in Figure 1 for the first stage. Note that the dual port feature of the Spartan-3 BlockRAM allows us to extract the histogram's individual values in pairs.

Port B input has been forced to "0" to clear the accumulators in the same clock cycle, leaving the hardware ready for processing the following frame. Several registers were inserted between output adders and at the address input of the BlockRAMs to pipeline the stage, so as to optimize the interface with external memory and double the internal clock frequency.

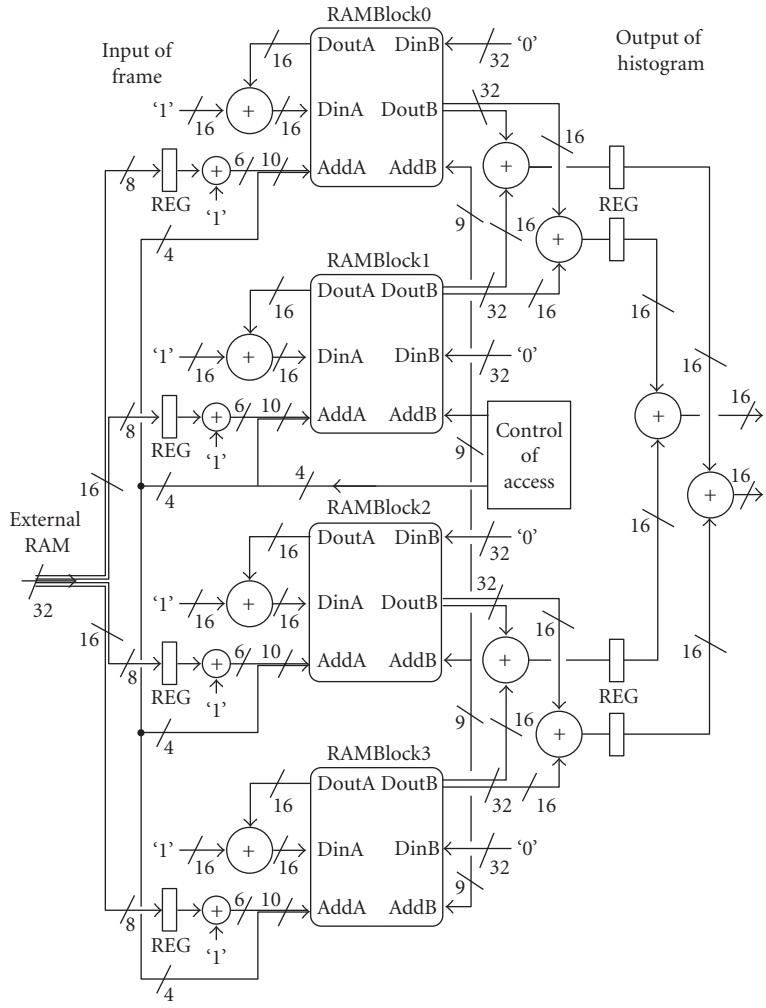


Figure 1: Circuit for calculating the histogram.

4.2. Calculating the Windowed Terms $W_i[b]$. To reduce the influence of slight variations in image luminance, each term of the histogram has been replaced with the sum of its adjacent neighbors. Figure 2 shows how the E_i windowed values are generated, as well as the convenience of using concurrent calculations due to the fact that the majority of H_i values are used in more than one E_i calculation. Excluding the two boundary values, each H_i value contributes to E_{i-1}, E_i, and E_{i+1}.

Figure 3 shows how the algorithm symmetry allows the same sum term to be used twice in calculating the windowed value. This stage was designed to calculate six windowed values concurrently, starting from eight histogram values at the input. The concurrent supply of these eight values is achieved by using two-storage BlockRAMs. As shown below, these two memories are also used for obtaining the sum terms needed for the correlation. These two blocks have four 32-bit-width output ports each, and thus are capable of supplying the eight necessary histogram values simultaneously.

Figure 4 shows the first version of this stage. The folded buses that appear at Port B inputs of both blocks are used to repeat some histogram values in particular positions of the memory bank, thus solving the boundary problem mentioned in the above calculation. The output summing stages were also pipelined to keep the maximum clock frequency of the whole stage close to 200 MHz, while the three output buses enable the concurrent extraction of six windowed values in each clock cycle.

4.3. Coupling Considerations. To prevent the delay introduced by the next calculation from piling up, BlockRAMs 4 and 5 retain data belonging to a frame while the following one is being processed in a different memory page. The BlockRAM dual port feature allows for decoupling and behaves like another pipeline stage. The only difference in this case is that the retention unit is the whole frame and not a single clock cycle.

Bearing in mind that 64 is the number of established histogram levels, each one represented by a 16-bit integer, the proposed structure can process frames containing up to 65,536 pixels. Taking as a reference a frame of only 1600 pixels, Table 1 shows the calculation time expressed as the number of elapsed clock cycles.

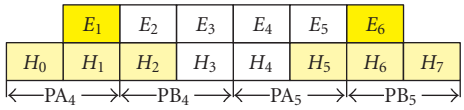


Figure 2: Concurrent windowing definition.

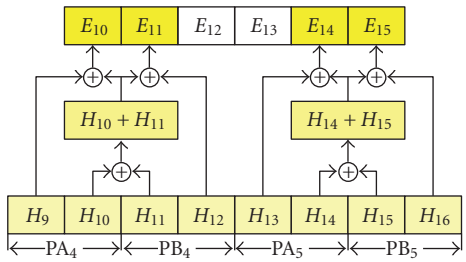


Figure 3: Grouping the windowed sum terms.

Table 1: Number of clock cycles per operation.

Histogram calculation	800
Intermediate storage	32
Windowing	11

At the same time that storing and windowing are being performed on the previous histogram, the input stage builds the following histogram. After the first frame is complete, the remaining calculations are done in parallel within the 800 clock cycles taken by the first stage.

4.4. Calculating the Correlation and the Sum. To implement this stage, we have modified the circuit depicted in Figure 4 to calculate two correlation sum terms simultaneously. We have placed six multipliers next to the windowing stage followed by a matrix of a three-level adder, responsible for obtaining the correlation. The whole block has been pipelined using six registered stages, keeping the clock frequency close to 140 MHz.

Two of the partial sums needed to calculate the similarity coefficient as defined by (1) are stored in the output. Partial terms of the denominator will be reused in the following frame calculation; hence, they are stored to avoid repeating the calculation (see Figure 5). Note that both summing terms use the windowed term of the same frame $W_i[b]$, but while one is multiplied by the current histogram $H_i[b]$, the other is multiplied by the previous one $H_{i-1}[b]$. The term $W_i[b]$ is retained during two clock cycles in the registers (shadowed in Figure 5) placed at the input of the multipliers, ensuring that in each clock cycle the correct histogram corresponds to its associated sum term.

Starting from the histograms stored in BlockRAM belonging to two frames, the stage only takes 28 clock cycles to complete two windowed sums. The key to such high performance lies in the concurrency of operations, the organization of the data, and the pipeline structure. It is also easy to interface this block with its neighbors because the input and output buses are only 32- and 64-bits wide, respectively. In contrast, the internal bus-width parallelism reaches 192 bits in the multiplier layer.

4.5. Square Root Calculation. The purpose of this stage is to obtain the square root of the denominator in (1). This calculation is performed by using shift and sum operations according to the algorithm described in [10]. The circuit depicted in Figure 6 shows the hardware implementation of this operation, starting from a slight modification at the output of the circuit shown in Figure 5.

It takes 16 clock cycles to complete processing a 32-bit-long input data. The sum term $S[h(n).w(n)]$ is available at the previous correlation block one clock cycle prior to completing the sum term $S[h(n-1)w(n)]$, a fact that can be useful in anticipating the beginning of the square root calculation. It should also be emphasized that some terms in the numerator of (1) belonging to histogram "n" will be used in the same calculation of the next frame. For this reason, a register has been provided to retain it.

4.6. Product and Division. Following the sequential stage responsible for calculating the square root, a multiplying block provides the product of the denominator in (1). In this way, numerator and denominator are ready and available at the output of the circuit shown in Figure 6 to make the final division and obtain the normalized similarity coefficient between two frames.

The division between two unsigned integers employing a shift and subtract algorithm is done by a circuit similar to the one shown in Figure 6, as described in [10]. The full process takes 32 clock cycles for the 32-bit solution required.

4.7. Control and Timing. The stages referred to above require a set of control and synchronization signals to properly manage data flow. Therefore, a control circuit is required to generate the appropriate timing signals. Figure 7 shows the diagram of the Finite State Machine responsible for controlling the whole system. This was implemented in VHDL description language and placed and mapped with the rest of the module at the top level of the hierarchy tree.

The design of this finite state machine is crucial, because the generated signals have to drive all the calculating modules, such that the fan-out and route length of these lines determine the maximum system clock frequency.

Because of the intensive use of pipeline architecture, the system exhibits a finite latency time from which a new value is output each 800 clock cycles (5.7 μs at 110 MHz clock frequency). Data flow has been organized in such a way that the first stage calculates the histograms on alternate RAM pages, taking 800 clock cycles to process each 1600-pixel frame. The rest of the calculation, which takes 109 clock cycles to complete, starts each time the processing of a frame is over. Except for the first stage (see Figure 1), the rest of the stages remain idle 86% of the time.

Clearly, the bottleneck is the width of the bus that connects the external memory with the above-described system. Widening this bus from 32 to 128 bits would make it possible to process 16 pixels at a time, reducing the time needed to

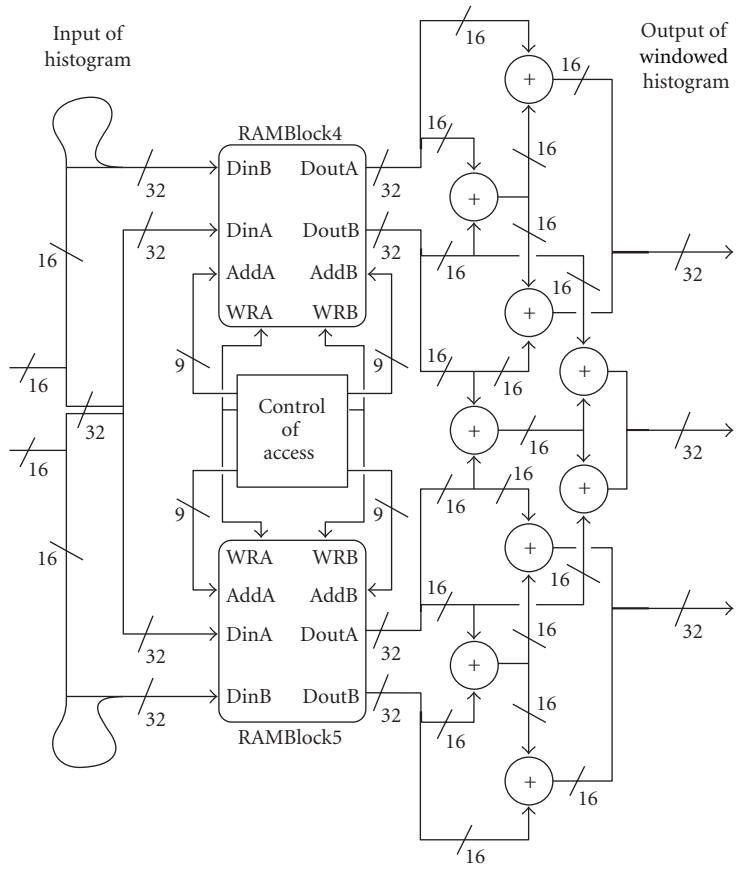


Figure 4: Circuit for calculating the windowed terms.

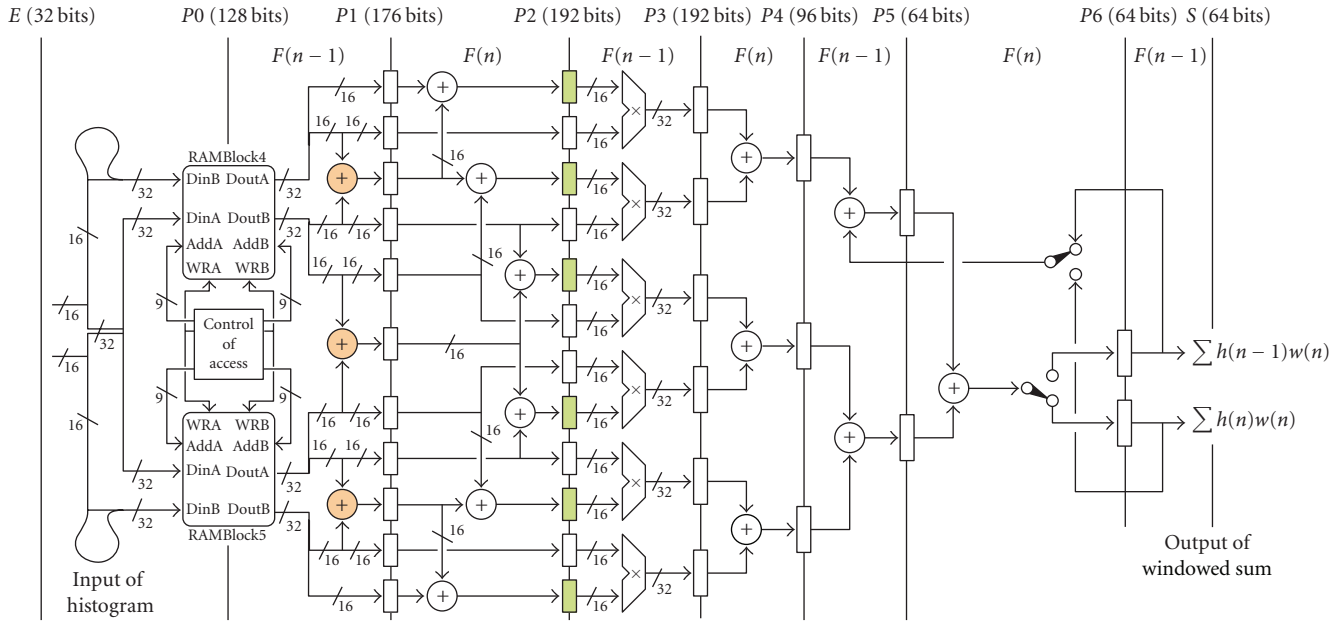


Figure 5: Circuit for calculating the windowed correlation (includes windowed calculation pipelined).

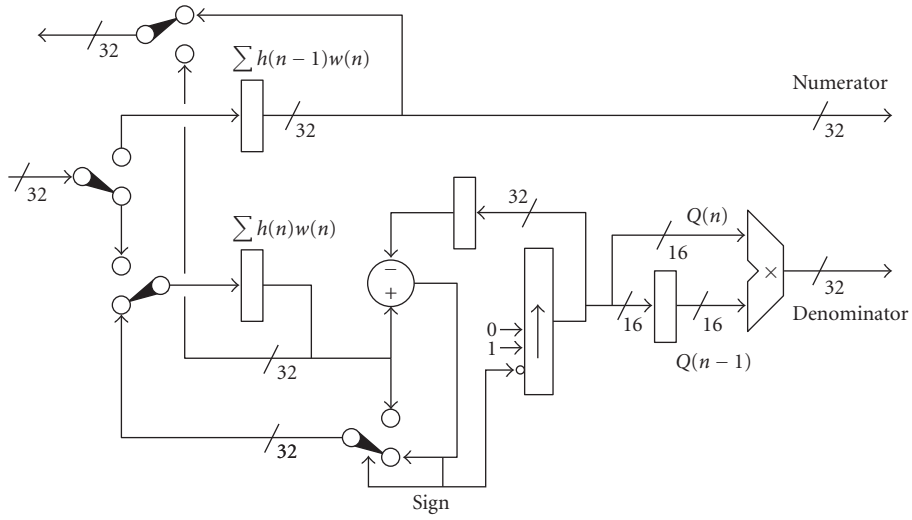


Figure 6: Circuit for obtaining the square root and terms of division.

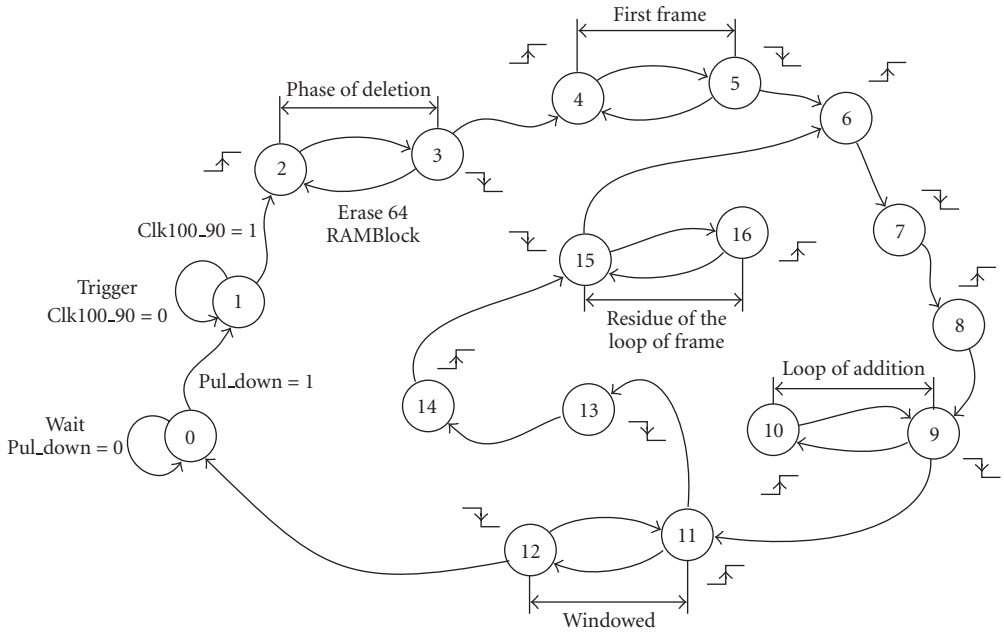


Figure 7: State diagram of the control machine.

complete the calculation to 200 clock cycles. Even with this addition, the usage factor of the remaining stages is never higher than 55% of the total time. However, the inactivity of these stages has no influence on the total processing time as they work in parallel with the histogram stage.

5. Experimental Results

The entire design was simulated and implemented using the software package ISE 8.2i from Xilinx and tested on a development board "SPARTAN-3 Starter Board" from Digilent provided with an XC3S1000 FT256 speed grade 4 chip, 1 Mbyte of SRAM (256 Kb × 32), and 10 ns access time.

The SOFT-HARD comparison uses four compressed videos from an MPEG-7 content set belonging to different genres (drama, sport, and news), all of which have an average length of 25,000 frames. The algorithm was implemented using fixed point arithmetic. Overflow has been avoided by selecting suitable resource sizes for each stage. The main considerations are discussed in the following.

The bus width at the multiplication output is twice as wide as the incoming factors and only half as wide for the square root case. The size of the accumulators for the sums of products enables storing the quantity $N \times \mathrm{MAX}$, where MAX is the maximum value attained by these factors and N is the number of terms to be added.

Table 2: Summary of resources usage.

Selected device: 3s1000ft256-4			
Resource	Used	Available	Ratio
Slices	952	7680	12%
Slice Flip Flops	1170	15360	7%
4 input LUTs	1531	15360	9%
Bonded IOBs	80	173	46%
BRAMs	7	24	29%
MULT18X18s	7	24	29%
GCLKs	3	8	37%
DCMs	1	4	25%

Source: ISE 8.2i provided by XILINX.

Table 3: Summary of number of clock cycles taken per operation.

Histogram calculation	800
Intermediate storage	32
Windowed correlation (2 sums)	28
Square root	16
Product	1
Division	32

The Xilinx ISE synthesis tool provided the report shown in Table 2 concerning the resources occupied and available at the chip level. In regard to these data, it should be made clear that in applications like the one described in this paper—where there is no direct link between the number of BlockRAMs and multipliers used—the report is inaccurate, since the use of a multiplier block is assumed to disable the use of its associated BlockRAM and vice versa, due to the peculiarities of the Xilinx FPGA. Thus, the usage factor must be calculated considering them as available pairs, in this case 14 multiplier-BlockRAM pairs out of 24 available (58% busy), which is almost twice the value (29%) reported by the Xilinx synthesis tool.

The interface with the external memory was debugged using an Agilent 64622D oscilloscope.

Close examination of the summary of the number of clock cycles taken by each individual stage shown in Table 3 suggests that it might be desirable to reduce the parallelism of nearly idle stages, which are very resource hungry. One example is the circuit depicted in Figure 5, which employs such scarce and valuable resources such as multipliers.

To evaluate circuit performance, we compared a software implementation presented in Table 4 (on a PC Pentium IV at 2 Ghz) for the calculation of windowed values and sum of products to the hardware implementation on FPGA (Spartan 3 at 120 MHz) developed in this work. This comparison is very illuminating because the FPGA is a reconfigurable device that is impaired by the overhead of configuration circuits that slow down clock rate and increase the silicon area compared to a custom VLSI device. Despite this handicap, the calculation speed obtained with the FPGA exceeds that obtained using SIMD instructions (SSE multimedia extension) by a factor of 1.75 : 1 as regards nominal delay, and by a factor of 25 : 1 as regards clock cycles. We attribute this improvement to the optimization of data flow and to concurrence.

Table 4: Hard-Soft solution comparison.

Implementation	ms	cycles
PC (sequential programming)	2500	5000
PC (sequential optimized)	750	1500
PC (SSE with Intrincsics)	350	700
FPGA (SPARTAN 3-VHDL)	200	28

6. Conclusions

In this paper we have presented an FPGA implementation to calculate similarities between two frames. Our design works with the DC coefficients of compressed video frames to compute the frame histogram. Similarity is calculated by applying a windowed cross-correlation to frame histograms. Our implementation has efficiently solved the problems arising from the management of constant data flow from video frames. Thus, we have designed a windowed sum of products stage which completes the calculation of 64 sum terms in only 28 clock cycles.

Although our system is a hardware implementation of a video segmentation technique, extending these results to other data processing applications is possible, simply because any calculations involving data correlation always takes the form of a sum of products. Its success in such applications will depend strongly on both the ability to maintain constant data flow through the system and how the numerical resolution of each stage is chosen.

Acknowledgments

This work was supported in part by the Ministry of Education and Science (CICYT) of Spain under Contract TIN2006-01078 and Junta de Andalucia under Contract TIC-02800.

References

[1] F. Bensaali, A. Amira, and A. Bouridane, "Accelerating matrix product on reconfigurable hardware for image processing applications," *IEE Proceedings: Circuits, Devices and Systems*, vol. 152, no. 3, pp. 236–246, 2005.

[2] N. İsmailoğlu, O. Benderli, S. Yeşil, R. Sever, B. Okcan, and R. Öktem, "GEZGİN-2: an advanced image processing subsystem for earth-observing small satellites," in *Proceedings of the 2nd International Conference on Recent Advances in Space Technologies (RAST '05)*, pp. 605–610, IEEE JNL, Istanbul, Turkey, June 2005.

[3] R. Porter, K. McCabe, and N. Bergmann, "An applications approach to evolvable hardware," in *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware*, pp. 170–174, IEEE JNL, Pasadena, Calif, USA, 1999.

[4] E. Saez Peña, *Segmentación automática de video*, Tesis Doctoral, 2006.

[5] H. Quinn, L. A. S. King, M. Leeser, and W. Meleis, "Runtime assignment of reconfigurable hardware components for image

processing pipelines," in *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '03)*, pp. 173–182, April 2003.

[6] T. Sugimura, S. Jeoung Chill, H. Kurino, and M. Koyanagi, "Parallel image processing field programmable gate array for real time image processing system," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '03)*, pp. 372–374, 2003.

[7] E. Sáez, J. I. Benavides, and N. Guil Mata, "Reliable real time scene change detection in MPEG compressed video," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '04)*, vol. 1, pp. 567–570, Taipei, Taiwan, June 2004.

[8] F. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard*, Chapman & Hall, Boca Raton, Fla, USA, 1996.

[9] M. Calviño, J. I. Benavides, S. Geninatti, F. J. Hormigo, and J. Villalva, "Hardware accelerators for the microblaze software embedded processor," in *Proceedings of the 6th Southern Programmable Logic Conference (SPL '07)*, 2007.

[10] J. P. Deschamps, G. J. A. Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits FPGA, ASIC, and Embedded Systems*, John Wiley & Sons, New York, NY, USA, 2006.