Configuration Sharing Optimized Placement and Routing

Piotr Stepien
School of DEC
Bournemouth University
Poole, UK
Email: pstepien@bournemouth.ac.uk

John Cobb
School of DEC
Bournemouth University
Poole, UK
Email: jcobb@bournemouth.ac.uk

Abstract—Reconfigurable systems have been shown to achieve very high computational performance. However, the overhead associated with reconfiguration of hardware remains a critical factor in overall system performance. This paper discusses the development and evaluation of a technique to minimize the delay associated with reconfiguration based upon optimized sharing of configuration bit streams between design contexts. This is achieved through modified placement and routing algorithms.

Keywords-FPGA; placement; routing; reconfiguration;

I. INTRODUCTION

Reconfigurable systems aim to achieve the execution performance of hardware with the operational flexibility of software. The most widely used implementation platform enabling this goal is the Field Programmable Gate Array (FPGA). Current FPGA design techniques tend to focus on maximizing algorithm execution and data throughput for single-context designs. However, reconfigurable systems need to implement different designs at different times on the same hardware architecture. If the overall performance of the system is to be maintained it is clear that reconfiguration latency must be minimized. Numerous approaches have been proposed to reduce this latency [1] including storing each design in separate context memories on the FPGA [2], [3]; optical reconfiguration [4]; bit stream compression [5], [6], [7], [8]; additional high level block optimization [9] and architectures specifically designed for reconfiguration, such as PACT[10]. However, the added cost or limited effectiveness of these approaches appears to have limited their widespread uptake. In this paper a novel solution is presented based on a modified FPGA placement and routing algorithm. The goal of the approach is to increase architectural commonality between the multiple hardware designs used in a reconfigurable system. The aim of this study was therefore to assess if the performance penalty imposed by making designs more generic could be offset by the advantages of minimizing reconfiguration latency. Clearly, such an approach is unsuitable for high performance computing where execution speed, data throughput and minimum reconfiguration latency are essential. However, in certain applications, such as, mobile technology a key requirement is seamless transition between operational contexts [11], [12]. The approach described here also offers important advantages in terms of power consumption which is of primary importance in mobile systems. Power consumption during reconfiguration increases proportionate to the size of the reconfiguration bit stream and can be an important factor in the operational capacity of a battery operated device.

II. BACKGROUND

A. Partial Reconfiguration

Direct access to a single configuration register on a FPGA, requires complex configuration interface circuitry; therefore single configuration registers are typically grouped into clusters. A single cluster is the smallest amount of configuration data loaded onto the FPGA. For the Xilinx Virtex FPGA used in this study, the smallest configuration data block is represented by a configuration frame spanning an entire CLB column.

To configure a single FPGA programmable cell, all the frames covering the particular cell need to be loaded into the FPGA. Detailed registers mapping depends on the cell configuration and requires definition of the logic/routing resources that need to be configured. Therefore the content of a FPGA configuration data bit stream is critically dependent on the results of placement and routing algorithms.

To demonstrate complexity of the single CLB column configuration process of Xilinx Virtex FPGAs, a summary of configuration data for Xilinx Virtex FPGA family is presented in Table I.

TABLE I. XILINX VIRTEX FPGA CONFIGURATION DATA SUMMARY

Device name	CLB Array size	Frame size	Column size
XCV50	16 x 24	324	15,552
XCV1000	64 x 96	1,188	57,024
XCV3200E	104 x 156	1,908	91,584

To configure an entire FPGA, all frames need to be loaded. However, design reconfiguration can be performed by either re-loading the entire bit stream or by re-loading only those frames which contain data different from the previous context i.e. partial reconfiguration.

B. *Configuration Data Optimisation Problem*

An important aim of this research was to achieve an automated method of maximizing configuration sharing in multi-context designs. Figure 1 presents the concept of a reconfigurable platform supporting partial reconfiguration with a two context design to be implemented on the platform, one context at a time.

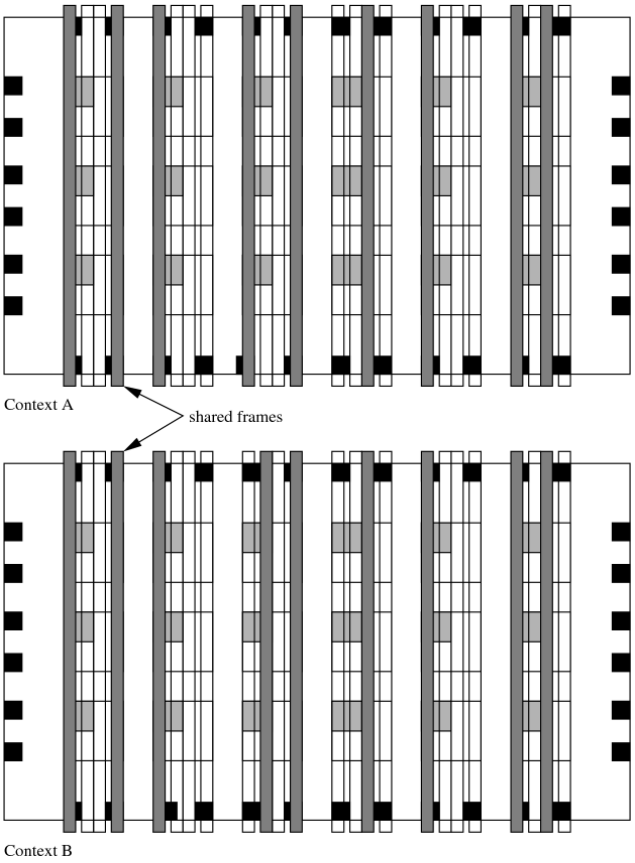

Figure 1. Example of multi-context design project using the same platform. Where frames can be shared between design contexts reconfiguration latency can be reduced.

Shared configuration represents the case where FPGA resources controlled by the shared frames are configured in exactly the same way in more than one context. Therefore when switching contexts shared frames do not need to be reloaded saving time and reducing storage requirements.

To demonstrate how placement decisions affect frame utilization a simple FPGA placement in two variants is presented in Fig. 2. This illustrates graphically how routing of the same netlist by two proprietary tools can result in frame utilization that whilst providing comparable design solutions has a detrimental impact on reconfiguration latency.

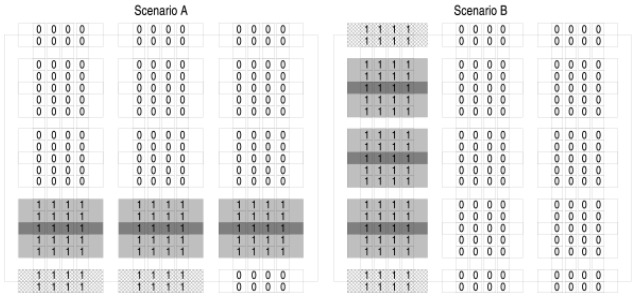

Figure 2. Two design implementations with different configuration data sizes: Scenario A = 12 frames, Scenario B = 4 frames. Scenario A imposes a significant overhead on reconfiguration performance.

III. METHODOLOGY

A. *Target Architecture*

To demonstrate applicability of the new placement and routing methodology development has focused on a real-world FPGA architecture. Xilinx Virtex technology was selected on the basis of availability of a bit stream manipulation - Xilinx JBits [13]. To simplify the process of evaluating new placement and routing methods initial development was targeted to the XCV50 – the smallest device in the Xilinx Virtex family [14]. The XCV50 has an 16 x 24 array of CLBs controlled by 1152 configuration frames [14], [15], [16].

Configuration of a single Xilinx Virtex CLB cell is controlled by 864 bits. Half of these bits are used to configure the routing switch matrix itself, whilst the other half control LUT configuration and the CLB input and output switch matrix.

B. *Benchmarks Suite*

A set of real–world FPGA benchmarks was developed using Verilog and VHDL sources available from ITC99 benchmark suite [17] and free IP cores available at www.opencores.com [18]. The LeonardoSpectrum compiler (Mentor Graphics) was used to compile circuit sources to Xilinx Virtex XCV50 netlists. Bit streams were generated for each netlist using Xilinx ISE, executing on a dual Intel Xeon 2GHz/512k Dell PC under MS Windows 2000 SP4. In both packages default settings were used (e.g. commercial temperature). Test circuits were randomly paired to provide two-context designs for evaluation of the new reconfiguration optimized placement and routing algorithms.

C. *Framework Description*

1) *Bitstream Comparison Tool*

To determine the frames utilization characteristic of each XCV50 bit stream, a Bit stream Comparison Tool (BCT) was developed in C using Kdevelop [19]. BCT takes two bit streams, scans through their content and performs frame to frame comparison. BCT delivers a set of statistics showing

the percentage of similar frames and frame distribution categorized by difference using a bin size of ten bits.

2) Simultaneous Multi-Context Placement and Routing

Using the Xilinx Virtex FPGA architecture together with the JBits tools [13] a new placement and routing framework was developed using the JBuilder compiler. This 'Simultaneous Multi-Context Java Placement and Routing tool' (SMC JPR) performs simultaneous placement and routing of two contexts, based on design information extracted from Xilinx Virtex bit streams. SMC JPR design flow is presented in Figure 3.

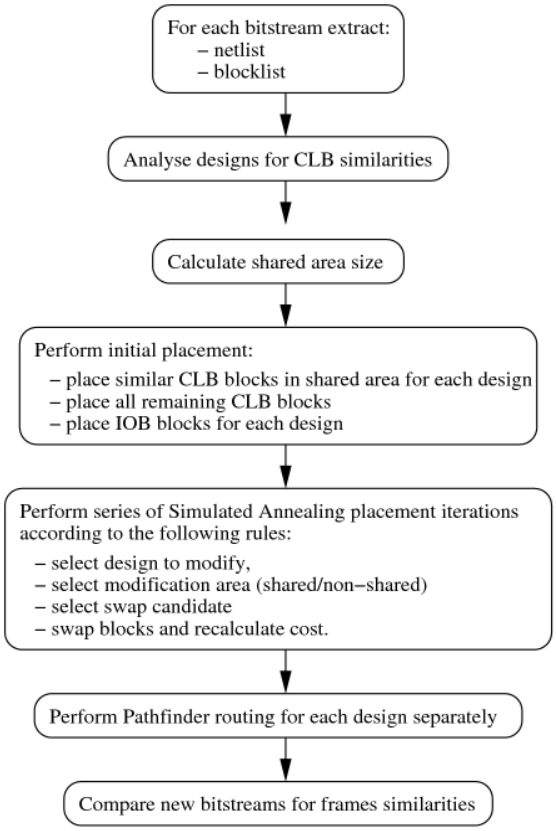

Figure 3. Simultaneous Multi-Context Placement and Routing Design Flow.

Placement and routing algorithms have been incorporated from our previous work presented in [8]. The placement algorithm is based on a simulated annealing scheduling scheme and the routing algorithm is based on the Pathfinder maze-router. Full details including code listings can be obtained from the principal author.

SMC JPR takes two bit streams as a design entry (one for each context), extracts a netlist from each bit stream and performs a series of comparisons to extract similarities between contexts netlists. Based on the results, the placement space is then divided into two areas: one for shared design blocks and the other for all the remaining designs blocks. During initial placement similar blocks are placed in exactly the same location for both contexts. This helps to ensure that such blocks are located within a common frame. Block swaps and/or moves within a shared column are allowed. The router was constrained to avoid using empty frames and prevent routing through any area already allocated by the placer this significantly simplifies the solution space. Consequently placement and routing times were not significantly increased compared to existing approaches.

IV. MULTI-CONTEXT PLACEMENT ALGORITHM EVALUATION

The total number of shared blocks was used to determine the minimum required placement area. Using this figure the minimum number of configuration frames can be calculated and is dependent on the technology. With a column-based configuration interface as used in Xilinx Virtex, the placer calculates the minimum number of columns required to host the shared blocks.

A. Initial Placement

With the entire FPGA divided into shared and design specific areas a traditional random initial placement does not make sense. Instead a two phase process was necessary; random placement of shared blocks within the shared area, followed by random placement of non-shared blocks in the non-shared area. If random placement is applied to the shared area, it can be done only once for all contexts otherwise this area will not be able to share configuration, as only entire columns of the same blocks can share configuration.

B. Next Step Criteria

Modifications proposed by the placer are constrained to improvement in design timing and are prevented from interfering with shared and non-shared area division. However, since the placement area has been divided into two separate areas, different modification rules apply to each of them. Blocks located in the shared area can be swapped or moved, but any placement modification has to be applicable to all designs involved. Blocks located within the non-shared area can be relocated within the entire non-shared area, however by using bit stream compression placement methods the number of frames required to implement the non-shared part of the design were found to be minimized.

V. MULTI-CONTEXT ROUTING ALGORITHM EVALUATION

FPGA routing identifies the resources necessary to provide connectivity for every design net. The location of net terminals is specified during the placement process, so the router has to prove that all design contexts can be successfully routed given a specific completed placement.

A. *Multi-Context Routing Algorithm Criteria*

From the router's point of view, nets can be divided into three categories depending on their terminal allocation as demonstrated in figure 4.

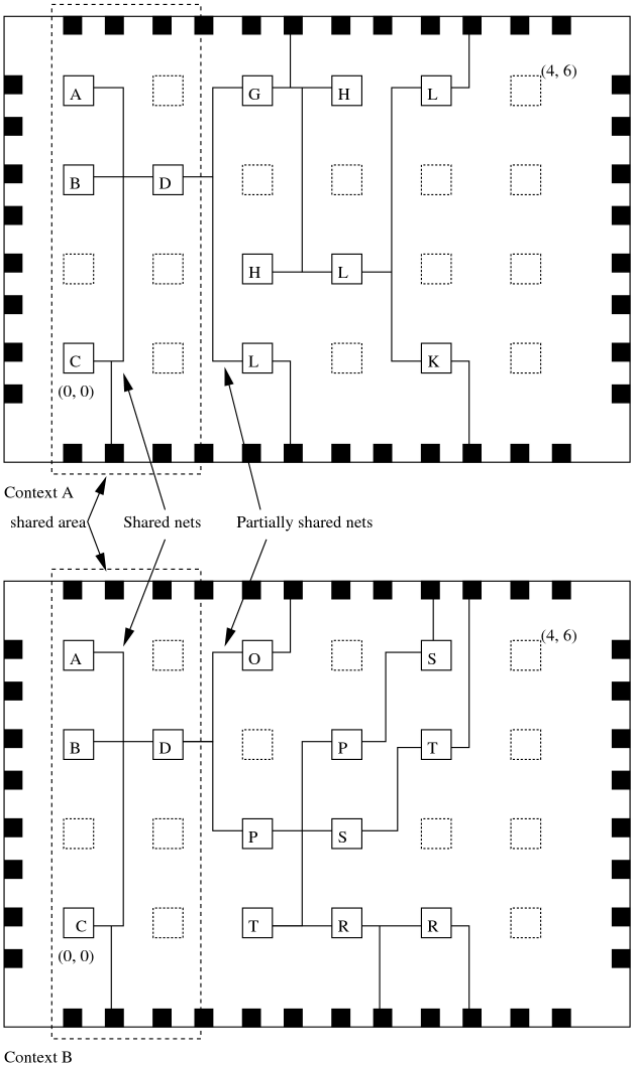

Figure 4. Example of placement showing three different nets examples.

Nets belonging entirely to the non-shared area can be routed using routers available for single design routing, such as the frames optimized Pathfinder router described in detail in [8].

Nets which belong entirely to the shared area and cross-area nets can potentially improve frame sharing if routed in such a way that they overlap each other as shown in figure 4.

B. *Routing Cost Function*

The Dijkstra algorithm was used to determine the best connection between blocks. This approach requires that a cost be associated with every routing resource to find the shortest path. This cost is usually calculated based on the wire length or delay caused by the single routing resource. The cost of each path represents the sum of all routing resources accrued along the path. To evaluate routing costs in this study it was necessary to introduce a connection cost function as defined by equation 1.

$$Cost_{(A,B)} = \sum_{i=A}^{i=B} [Cost_{wire}(i) + Cost_{switch}(i, i+1)] \quad (1)$$

Where $Cost_{wire}$ for each routing resource is calculated using equation (2) and $Cost_{switch}$ connecting two routing resources i and i + 1 is based of the current frames utilization cost associated with this switch.

When the routability–driven routing algorithm is used, the cost of using routing resource n when it is reached by routing resource m has been calculated according to the following formula:

$$Cost(n) = b(n) \cdot h(n) \cdot p(n) + BendCost(n, m) \quad (2)$$

Where b(n) is a base cost, h(n) is a measure of historic congestion, p(n) represents present congestion and BendCost(n, m) penalizes bends when global routing is performed.

Congestion was calculated using the following equations:

$$p(n) = 1 + max(0, [oc(n) + 1 - cp(n)] \cdot p_{fac}) \quad (3)$$

$$h(n)^i = 1, i = 1 \quad (4)$$

$$h(n)^i = h(n)^{i-1} + max(0, [oc(n) - cp(n)] \cdot h_{fac}), i > 1 \quad (5)$$

Where oc(n) is the number of nets claiming to use routing resource n and cp(n) represents maximum number of nets that can legally use resource n. The values of hf_{ac} and pf_{ac} define routing schedule and according to [20] the best router performance has been achieved for $0.2 < h_{fac} < 1$ and $p_{fac} = 0.5$ during the first routing iteration, and then 1.5 to 2 times its previous value in each subsequent iteration.

Frame utilization cost FUCost(i) has been calculated according to the equation 6:

$$FUCost(i) = \left[1 - \frac{BitsSet}{FL}\right] \quad (6)$$

Where BitsSet denotes the number of bits set within the frame and FL denotes the total number of configuration bits available within the frame, which for XCV50 equals 324 [14].

VI. MULTI-CONTEXT PLACEMENT AND ROUTING RESULTS

A. Designs Similarity Analysis

To analyze similarities between different bit streams bit streams from traditionally placed and routed benchmark designs were cross correlated to establish similarities in their bit streams. The results of this analysis are presented in Table II. These results show that although there are pairs of designs with a certain number of similar CLBs, the average bit stream similarity is less then 1%. This is due to the fact, that they all have been placed and routed individually.

TABLE II. MULTI-CONTEXT DESIGNS SIMILARITY SUMMARY.

	Context pair	Size [%]	Shared blocks	Shared frames
A	b04_out.bit tb_04_chain_x02.bit	11 22	25	239
B	b12_out.bit tb_12_chain_x2.bit	30 60	63	4
C	tb_05_12.bit tb_04_chain_x4.bit	45 45	58	0
D	cf_fp_mul_c_5_10_out.bit cf_fp_mul_p_5_10_out.bit	34 43	113	0
E	cf_interleaver_6_8_out.bit cf_interleaver_6_64_out.bit	13 86	74	0
F	tb_10_chain_x20.bit tb_11_chain x6.bit	86 48	21	0

B. Experimental Results

To test the feasibility of the new approach Simultaneous Multi-Context Placement and Routing (SMC P&R) was performed on sets of design pairs. In this experiment the cost functions used by the placement and routing algorithm were purely frames-sharing oriented without any other constraints. Figure 5 shows graphically the high level of design commonality typically achieved by the new approach. Table III summarizes the results obtained.

C. Frames Sharing Analysis

The results presented in Table III show that it is possible to significantly increase the number of shared frames in comparison to currently used P&R approaches. As might be expected the achieved improvement ratio is considerable for small circuits (51% for circuit sizes requiring 20-30% FPGA utilization). However, these results also demonstrate that the new technique is also beneficial for bigger circuits (by an average of 33% for circuits utilizing up to 60% of FPGA resources).

Figure 3. Example of SMC P\&R performed on two-context design.

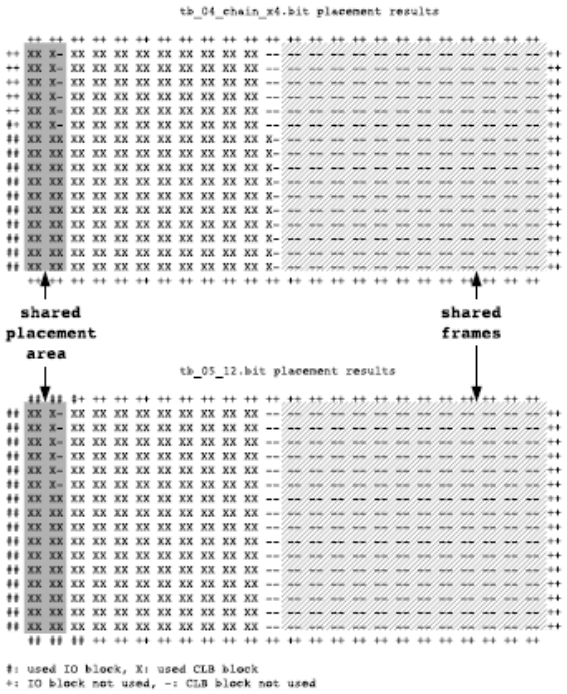

Figure 5. Example of SMC P\&R performed on a two-context design illustrating the high level of physical commonality.

TABLE III. MULTI-CONTEXT DESIGNS BENCHMARKS SUMMARY.

Design pair	Size [%]	SB	SF SC	SF SMC	F [%]	CP SC	CP SMC	CP [%]
A	11 22	25	239	821	51	9 12	12 14	-25 -14
B	30 60	63	4	382	33	10 12	13 14	-23 -14
C	45 45	58	0	527	46	9 8	10 11	-10 -27
D	34 43	113	0	518	45	9 9	12 11	-25 -27
E	13 86	74	0	102	9	8 9	11 11	-25 -18
F	86 48	21	0	65	6	8 7	10 9	-20 -22

D. Timing Analysis

Timing analysis shows, that Critical Path (CP) uses on average 25% more routing resources when placed and routed with SMC P&R. This is due to the heavily packed placement which constrains router freedom, and the fact that long wires can be directly accessed at selected CLB locations, so the router has to use single wires to access them in the neighboring location, which increases the number of routing resources used by the net on the critical path.

E. CLB Blocks Sharing

The results of CLB blocks sharing (SB) analysis between two designs as presented in Table III show that there are a number of blocks with the same content (up to 15%), although all the benchmark designs used were compiled separately without the intention to share any of their resources. It is evident therefore that the number of shared CLB blocks could be further improved by using resource-sharing methodologies during the design automation steps that precede placement and routing.

F. IO Pins Allocation

SMC JPR focuses on frame sharing within the CLB area. As explained in our groups previous work [8], pins allocated at the top and the bottom side of the Xilinx Virtex FPGA are configured by the frames controlling the CLB area. Only IO pins located on the left and right hand sides of the chip are controlled by a dedicated set of frames and therefore do not influence CLB frame content. SMC JPR uses IO pins on the side of the chip first, and then top/bottom IO pins if necessary.

IO pin allocation is usually defined prior to FPGA design flow and is yet another design constraint, although the approach of using floating IO pins shows benefits of combining certain IO pin allocation with configuration interface architecture to achieve a better frames sharing ratio.

G. Scalability Of The Approach

The presented SMC JPR approach has been tested for two-context designs, but it can be extended to handle designs with three or more contexts. Overall frames sharing ratio with three contexts or more will depend on the netlists similarity factor and is likely to decrease with increasing number of contexts.

VII. CONCLUSIONS

Results from evaluation of the novel placement and routing methodology outlined in this paper demonstrate the feasibility of increasing the amount of shared configuration between multiple context designs. As a consequence the authors believe the approach offers potential for significantly reducing FPGA reconfiguration latency. The ability to transform netlist similarities into configuration data similarities has been shown to work well for the difficult case, where the configuration data frame consists of bits controlling parts of different FPGA resources (IOB, CLB, routing), as any modification to placement or routing applies to a number of configuration frames at once. However, the approach typically incurs a design performance penalty due to an increase in critical path net delays. This would prevent its application in high performance computation applications however it offers an effective solution for applications where seamless context switching and low power are the primary performance criteria.

VIII. REFERENCES

[1] S. Hauck and A. DeHon, Eds., Reconfigurable Computing. The Theory And Practice Of FPGA-Based Computation. Morgan Kaufmann Publishers, 2008.

[2] J. Brown, D. Chen, I. Eslick, E. Tau, and A. DeHon, “DELTA: Prototype for a first-generation dynamically programmable gate array”, in Transit Note 112. MIT Artificial Intelligence Laboratory, 1994.

[3] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, “A time-multiplexed FPGA”, in IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), 1997, pp. 22–28.

[4] M. Vasilko and D. Ait-Boudaoud, “Optically Reconfigurable FPGAs: Is This a Future Trend?” in Field-Programmable Logic. Smart Applications, New Paradigms and Compilers, R. W. Hartenstein and M. Glesner, Eds. Springer, September 1996, pp. 270–279.

[5] Z. Li and S. Hauck, “Configuration compression for Virtex FPGAs”, in Proceedings of the IEEE Symposium for Custom Computing Machines, April 2001.

[6] S. Hauck, Z. Li, and E. Schwabe, “Configuration compression for the Xilinx XC6200 FPGA”, in IEEE Symposium on FPGAs for Custom Computing Machines, April 1998, pp. 138–146.

[7] L. Sterpone, M. Violante, A. Molino, F. Vacca, and G. Masera, “A new approach to compress the configuration information of programmable devices”, in Proceedings of the DATE2006: IEEE Design, Automation and Test in Europe Conference, 2006.

[8] P. Stepien and M. Vasilko, “On Feasibility of FPGA Bitstream Compression During Placement and Routing”, in Proceedings of 2006 International Conference on Field Programmable Logic and Applications (FPL), 2006, pp. 749–752.

[9] M. Dyer, C. Plessl, and M. Platzner, “Partially Reconfigurable Cores for Xilinx Virtex”, in Field-Programmable Logic and Applications (FPL), M. Glesner, P. Zipf, and M. Renovell, Eds. Springer, September 2002, pp. 292–301.

[10] PACT XPP Technologies, “http://www.pactcorp.com”, 2003, PACT XPP.

[11] J. Helmschmidt, E. Schuller, P. Rao, S. Rossi, S. di Matteo, and R. Bonitz, “Reconfigurable Signal Processing in Wireless Terminals,” in Designers’ Forum (DATE ’03), D. Sciuto and D. Verkest, Eds. IEEE Computer Society, March 2003, pp. 244–249.

[12] N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, “A Reconfigurable Platform for Real-Time Embedded Video Image Processing”, in Field-Programmable Logic and Applications, P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, Eds. Springer, September 2003, pp. 606–615.

[13] Xilinx, “JBits SDK”, 2000. http://www.xilinx.com/products/jbits/index.htm

[14] Xilinx, “Virtex 2.5V Field Programmable Gate Arrays, Product Specification”, Data sheet DS003, 2 April 2001.

[15] Xilinx, “Virtex FPGA Series Configuration and Readback”, Application Note XAPP138, 11 July 2002.

[16] Xilinx, “Virtex Series Configuration Architecture User Guide”, Application Note XAPP151, 27 September 2000.

[17] F. Corno, M. S. Reorda, and G. Squillero, “RT-level ICT’99 benchmarks and first ATPG results”, IEEE Design & Test of Computers, vol. 17, no. 3, pp. 44–53, July-August 2000.

[18] Opencores, “http://www.opencores.org”, 2005.

[19] KDevelop, “Kdevelop3”, http://www.kdevelop.org/.

[20] V. Betz, J. Rose, and A. Marquardt, “Architecture and CAD for Deep-Submicron FPGAs”, Kluwer Academic Publishers, 1999.