A Modified Merging Approach for Datapath Configuration Time Reduction

Mahmood Fazlali[1,2], Ali Zakerolhosseini[1], and Georgi Gaydadjiev[2]

[1] Department of Computer Engineering, Shahid Beheshti University G.C, Tehran, Iran
[2] Computer Engineering Lab., Delft University of Technology, Delft, The Netherlands
fazlali@cc.sbu.ac.ir, a-zaker@sbu.ac.ir,
g.n.gaydadjiev@tudelft.nl

Abstract. This paper represents a modified datapath merging technique to amortize the configuration latency of mapping datapaths on reconfigurable fabric in Run-Time Reconfigurable Systems (RTR). This method embeds together the different Data Flow Graphs (DFGs), corresponding to the loop kernels to create a single datapath (merged datapath) instead of multiple datapaths. The DFGs are merged in steps where each step corresponds to combining a DFG onto the merged datapath. Afterwards, the method combines the resources inside the merged datapath to minimize the configuration time by employing the maximum weighted clique technique. The proposed merging technique is evaluated using the Media-bench suit workloads. The results indicate that our technique outperforms previous HLS approaches aimed at RTR systems and reduces the datapath configuration time up to 10%.

Keywords: Reconfigurable Computers, Run-Time Reconfiguration, CAD Algorithms for FPGA and Reconfigurable Systems.

1 Introduction

Multimedia applications contain computationally intensive kernels that demand hardware implementation in order to exhibit real time performance. The kernels can be accelerated by employing reconfigurable units which provide flexibility and reusable hardware resources. Often FPGA resources are limited and all the kernels cannot be mapped inside the FPGA. Hence, the kernels are to be configured at run-time in order to create a virtual hardware and accelerate more sections of the applications [1,2]. This is called Run-Time Reconfiguration (RTR).

The RTR in FPGA involves dynamic reconfiguration that with the current technology is a time-consuming process and hence affects the available system performance. For example, the MOLEN reconfigurable processor which is depicted in Fig.1 utilizes custom configured hardware to execute computational intensive functions [3]. The MOLEN architecture consists of two parts; General-Purpose Processor (*GPP*) and Reconfigurable Processor (*RP*), which is usually implemented on an FPGA. The RP is used for hardware acceleration. The execution phase by the RP is divided into two distinct steps: *set* and *execute*. In the *set* phase, RP is configured to perform the

required datapath and, in the *execute* phase the actual function is executed. However, the run-time reconfiguration in the *set* phase imposes considerable overhead to the performance of the system. Therefore, the configuration should be done as soon and efficient as possible.
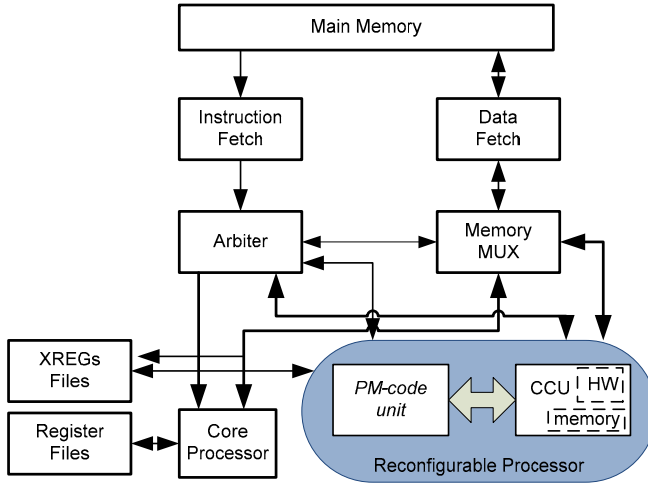

Fig. 1. MOLEN Hardware Organization

There are many proposals for the configuration time reduction in RTR system. Some researchers have attempted to reduce configuration overhead by reusing the same hardware for different applications or rearranging the execution order of tasks in a sequence that requires lower number of reconfigurations [4]. Thus, however, it can not be applied to many realistic applications. Cashing the configuration bit-stream is another solution that was represented in [2] for the configuration overhead reduction however, spatial and temporal locality of references is not yet proven or accepted as a general principle in replacement policy for RTR systems, and needs more investigation. The configuration overhead can be efficiently hidden by employing approaches such as configuration perfecting where a static schedule is present [2]. Such totally predictable application schedules form only a small subset of the total class of the applications suitable for reconfigurable hardware acceleration.

The average configuration time can be computed using the average configuration bit-stream and the speed of the configuration interface [5]. So, it is apparent that the reduction in the bit-stream length is conducting to the reduction in the configuration time. Some researchers focus on reducing the configuration time of a FPGA by compressing the bit-stream. However, such techniques are costly and affect the performance of the system [6]. Therefore, it is better to amortize the configuration time using High Level Synthesis (HLS) [7].

In HLS, Data Flow Graphs (DFG) are created for the computational intensive kernel loops. Afterwards, the resources in the DFG are shared to create a datapath. In this way, the hardware cost is reduced. The synthesis process comprises the major tasks of

scheduling, resource allocation, resource binding, and interconnection binding [8,9,10].

Making a multimode datapath instead of multiple datapaths can effectively reduce the hardware cost [11]. Datapath merging is a technique introduced to create a reconfigurable datapath for two or more DFGs [12,13]. It enables the reuse of hardware resources by identifying similarities among several DFGs. This technique was employed in [14] for the reduction of configuration time. This technique, however, cannot minimize the configuration time in *RTR* systems.

The main contribution of this paper is presenting a modified datapath merging technique for the configuration time reduction in an RTR system. In the proposed technique we combined the resources inside the merged datapath to reconstruct it to a new merged datapath which requires fewer functional units and multiplexers in order to minimize the configuration bit-stream.

The next section explains the configuration time reduction in datapath merging. Section 3 presents the suggested datapath merging technique. The experimental results are included in Section 4, and Section 5 concludes the paper.

2   Configuration Time Reduction in Datapath Merging

The problem can be considered as merging only those DFGs that correspond to computational intensive kernels while making a merged datapath with a reduced configuration time.

Let DFG $G=(V,E)$, where $V=\{v_i \mid i=1...n\}$, is a set of vertices and $E=\{e_j \mid j=1...m\}$, is a set of edges. A vertex $v \in V$ represents an operation executable by a functional unit that has a set of input ports p. An edge $e=(u,v,p) \in E$, indicates a data transfer from the vertex u to the input port p of vertex v.

Vertex-merging is the creation of a vertex v' replacing vertices $v_i\ v_j...\ v_k$ and edge-merging is the creation of an edge $e'=(u',v',p')$ replacing edges $e_i = (u_i,v_i,p_i)$, $e_j = (u_j,v_j,p_j)..\ e_k = (u_k,v_k,p_k)$.

In order to merge edges, their vertices are to be merged while their corresponding input ports are matched together. Here, a functional unit is used for each merged vertex v' capable of performing the functions of vertices mapped onto v'.

A merged datapath $MD=(V',E')$, corresponding to DFGs $G_i=(V_i,E_i)$ $i=1...n$, is a directed graph. A vertex $v' \in V'$ represents a merge of vertices $v^j \in V_j$ and an edge $e'=(u',v',p') \in E'$ represents a merge of edges $e^j \in E_j$ where $j \in J \subseteq \{i,...,n\}$.

The configuration time T_C of the merged datapath $MDP=(V',E')$ is:

$$T_C= T_F + T_I$$

Where $T_F = \sum_{\forall v' \in V'} T_f(v')$ is the functional units configuration time and

$T_I = \sum_{\forall v' \in V'} T_i(MUX)$ is the multiplexer's configuration time in the merged datapath [14]. $T_f(v')$ is the configuration time of a functional unit (or storage unit) allocated to v', and $T_i(MUX)$ represents the configuration time of multiplexers employed at the input port of a vertex. There are different ways to merge vertices and edges from DFGs and this, in effect, can produce several merged datapaths for the input DFGs.

An optimized merging method is required in order to manage resource allocation, resource binding, interconnection binding and also minimizing the datapath configuration time of *DPM*. It means the realization of a *MDP* for the DFGs is desirable if T_C is minimal.

Fig.2 illustrates an example of datapath merging where DFGs G_1 and G_2 from this figure are merged and the merged datapath *MDP* is created. Considering these DFGs, if operation of a vertex from G_1 and operation of a vertex from G_2 can be computed with the same type of functional unit, they will become potential for merging. For example, $a_1 \in G_1$ and $b_1 \in G_2$ can be executed by the same functional unit. Thus, these vertices are merged together and the vertex (a_1/b_1) is created for them in *MDP*. If a vertex cannot be merged onto other vertices, it will remain in the merged datapath without any modification. After merging two vertices, multiplexers are employed in the merged datapath to select the currect input operand. This is illustrated in the input ports of vertex (a_5/b_3) in Fig.2
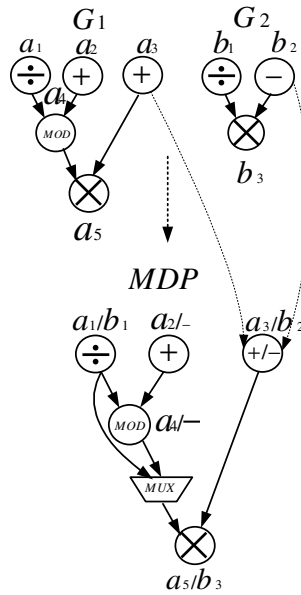

Fig. 2. The merged datapath *MDP* for DFGs G_1 and G_2 [14]

An edge from G_1 cannot be merged onto an edge from G_2 unless the vertices of the edges are merged. As it can be seen in Fig.1, because of merging both of the vertices a_3 and $a_5 \in G_1$ onto the other vertices b_2 and $b_3 \in G_2$, the edges (a_3, a_5) and (b_2, b_3) are merged together and the edge $(a_3/b_2, a_5/b_3)$ is created instead in *MDP*. In this case, it is not necessary to use a multiplexer in the input ports of the vertex (a_5 / b_3) to select the input operands.

3   The Proposed Datapath Merging Technique

Although merging DFGs using Integer Linear Programming (*ILP*) can minimize the merged datapath configuration time, it is an *NP*-complete problem where the number of DFGs increases or number of nodes in DFGs increases [12]. Therefore, in the proposed datapath merging technique, the DFGs are merged together in steps. Afterward, the resources inside the merged datapath are combined to optimize the configuration time. In this way, the desired merged datapath for the input DFGs is created.

Combining the resources inside the merged datapath paves the way for having the merged datapath with smallest configuration time. Thus, we can define the merging problem to create the desired merged datapath as follows:

Given a merged datapath MDP, the desired merged datapath, MDP', is realized such that the merged datapath configuration time T_C is minimal.

Fig.3 shows an example of datapath merging proposed in this paper to merge DFGs. In Fig. 3(a), five simple DFGs, G_1,..., G_5 are illustrated. Each hardware unit and interconnection unit (multiplexer) has its own configuration time. These DFGs are merged in steps employing the method in [14] while the datapath configuration time, T_C, is reduced and *MDP* is created. Fig. 3(b) shows the results of sharing the resources inside *MDP* to create *MDP'* for the DFGs. It combines the resources inside the *MDP,* together to minimize the configuration time. In this figure, the vertices c_1 and e_1 from *MDP* have been merged to create a vertex c_1/e_1 in the *MDP*'. On the other hand, vertices $a_2/b_2/c_2$ and d_2/e_2 are merged together to create vertex $a_2/b_2/c_2 /d_2/e_2$ in *MDP*'. Therefore, two edges $(a_2/b_2/c_2 , c_1)$ and $(d_2/e_2 , e_1)$ in *MDP* can be merged together to create an edge $(a_2/b_2/c_2 /d_2/e_2, c_1/e_1)$ in *MDP*'.
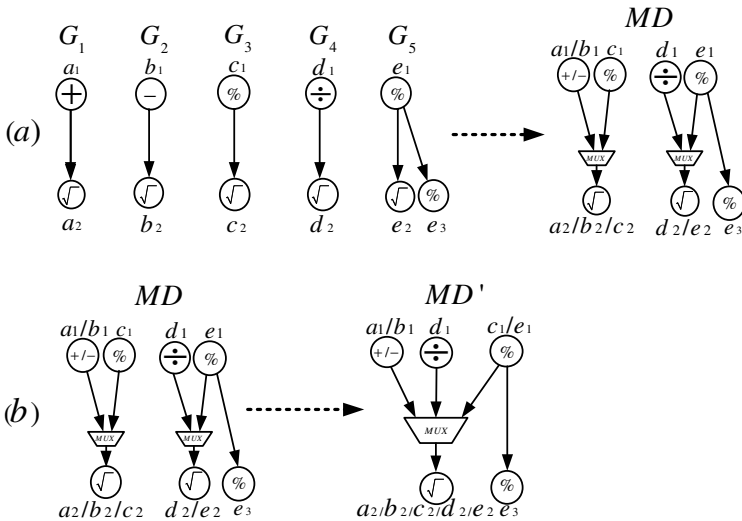

Fig. 3. (a) merging five DFGs G_1,…, G_5 in steps to create merged datapath *MDP*, (b) sharing the resources inside *MDP* to create *MDP'*

We merge the resources inside *MDP* in three stages. In the first stage all merging possibility among the resources inside *MDP* are to be considered as a compatibility graph. To do this, we employed the compatibility graph in [13] but use different approach to reduce the configuration time. Then in the second stage, the maximum weighted clique in this graph " M_c " is found. By searching M_c in the compatibility graph and reconstructing the *MDP* using this clique, *MDP*' is created.

To combine the resources inside the merged datapath, the merging possibilities among the vertices and the merging possibilities among the edges in the merged datapath *MDP* should be taken into consideration. The compatibility graph shows the merging possibility among the vertices inside *MDP* to create the same type of vertex in *MDP*' or, the merging possibility among the edges inside *MDP* to create the same type of edge in *MDP*'. Below, the compatibility graph G_c for the input merged datapath *MDP* is defined formally.

A compatibility graph corresponding to input merged datapath, *MDP* is an undirected weighted graph $G_c=(N_c,A_c)$ where:

- Each Node $n_c \in N_c$ with weight w_c corresponds to:
  — Vertex-merging that is a possible merging of the vertices $v_i,v_j \ldots v_k \in MDP$ to create a vertex $v_{i..k} \in MDP'$ where it does not merge the same vertex from a DFG G_x onto different vertices from a DFG G_y or vice-versa.
  — Edge-merging that is a possible merging of edges $e_i=(u_i,v_i,p_i)$, $e_j=(u_j,v_j,p_j)$ … $e_k=(u_k,v_k,p_k)\in MDP$ to create an edge $e_{i.k}\in MDP'$ where it does not merge the same vertex from a DFG G_x onto a different vertices from a DFG G_y or vice-versa.
- Each arc $a_c=(n_c,m_c)\in G_c$ illustrates that its nodes n_c and m_c (merging nodes) are compatible. It means they do not merge the same vertex from a DFG G_x onto different vertices from a DFG G_y or vice-versa.
- Each node's weight, w_c, represents the reduction in configuration time resulting from merging the vertices or merging the edges.

By merging vertices $v_i,v_j..v_k \in MDP$ together, a vertex v' is created in *MDP*'. Besides, for each input port of v' which has more than one incoming edge, *MDP*' will have a multiplexer to select the input operand. That is the reason to add multiplexer to the input ports of v', or add an input to previous multiplexers (if v_i has a multiplexer).

In this case, configuration time reduction of the $n_c \in G_c$ is equal to the difference between the configuration time of the hardware units and multiplexers before merging vertices and, their configuration time after merging, that is:

$$w_i = (T_f(v_i) + T_f(v_j)) - (T_f(v') + m \times T(mux_i)) \tag{1}$$

$T_f(v_i)$ and $T_f(v_j)$ in equation (1) are the hardware units configuration time before the merging and ($T_f(v')$ is the hardware units configuration time after the merging. Furthermore, $m \times T(mux_i)$ is used to show the increase due to the multiplexer configuration time. If the multiplexer has the same number of inputs as it had before, then $m=0$. Otherwise, m shows the increase in the size of the multiplexer (for example going from 4 ports multiplexer to 8 port multiplexer, m=1).

By merging edges $e_i,e_j..e_k \in MDP$, for each input port of v', one incoming edge is created in *MDP*'. Hence, there will be no multiplexer creation or there will be no change in the number of multiplexer inputs. Configuration time reduction achieved by

this type of merging corresponds to equation (2) that is the weight of removing the multiplexers, or decreasing the size of the multiplexers.

$$w_i = m \times T\,(mux_i) \tag{2}$$

The nodes in the compatibility graph have signed integer weights. Because, although some vertex-merging have negative weights (i.e., increase in the configuration time), their resultant weight with edge-merging has positive weight (i.e., reduction in configuration time). Note that the edges from *MDP* are not merged unless their vertices are merged. In this way, negative vertices should be included in the compatibility graph.

Up to here, all the merging possibilities inside the *MDP'* have been presented as the compatibility graph G_c. Choosing compatible nodes with more weight in G_c in order to create *MDP'*, is equal to finding a completely connected sub graph with more weight from G_c. This graph is called clique in the graph algorithms. A **clique** C_c is a subset of nodes in a compatibility graph, $C_c \subset G_c$, such that for all the distinct nodes $u, v \in C_c$, they are connected together ($u, v \in E_c$). A clique is maximum if there are no larger cliques available in G_c. The **maximum weighted clique** M_c, is a clique in G_c that the total weight of its nodes is more than any other C_c in G_c. By employing the maximum weighted clique, *MDP'* can be created.

Calculation of the maximum weighted clique from the compatibility graph is known to be an *NP*-complete problem [15]. In order to solve the problem, Branch&Bound algorithms can be employed. They provide an appropriate problem search space in order to solve the maximum weighted clique problem. The algorithm presented in [16] chooses an efficient method to select nodes and predicts bounds for quick backtracking. Our proposed technique uses the same optimizations as indicated in [16], but the assumption to have positive weights for the nodes in the input graph have been changed to the signed integer weights for the nodes.

After finding M_c in the compatibility graph, the merging possibilities represented by the nodes in M_c is used to reconstruct *MDP* and create *MDP'*. Each node in M_c gives a merging possibility among the edges (or among the vertices) inside *MDP*. This way, edges and vertices inside the *MDP* are merged together to minimize the configuration time where, these merges do not merge the same vertex from a DFG G_x onto a different vertices from a DFG G_y.

4   Experimental Results and Analysis

To evaluate the effectiveness of the proposed technique, we made comparison with techniques that were proposed in [14] and [7]. In [14] a datapath merging technique, based on inter-DFGs resource sharing, has been proposed. It applies the HLS algorithms such as functional unit allocation, register allocation, and interconnection binding, simultaneously to the DFGs in steps. In [7] conventional HLS technique for RTR system, based on intra-DFG resource sharing, has been proposed. It applies HLS algorithms to each DFG to create its datapath. These techniques and the proposed technique in this paper were applied to five benchmarks from the Media-bench suite [17]. There are computational intensive kernels (inner loop kernels) in each benchmark that their entity makes them suitable for mapping into a reconfigurable unit in the RTR system. Each benchmark was compiled using the GCC compiler and for each loop, a DFG was generated from the loop RTL code.

For each benchmark, up to 3 kernels were considered and their corresponding DFGs were iteratively merged from the larger DFG into the smaller one. The configuration time of bit-stream in a FPGA is estimated as: (size of bit-stream) / (configuration clock frequency). After obtaining the bit-stream of a functional unit and a multiplexer by ISE 10.2, their configuration times were calculated based on their configuration bit-stream. Xilinx FPGAs support partial reconfiguration therefore; they are suitable for RTR systems. In our experiment, FPGA Virtex5-xc5vlx was employed as a targeted platform. In order to present the efficiency of our technique, the maximum configuration clock frequency of the FPGA (100 Mbps) was considered which is the worst case scenario for our method.

Table 1. Kernels configuration time T_C resulted from datapath merging algorithms in Media-bench applications

Benchmark (Number of DFGs)	T_C in [7] (ms)	T_C in [14] (ms)	T_C in Proposed Algorithm (ms)
Epic-decoder(3)	11.04	6.39	5.82
Epic-coder(3)	4.87	2.66	2.49
Mpeg2-decoder(3)	5.83	4.11	3.64
Mpeg2-coder(3)	7.51	5.68	4.87
G721(2)	10.6	6.39	5.82

To apply the proposed technique and previous datapath merging technique to DFGs, initially each DFG was scheduled using As Soon As Possible (ASAP) scheduling algorithm in advance. Later, the proposed datapath merging technique and datapath merging technique in [14] were added to the scheduled DFGs to archive their merged datapaths. To implement the conventional HLS technique in [7], each DFG was scheduled in advance and the resources inside the DFG were shared using the Integer Linear Programming (ILP) algorithm afterward to obtain the datapath. The configuration time of each datapath, and the merged datapaths were calculated based on the configuration clock frequency of the FPGA.

Table 1 illustrates the configuration time to map the kernels on reconfigurable fabric for each benchmark resulted from applying the proposed technique and the techniques in [14] and [7] to DFGs. As illustrated in Table 1, the datapath merging techniques have lower configuration time in general and the proposed technique has the least configuration time. The main reason for this is the shorter length of the generated bit-stream by the proposed technique. For G721, there are two DFGs in the benchmark and the improvement of the proposed technique and previous merging algorithm is the same.

Fig.4 shows the configuration time reduction percentage of the DFGs after applying the above mentioned-techniques to each benchmark where the FPGA Xilinx Virtex5-xc5vlx is the target platform. As illustrated in this figure, there is a substantial difference between the results of the datapath merging techniques and conventional HLS technique in [7] in terms of the configuration time reduction. The proposed datapath merging technique lowered the configuration time up to 50% in comparison to the technique in [7]. On the other hand, it can decrease the configuration time up to 10% more than the datapath merging technique in [14] for these benchmarks.
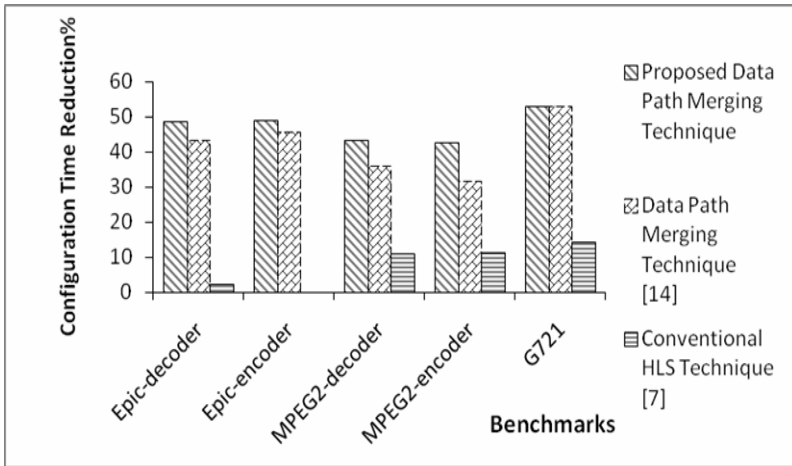
Fig. 4. Percentage configuration time reduction in datapath merging algorithms for Media-bench applications

We conclude that sharing the resources among the DFGs is the main cause of having shorter bit-stream. In the same line, merging more resources in the proposed datapath merging technique results in more reduction in configuration time compared to the previous datapath merging technique. The results lead us to the conclusion that the proposed datapath merging approach is suitable for the synthesizer in run-time reconfigurable systems.

5   Conclusion

In this paper a modified datapath merging technique for the reduction of datapath configuration time was introduced. Our proposed technique combines the resources inside the merged datapath to minimize the configuration time. To this end, we set off to find a compatibility graph that indicates to similarity between the resources inside the merged datapath. Afterwards, we determined a maximum weighted clique in the compatibility graph to reconstruct the merged datapath. This is an *NP*-complete problem thus; a Branch&Bound algorithm was employed to solve the problem. Applying the proposed technique to the workloads from the Media-bench suite in terms of the configuration time reduction, shows an improvement up to 50% in comparison to the conventional HLS algorithm in [7] and an improvement up to 10% in comparison to the previous datapath merging technique in [14]. Overall, the proposed merging technique in this paper can efficiently decrease the configuration time in RTR systems. It should be mentioned that datapath merging approach will increase the kernel execution time; however, the configuration time is measured in milliseconds while the kernel execution time is in nanoseconds. Therefore, the increase in the execution time of kernels is negligible compared to the configuration time reduction. Nonetheless, where kernels have significant number of iterations, this time overhead (increase in the execution time of the kernels) is comparable to the configuration time reduction and we should consider it in datapath merging. Furthermore, we will investigate this factor in datapath merging.

Acknowledgment

References

1. Wenyin, F., Compton, K.: An Execution Environment for Reconfigurable Computing. In: Proc. of 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), CA, USA, April 2005, pp. 149–158 (2005)
2. Li, Z.: Configuration Management Techniques for Reconfigurable Computing.: Ph.D. Thesis. Northwestern University (June 2002)
3. Vassiliadis, S., Wong, S., Gaydadjiev, G.N., Bertels, K.L.M., Kuzmanov, G.K., Panainte, E.M.: The Molen Polymorphic Processor. IEEE Transactions on Computers (TC) 53(11), 1363–1375 (2004)
4. Ghiasi, S., Nahapetian, A., Sarrafzadeh, M.: An Optimal Algorithm for Minimizing Runtime Reconfiguration Delay. ACM Transactions on Embedded Computing Systems (TECS) 3(2), 237–256 (2004)
5. Rollmann, M., Merker, R.: A Cost Model for Partial Dynamic Reconfiguration. In: Proc. of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Greece, July 2008, pp. 182–186 (2008)
6. Farshadjam, F., Dehghan, M., Fathy, M., Ahmadi, M.: A new compression based approach for reconfiguration overhead reduction in Virtex-based RTR systems. Elsevier Journal on Computers &Electrical Engineering 32(4), 322–347 (2006)
7. Qu, Y., Tiensyrj, K., Soininen, J.P., Nurmi, J.: Design Flow Instantiation for Run-Time Reconfigurable Systems. EURASIP Journal on Embedded Systems (TECS) 2(11), 1–9 (2008)
8. Yankova, Y.D., Kuzmanov, G.K., Bertels, K.L.M., Gaydadjiev, G.N., Lu, Y., Vassiliadis, S.: DWARV: DelftWorkbench Automated Reconfigurable VHDL Generator. In: Proc. of 17th International Conference on Field Programmable Logic and Applications (FPL), Amsterdam, The Netherlands, August 2007, pp. 697–701 (2007)
9. Meeuws, R.J., Yankova, Y.D., Bertels, K.L.M., Gaydadjiev, G.N., Vassiliadis, S.: A Quantitative Prediction Model for Hardware/Software Partitioning. In: Proc. of 17th International Conference on Field Programmable Logic and Applications (FPL), Amsterdam, The Netherlands, August 2007, pp. 735–739 (2007)
10. Coussy, P., Morawiec, A.: High-Level Synthesis from Algorithm to Digital Circuit. Springer, Heidelberg (2008)
11. Chiou, L., Bhunia, S., Roy, K.: Synthesis of Application-Specific Highly Efficient Multimode Cores for Embedded Systems. ACM Transaction on Embedded System Computing (TECS) 4(1), 168–188 (2005)
12. Moreano, N., Borin, E., Souza, C.D., Araujo, G.: Efficient Datapath Merging for Partially Reconfigurable Architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems (TCAD) 24(7), 969–980 (2005)
13. Fazlali, M., Fallah, K.F., Zolghadr, M., Zakerolhosseini, A.: A New Datapath Merging Method for Reconfigurable System. In: Proc. of 5th International Workshop on Applied Reconfigurable Computing (ARC), Karlsruhe Germany, March 2009, pp. 157–168 (2009)
14. Fazlali, M., Zakerolhosseini, A., Sabeghi, M., Bertels, K.L.M., Gaydadjiev, G.: Datapath Configuration Time Reduction for Run-time Reconfigurable Systems. In: Proc. of International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), Las Vegas Nevada, USA, July 2009, pp. 323–327 (2009)

15. Garey, M., Johnson, D.S.: Computers and Intractability-A Guide to the Theory of NP Completeness. Freeman, San Francisco (1979)
16. Ostergard, P.R.J.: A New Algorithm for the Maximum-Weight Clique Problem. Nordic Journal of Computing (NJC) 8(4), 424–436 (2002)
17. Lee, C., Potkonjak, M., Mangione, W.S.: Mediabench: a Tool for Evaluating and Synthesizing Multimedia and Communication Systems. In: Proc. of 13th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), California, USA, December 1997, pp. 330–335 (1997)