Data path Configuration Time Reduction for Run-time Reconfigurable Systems

M. Fazlali[1], A. Zakerolhosseini[1], M. Sabeghi[2], K. Bertels [2] and G. Gaydadjiev [2]

[1] Department of Computer Engineering, Shahid Beheshti University G.C, Tehran, Iran
[2] Computer Engineering Laboratory., Delft University of Technology, Delft, The Netherlands

Abstract - *The FPGA (re)configuration is a time-consuming process and a bottleneck in FPGA-based Run-Time Reconfigurable (RTR) systems. In this paper, we present a High Level Synthesis (HLS) method, based on the data path merging technique to amortize the hardware configuration time in RTR systems. It merges the Data Flow Graphs (DFGs) of two or more computational intensive parts of the application and makes one general purpose data path (merged data path) which results in shorter bit-stream length and therefore reduces the configuration time. Our experimental results using the proposed method on media-bench applications, show up to 40% reduction in the configuration time compared to conventional synthesis method.*

Keywords: FPGA, Run-time Reconfigurable Systems, High Level Synthesis (HLS).

1. Introduction

Many applications contain computational intensive parts which can be implemented in hardware to increase the performance. In this paper, we refer to each of these parts as a kernel and a group of them as a "module". A reconfigurable system can accelerate such modules by executing them on the reconfigurable fabric. However, the FPGA resources are constrained and applications may have several modules. Therefore, RTR system should be able to share the FPGA resources by (re)configuration of the hardware whenever it is necessary [1]. Nonetheless, the run-time reconfiguration imposes a considerable overhead to the performance of the system and the reconfiguration should be done as fast and efficient as possible to prevent the benefit gained by hardware acceleration to be eclipsed by the overhead of the configuration [2].

The bit-stream length and the configuration time of the hardware into FPGA are directly proportional [3]. In fact, the most of the configuration time is the time to transmit the bit-stream into FPGA and therefore reducing the bit-stream length reduces the configuration time [4].

Several researches have been carried out to reduce the configuration time and improve the performance of the RTR systems. The works presented in [5, 6] change the execution order of the kernels to reduce the number of configurations. Authors in [7] present a temporal algorithm for partitioning and scheduling the DFGs of the applications. They have attempted to increase similarity of subsequent configurations in such a way that the configuration time decreases. The COMMA methodology for dynamic reconfiguration has been described in [8] which is mainly about the generation of a communication infrastructure that is supported by a sequence of dynamically-placed modules with the aim of minimizing the configuration time. An algorithm which solves the wire delay estimation and merging sub-problems for minimizing configuration time is also presented in [9].

Another direction in research elaborates on the physical configuration of the FPGA. The FPGA configuration time is amortized by reducing the size of the bit-stream. Some compression techniques have been employed in [10, 11] to shorten the bit-stream length. Authors in [12] used caching technique for the reduction of the configurations time. [13] removes a piece of configuration bit-stream on Virtex FPGA and replaces it with another piece to create a new configuration and reduce the overhead of placement.

Although these techniques are suitable for reducing the configuration time of RTR systems and new generation of the FPGAs supports some of these features, they are usually costly. For instance, compressing bit-stream to reduce the transmission time has additional time-overhead for decompressing the bit-stream. We can reduce configuration time in off-line stages of creating hardware. Therefore, we can use high level synthesis technique for the reduction of configuration time [14].

Data path merging is a high level synthesis method which has been presented to reduce the resource area usage for partially reconfigurable system [15]. In this paper, we apply the same method based on datapath merging but use different approach to reduce the configuration time. It means that the main contribution of this paper is to present a new synthesis approach, based on the data path merging technique for reducing the bit-stream length and consequently reducing

the configuration time. In fact, merging multiple data paths into a larger multipurpose data path will reduce the number of resources we need and results in a shorter bit-stream length.

The organization of this paper is as follows. In the next section, the basic idea of data path merging is explained and the impact of data path merging on the module configuration time is described. Section 3 presents the proposed method for configuration time reduction followed by the experimental results in section 4. Ultimately, chapter 5 concludes this paper.

2. Reducing the Configuration Time Using Data path Merging

High-level synthesis methods aim of exploiting the intra-DFG resource sharing to reduce the hardware cost [14]. On the other hand, Data path merging takes advantage of inter-DFGs resource sharing for the same purpose. In [16] a novel data path merging method is presented for the area reduction in partially reconfigurable system. In this paper, we apply the same method based on datapath merging but use different approach to reduce the configuration time in the FPGA-based RTR systems. The proposed method is explained as follows.

Let a DFG be a directed graph $G=(V,E)$, where $V=\{v_1,v_2,\dots v_n\}$ is the set of vertices and $E=\{e_1=(u_1,v_1,p_1),\dots e_n=(u_n,v_n,p_n)\}$ is the set of edges. A vertex $v_i \in V$ represents an operation that can be performed with a functional unit while each v_i has a set of input ports p_i. An edge $e_i=(u_i,v_i,p_i) \in$E indicates a data transfer from vertex u_i to the input port p_i of vertex v_i.

A data path $D=(V',E')$ is a directed graph, where $V'=\{v'_1,v'_2,\dots v'_n\}$ is the set of vertices and $E'=\{e'_1=(u'_1,v'_1,p'_1),\dots e'_n=(u'_n,v'_n,p'_n)\}$ is the set of edges. A vertex $v' \in V'$ represents a merge of vertices v_i from G_i and an edge, and $e'=(u',v',p') \in E'$ represents a data transfer from vertex u' to the input port p' of vertex v'.

After applying intra-DFG resource sharing possibilities to a DFG G_i, the data path D is generated for it. This way, some multiplexers are added into the input ports of the vertices in D. Data path configuration time T_{cc}, is the required time for configuring the data path into the FPGA. Considering a data path $D=(V',E')$, the data path configuration time is:

$$T_{cc} = T_f + T_i \qquad (1)$$

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 multiplexers configuration time. T_f (v) is the configuration time of a functional unit allocated to v, and T_i (MUX) represents the configuration time of a multiplexer used in the input port of each vertex.

The module configuration time T_C, is the aggregate sum of the data paths configuration time for the DFGs corresponding to all the kernels in the module.

$$T_C = \sum T_{cc} \qquad (2)$$

A merged data path, $MDP=(V'',E'')$, corresponding to DFGs G_i, i=1…n is a directed graph, where: a vertex $v'' \in V''$ represents a merging of vertices v_j from various G_i. An edge $e''=(u'',v'',p'') \in E''$ represents a merging of edges, $e_j=(u_j,v_j,p_j)$, each one from a different G_i, in such a way that all u_j have been mapped onto u'' and all v_j have been mapped onto v'' and corresponding input ports p_i have been matched together and mapped to p''.

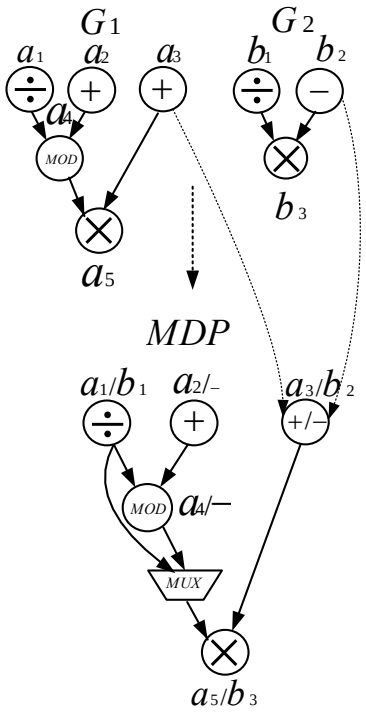

Fig.1: The merged data paths *MDP* for DFGs G_1 and G_2 [16].

Fig.1 illustrates an example of data path merging where DFGs G_1 and G_2 from this figure are merged and the merged data path *MDP* is made. Considering these DFGs, if operation of a vertex from G_1 and operation of a vertex from G_2 can be performed with the same 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 a functional unit. Thus, these vertices are merged together and the vertex (a_1/b_1) is made for them in *MDP*. If a vertex cannot be merged onto other vertices, it will remain in the merged data path without any modification. After merging two vertices, multiplexers are employed in the input ports of their corresponding vertex in the merged data path to

select the input operand. This is illustrated in the input ports of vertex (a_5/ b_3) in Fig.1.

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 made instead in the resulting *MDP*. In this case, there is no need for any multiplexer in the input ports of the vertex (a_5 / b_3) to select the input operands.

If we use a data path merging as HLS tool, the module configuration time (T_C) is equal to the merged data path configuration time and our ultimate goal here is to reduce this configuration time.

3. The Proposed Synthesis Method

To merge DFGs, we need to merge the hardware units and the interconnection units simultaneously. If we merge vertices without considering the interconnections or using only estimates for the interconnections, the resulting merged data path's configuration time will not be optimized. To do this, we use the graph-based technique presented in [16]. It merges DFGs in steps to compute the merged data path. At each step, one DFG is merged onto the merged data path. To merge, we should find the similarities between DFG resources and the resources from the merged data path. To do that, we use the concept of compatibility graph.

A compatibility graph $G_c=(N_c,A_c)$ corresponding to the merged data path *MDP* and a DFG G_j, is an undirected weighted graph where:

- Each node $n_c \in N_c$ represents a merging between vertices, or edges. It corresponds to merging vertices $v_j \in G_j$ and $v_i \in MDP$ (to create the vertex (v'') in the next merged data path) or, merging edges $e_j=(u_j,v_j,p_j) \in G_j$ and $e_i=(u_i,v_i,p_i) \in MDP$ (to create the edge $e''_i=(u''_i,v''_i,p''_i)$ instead). For merging edges, their corresponding vertices should be merged together.
- Each arc $a_c=(n_c,m_c)$ illustrates that its nodes n_c and m_c do not merge the same vertex from G_j, onto two different vertices from *MDP* or vice-versa.
- Each node's weight w_c represents the reduction in configuration time obtained by merging.

After merging $v_i \in MDP$ onto $v_j \in G_j$, a vertex v'' is made instead of them in the next merged data path. Moreover, for each input port of v'' which has multiple incoming edge, a multiplexer is needed to select the input operands. 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:

$$w_i = (T_f(v_i) + T_f(v_j)) - (T_f(v'') + m \times T(mux_i)) \qquad (3)$$

T_f (v_i) and T_f (v_j) in equation (3) 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 has before, the merging then $m=0$. Otherwise, m shows the increase in the size of multiplexer (for example going from 4 ports multiplexer to 8 port multiplexer, m=1).

To merge edges $(u_j,v_j,p_j) \in G_i$ and $(u_i,v_i,p_i) \in MDP$, each input port of v'', has just one incoming edge. So, it does not increase in the size of the multiplexer and therefore, the configuration time reduction achieved by this type of merging corresponds to equation (4) that is the weight of removing the multiplexers, or decreasing the size of the multiplexers.

$$w_i = m \times T(mux_i) \qquad (4)$$

For merging, we should find a number of compatible nodes from the compatibility graph that provides the maximum reduction in configuration time. This overall reduction is equal to the aggregated weights of all the compatible nodes in G_c. Choosing compatible nodes from the compatibility graph $G_c=(N_c,A_c)$, is equal to finding a completely connected sub graph in G_c, which is called a clique in graph theory. A clique is called maximal clique, if there are no larger cliques in G_c. The maximum weighted clique M_c is a maximal clique in G_c that total weight of its nodes is larger than any other maximal clique in compatibility graph. By using the maximum weighted clique, the desired merged data path is made.

Listing 1: Merging algorithm to make the merged data path

```
Program DPM (Input: DFGs G_i=(V_i,E_i) i=1…n,
Output: merged data path MDP=(V,E))
  {assuming G_i i=1…n are sorted};
Begin

 MDP<-G_1;
 for i<-2 to n do
   G_c<-MakeCompatibilityGraph(MDP,G_i);
   M_c<-MaximumWeightedClique(G_c);
   MDP<-reconstructMergedDatapath(M_c,MDP,G_i);
 Endfor

End DPM.
```

Finding the maximum weighted clique is a well-known problem in the graph algorithms. It is known to be an *NP*-Hard problem [17]. Several optimized and fast Branch &Bounds solution for the problem of finding maximum weighted clique have been presented in literature [18, 19].

The function *MaximumWeightedCliqueClique*() use the method in [19] to find the desired maximal weighted clique.

After finding the maximal weighted clique in the compatibility graph, the merging possibility represented by the nodes of this clique is used to reconstruct the merged data path. Each node from the clique indicates a merging possibility between an edge (a vertex) from *MDP* and an edge (a vertex) from G_j. Other vertices and edges which cannot be merged are added to *MDP* without any merging.

To merge the DFGs in the module, the algorithm merges DFGs in steps considering a sequence from the biggest DFG to the smallest one. The pseudo code of the proposed method is shown in Listing 1. This algorithm considers the biggest DFG as a merged data path *MDP* then starts adding other DFGs to it in steps. It creates the compatibility graph in each step then it finds the maximum weighted clique M_c in compatibility graph. Finally it reconstructs the merged data path. These steps are repeated for all DFGs in a Loop for *N* number of DFGs until the merged datapath is made.

4. Experimental Results

Our proposed technique is general (and technology independent). It can reduce the kernels configuration time for FPGAs that support partial reconfiguration, (such as Xilinx FPGAs) and can be used in run-time reconfiguration systems. To validate this claim, we have implemented the proposed technique using FPGA Virtex5-xc5vlx. Enough experimental evidences exist to support the fact that there are some computational intensive kernels in each application of Media-bench which have the largest share of execution time [20]. In our experiments, each application was compiled using the GCC version compiler, and was profiled so as to determine which kernels contributed the most to the application execution time. For each such a kernel, a DFG was generated.

The configuration time of a bit-stream in a FPGA is equal to [(size of bit-stream) / (configuration clock frequency)] [3]. After obtaining the bit-stream of the functional units and multiplexers by ISE 10.2, their configuration times were obtained. In our experiments, we considered the maximum configuration clock frequency 100 *Mbps* in case of the FPGA Virtex5-xc5vlx.

We applied the proposed data path merging synthesis technique and the technique in [14] to several DFGs (corresponding to the computational kernels of the applications from Media-bench). In these experiments, the Epic-Decoder module, Epic-Encoder module, Mpeg2-Decoder module, and Mpeg2-Eecoder module includes three DFGs and, G721module includes two DFGs.

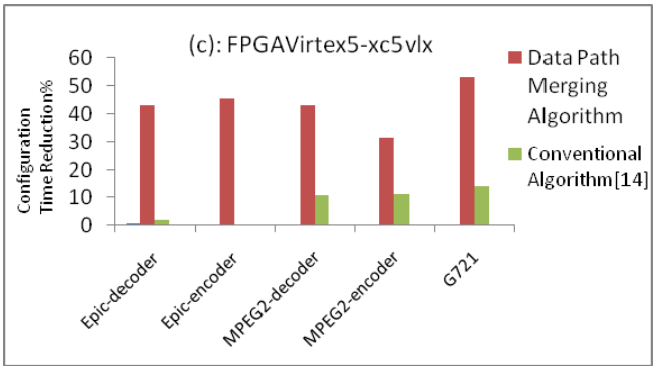

Fig.2: The configuration time reduction percentage for the proposed data path merging algorithm and the algorithm in [14].

Fig.2 shows the percentage of the reduction in module configuration time for each benchmark after Appling the proposed method and the conventional synthesis method in [14] to the DFGs for each benchmark. As illustrated in this figure, the conventional technique in [14] cannot reduce the configuration time for epic-encoder benchmark however; the proposed data path merging method can reduce its configuration time efficiently. The proposed approach can reduce the data path configuration time up to 40% in comparison to the synthesis technique in [14]. The main reason for this is the shorter length of the generated bit-stream by the proposed technique that causes to shorter configuration time. It should be mentioned that our approach will slow down the hardware execution. However, in RTR systems we configure the hardware so many times. While the configuration time unit is in milliseconds scale and, hardware execution time is in nanoseconds scale. Therefore, until hardware has not so many iterations, this overhead compared to the configuration time reduction is negligible.

We observed that data path merging is a suitable method for the configuration time reduction of the kernels mapped on FPGA. But it introduces additional multiplexers when the operations in DFGs have different operands. The merge technique, on the other hand, maps input and output operands, as well as operations, and permits us to map exclusive operations which may have different operands, if it results in the final data path configuration time reduction. In order to do this, it analyzes hardware units and interconnection mapping together. The results strongly indicate that the data path merging technique indeed provides configuration time reduction, when compared to the conventional approach.

5. Conclusions

This paper presented a new synthesis method to reduce the configuration time of a data path in FPGA-based RTR system. Due to large cost of mapping of the modules on the FPGA, the proposed synthesizer merges the DFGs of the module onto a single merged data path to amortize the configuration time.

Sharing the recourses between the DFGs is the main cause of reducing the bit-stream length and shorter bit-stream means shorter configuration time. We performed the experiments for FPGA Virtex5-xc5vlx to evaluate the efficiency of the proposed method. We applied the proposed synthesizer on the Media-bench applications' kernels. The obtained results confirmed that the module configuration time is lowered up to 40% for these benchmarks compared to conventional synthesis method.

ACKNOWLEDGEMENTS

This research was supported by the Iran Telecommunication Research Center (ITRC) in the context of the project T/500/3462. It partially supported by the Hartes project EU-IST-035143, the Morpheus project EU-IST-027342 and the Rcosy Progress project DES-6392

6. References

[1] K. Compton, S. Hauck ,"Reconfigurable Computing: A Survey of Systems and Software, " In ACM Computing Surveys (CSUR),vol (34),no (2),pp.171-210, 2002.

[2] Z. Li, "Configuration Management Techniques for Reconfigurable Computing," Ph.D. Thesis, Northwestern University, June 2002.

[3] M. Rollmann and R Merker," A Cost Model for Partial Dynamic Reconfiguration, " In proc. of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2008), pp.182-186,Greece, July 2008.

[4] Z. Huang and S. Malik , " Managing dynamic reconfiguration overhead in systems-on-a-chip design using merged data paths and optimized interconnection networks," In Proc. Design Automation and Test in Europe (DATE 2001), pp. 735–740, Munich, Germany, March 2001.

[5] S.Ghiasi, Majid. Sarrafzadeh, "Optimal Reconfiguration Sequence Management," In Proc. IEEE/ACM Asia South Pacific Design Automation Conference (ASP-DAC), pp. 359–365, Kitakyushu, Japan, January 2003.

[6] S. Ghiasi, A Nahapetian and M Sarrafzadeh, "An Optimal Algorithm for Minimizing Run-time Reconfiguration Delay," ACM Transactions on Embedded Computing Systems (TECS), Vol. 3, No 2, pp. 237-256, May 2004.

[7] F. Mehdipour, M. Sahebzamani, H. R. Ahmadifar, M. Sedighi and K. Murakami, "Reducing Reconfiguration Time of Reconfigurable Computing Systems in Integrated Temporal Partitioning and Physical Design Framework," In Proc. 20th IEEE International Parallel&Distributed Processing Symposium.(IPDPS 2006), pp. 25–29, Greece, April 2006.

[8] S. Koh and O. Diessel, " Communications Infrastructure Generation for Modular FPGA Reconfiguration," In Proc. IEEE Int. Conf on Field-Programmable Technology (FPT06), pp. 321–324, Bangkok, Thailand, December, 2006.

[9] S. Koh and O. Diessel, " Module Graph Merging and Placement to Reduce Reconfiguration Overheads in Paged FPGA Devices," In Proc. Int. Conference on Field Programmable Logic and Applications. (FPL 2007), pp.293–298, Amsterdam, Netherland, August. 2007.

[10] P. J. Hwa, T. Mitra and W.F. Wong, "Configuration Bit-stream Compression for Dynamically Reconfigurable FPGAs," In Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD-2004) , pp. 766–773, CA, USA, November, 2004.

[11] F. Farshadjam, M. Dehghanb, M. Fathy and M. Ahmadi, " A New Compression Based Approach for Reconfiguration Overhead Reduction in Virtex-Based RTR Systems," Elsevier journal on Computers &Electrical Engineering, Vol. 32 , No. 4, pp 322–347, 2006.

[12] K. Compton and S. Hauck, " Reconfigurable Computing: A Survey of Systems and Software," ACM Computing. Surveys, Vol. 34, No.2, pp. 171–210, 2002.

[13] S. Raaijmakers and S. Wong, "Run-Time Partial Reconfiguration for Removal, Placement and Routing on the Virtex-II Pro," In Proc. Int Conference on Field Programmable Logic and Applications (FPL 2007, pp. 679–683, Amsterdam, Netherland, August, 2007.

[14] Y. Qu, K. Tiensyrj, J.P. Soininen and J.Nurmi, " Design Flow Instantiation for Run-Time Reconfigurable Systems," EURASIP Journal on Embedded Systems (TECS), Vol. 2. No.11, pp. 1–9, 2008.

[15] C. de. Souza, A. M. Lima, N. Moreano and G. Araujo, " The Data path Merging Problem in Reconfigurable Systems: Lower Bounds and Heuristic Evaluation," ACM Journal of Experimental Algorithms (JEA), Vol. 10, No.2, pp 1–19, 2005.

[16] N. Moreano, Ed. Borin, C. d. Souza and G. Araujo, " Efficient Data path Merging for Partially Reconfigurable architectures," IEEE Transaction on Computer Aided Design, Vol. 24. No.7, pp. 969–980, 2005.

[17] M.Garey, and D. S. Johnson , "Computers and Intractability-A Guide to The Theory of NP-Completeness," San Francisco, CA, Freeman, 1979.

[18] Patric R. J. Ostergard. "A New Algorithm for the Maximum-Weight Clique Problem," Nordic Journal of Computing (NJC), Vol. 8, No. 4, pp.424–436, 2002.

[19] D. Kumlander, "A New Exact Algorithm for The Maximum-Weight Clique Problem Based on a Heuristic Vertex-Coloring And a Backtrack Search," In Proc. Fourth European Congress of Mathematics(4ECM),pp.31–39, 27June–2July, Stockholm, Sweden, 2001.

[20] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Media-bench: a Tool for Evaluating And Synthesizing Multimedia And Communication Systems," In Proc. Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-03, pp. 330–335, CA, USA, December, 1997.