International Journal of Electronics and Communication Engineering (IJECE)
ISSN 2278-9901
Vol. 2, Issue 2, May 2013, 65-76
© IASET

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

FPGA IMPLEMENTATION OF RECONFIGURABLE ARCHITECTURE FOR LOGICAL APPLICATIONS

R. PHANI VIDYADHAR[1] & J. SELVA KUMAR[2]

[1]PG Student, Department of ECE, Chennai, India

[2]Assistant Professor, Department of ECE, Chennai, India

ABSTRACT

Coarse-Grained Reconfigurable Architecture (CGRAs) requires many Processing Elements (PEs) and a configuration cache memory unit for reconfiguration of its PE array. This structure is meant for high performance and flexibility, it consumes significant power. The applications of Field Programmable Gate Arrays (FPGAs) are multi fold in real-time systems. They have several advantages over Application Specific Integrated Circuits (ASICs), but CGRAs applications have been restricted to integer arithmetic, since existing CGRAs supports only integer arithmetic or logical applications. In this work proposed here main objective is to design existing 4 x 4 Processing Elements (PEs) array for integer arithmetic. The main idea of this paper is to explore the advantages of FPGA in real world by mapping applications that supports integer arithmetic and the mapping can be done by using the Fast Heuristic algorithm to get the required results. The focus is to do synthesis of both existing 4 x 4 PE array design and modified 4 x 4 PE array design for speed, power and delay using Xilinx and Xpower analysis tool. This design uses HDL, Modelsim simulator and Xilinx9.1i Synthesizer targeted on Vertex platform. Heuristic approach using Quantum- inspired Evolutionary Algorithm (QEA) used here supports for integer arithmetic applications. The proposed Modified Processing Elements proves to be 20 - 25% reduction in delay and power dissipation, when compared to the existing PEs of 4 x 4 elements. The proposed PEs might lead a significant reduction in power and delay when used in multimedia application, with maximum throughput.

KEYWORDS: Coarse-Grained Reconfigurable Architecture, Heuristic Approach, Processing Elements, Modified Quantum-Inspired Evolutionary Algorithm (QEA)

INTRODUCTION

General

Coarse Grained Reconfigurable Architectures (CGRAs) have increasingly gained the attention of Academia and industry in recent years. In comparison to Field-Programmable Gate Arrays (FPGAs), they reduce area, power, and configuration time. They also offer a more predictable timing, less configuration storage space, and an easier integration with a processor. These advantages are obtained at the cost of postproduction flexibility, and thus these architectures must be targeted earlier at design phase toward a set of applications.

Brief Description

A coarse-grained reconfigurable architecture has come into the spotlight with enormous increase in demand of several kinds of high performance applications in mobile devices because of their flexibility and performances [1]-[3]. However, existing coarse-grained reconfigurable architectures have serious limitations in handling kernels that have control intensive characteristics. It is because of the fact that the Processing Elements (PEs) in a typical reconfigurable array execute the operations passively as configured by the configuration controller.

They cannot control the execution according to the computation results. In addition, since the PEs are tightly coupled with other PEs in terms of configuration, the execution in each PE cannot be controlled independently.

The coarse grained reconfigurable architecture has the potential to change the way radically the embedded systems are built. To utilize the potential preferably, the efficient mapping and developing environment for coarse grained reconfigurable architecture is needed imperatively. Consequently, the coarse grained reconfigurable architecture has more restrictions when mapping an algorithm on it. In view of that the coarse grained reconfigurable architecture has limited Functional Units (FU), the amount of FUs and connections among them must be considered when mapping the advanced arithmetic operations.

The success of CGRAs critically hinges on how efficiently the applications are mapped onto them. For the efficient use of CGRAs, it is required to exploit the parallelism in the applications and minimize the number of computation resources since it is directly translated into either reduced power consumption or an increased throughput. However, the problem of mapping an application onto a CGRA to minimize the number of resources has been shown to be NP-complete, and therefore several heuristics have been proposed. However, existing heuristics do not consider the details of the CGRA such as the following.

PE Interconnection: Most existing CGRA compilers assume that the PEs(Processing Elements) in the CGRA are connected only in a simple 2-D mesh structure, i.e., a PE is connected to it's neighbors only. However, in most existing CGRAs, each PE is connected have more PEs than just the neighbors. In Morphosys, a PE is connected to every other PEs with the help of shared buses. A PE in RSPA also having the connections with the immediate neighbors and the next neighbors.

Shared Resources: Most CGRA compilers assume that all PEs are similar in the sense that an operation can be mapped to any PE operations. However, not all PEs in the current CGRAs contain multipliers in order to reduce the cost, power and complexity. Few multipliers and memory units are made available as shared resources in each row. For example, the PEs in each row of RSPA share two multipliers, two load units, and one store unit.

Routing PE: Most CGRA compilers cannot use a PE just for a routing. This implies that in a 4X 4 CGRA which has a mesh interconnection, it is not possible to mapping application DAGs in which any node has more than 4 degrees. However, most CGRAs allow a PE to be used for routing only, which makes it possible to mapping DAGs(Data Acyclic Graph) with any degrees onto the CGRA. Owing to the simplistic model of CGRA in the compilers, existing CGRA compilers are a) unable to mapping applications onto the CGRA, even though it is possible to map them, b) using too many PEs in their solution. Thus, even if a mapping exists for an application, it consumes significant power on CGRAs. The mapping techniques can be broadly categorized into the temporal mapping in which the functionality of PEs changes with time, and the spatial mapping in which an instruction which in a kernel loop corresponds to one PE and does not change during the execution of the kernel. Although many temporal mapping techniques can efficiently map loop structures using modulo scheduler. The CGRAs which support the spatial mapping do not need the context memory and therefore it has the advantages of smaller chip area, more power efficiency, and simpler hardware design as compared to the CGRAs with the temporal mapping. In an attempt to develop better heuristics for mapping applications to CGRAs, we observe that the problem of application mapping is very similar to that of graph drawing.

EXISTING ARCHITECTURE

As shown in Figure 1, our CGRA consists of a RISC processor, a main memory block, a DMA controller, and a coarse-grained Reconfigurable Array Architecture (RAA) connected to the shared bus as an attached IP core. The communication is through bus which couples the RISC processor and the DMA controller as master devices and the RCM as a slave device. The RISC processor executes control intensive irregular code segments and the RCM executes data-intensive kernel code segments [4].

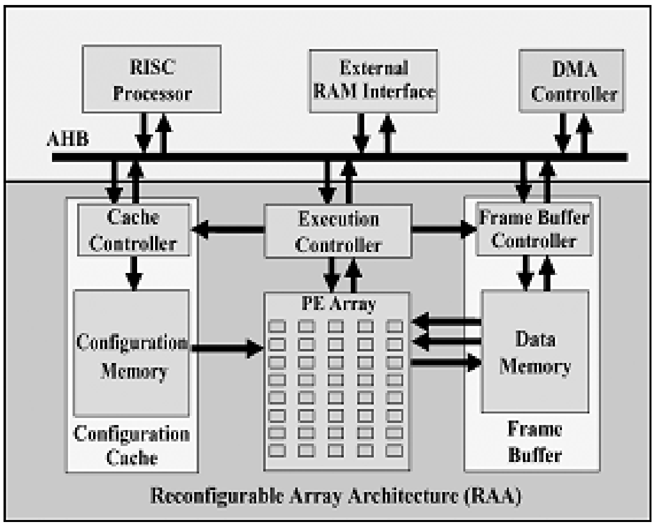


Figure 1: Basic Block Diagram of CGRA

PE Array

Figure 2 shows the current implementation of the PE array. It contains an reconfigurable array of PEs, which is sufficient for running several of the applications. Each PE is a dynamically reconfigurable unit which can perform arithmetic and logical operations. It is connected with the nearest neighboring PEs-top, bottom, left and right. Along with these connections, it is also connected in a hopping and pair-wise manner with the PEs in the same column. As shown in Figure 2, we put only one column of area critical resources such as multipliers and dividers, each of which is shared by all PE's in the same row. This technique is used to decreases the area cost significantly.

Frame Buffer

Frame buffer consists of two sets of memory each of which consists of three banks: one connected to the write bus and the other two connected to the read buses. Any combination of one-to-one mapping between these banks and the buses is possible.

Configuration Cache

Our configuration cache is composed of spatial Cache Elements (CE's), temporal CE's, and a cache controller for controlling each of the CE. Each spatial CE has layers and temporal CE has layers so that each PE can be reconfigured independently with the help of different contexts. Context register between a PE and CE is used to prohibit the path from being a critical path. The hybrid structure of configuration cache combining spatial cache and temporal cache helps **to** reducing power consumption without degrading performance.

EXTENSION OF PE

Figure 3 shows the mapping a complex operation on the PE array may take many layers of cache. If a kernel consists of a multitude of complex operations, then mapping can be done by the array easily runs out of the cache layers, causing costly fetch of additional context words from main memory. Instead of using multiple cache layers to perform such a complex operation, we can add some control logic to the PEs so that the operation can be completed in multiple cycles but without requiring multiple cache layers. The control logic can be implemented with a small Finite State Machine (FSM) that controls the PE's existing datapath for a fixed number of cycles. Normally, each PE fetches a new context word from the configuration cache every cycle to execute the operation corresponding to the context word. However, as the fetched context word is for a multi-cycle operation, the control is passed over to the FSM. A context word for such multi-cycle operation contains information about how long the operation will be executed, so that the cache control unit can wait until the current multicycle operation is finished. During the multi-cycle operation, cache control unit does not send the next context words to the PEs but resumes sending the context words right after the multi-cycle operation is terminated. Although this complicated control together with the added FSM leads to a slight increase in the area of the PE, this approach is more effective than increasing the number of cache layers and the context word width.

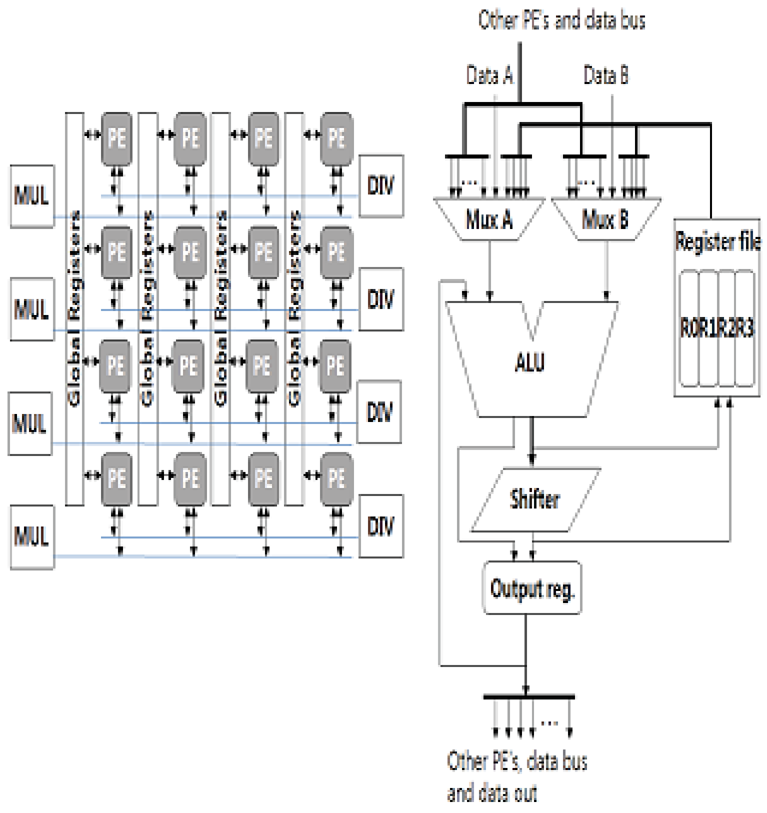


Figure 2: PE Array with Shared Logic and Internal Structure of PE

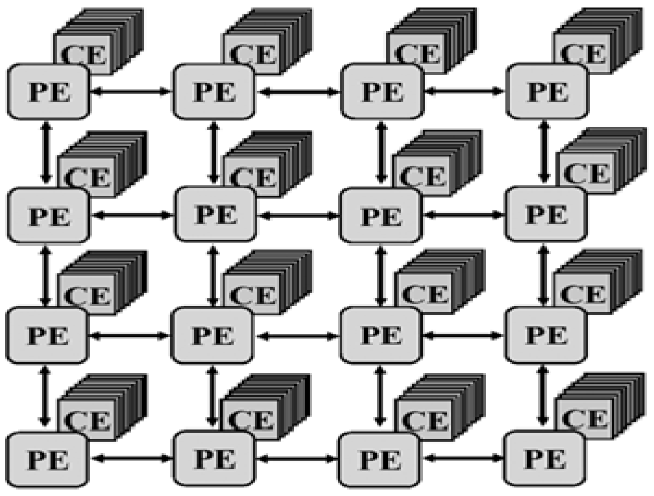


Figure 3: Experimental Setup for Existing 4 X 4 PEs Array

MODIFIED DESIGN FLOW FOR CGRA APPLICATION MAPPING

In this section, figure 4 explains about the proposed mapping algorithm for coarse-grained reconfigurable architecture based on HLS (High Level Synthesis) techniques [5]. Especially we focus on the temporal mapping since it fits well with adopting HLS techniques and performing loop level pipelining.

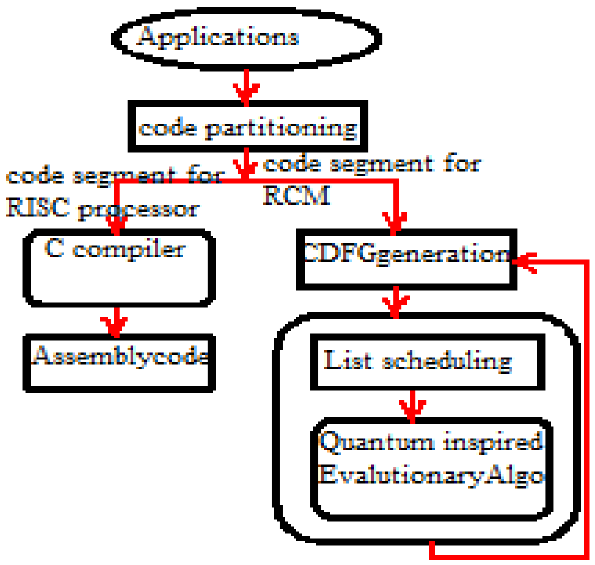


Figure 5: Over Design Flow for CGRA

High-level synthesis has been widely used for generating a hardware module from plain C source code as well as HDL[6]-[8] (hardware description language) code. With a given resource constraint (e.g. number of adders, multipliers, or buses), it produces schedule and binding information for each resource. The proposed mapping algorithm performs HLS with the number of PEs in each column as the resource constraint.

From the given plain C source code, we generate a control Data Flow Graph (CDFG) using SUIF2 [9] parser. In this process, we perform loop unrolling to maximize the utilization of PEs. To calculate the unroll factor, we first run ASAP (As Soon As Possible) scheduling over one iteration of the loop without any resource constraint. Then we calculate the maximum number of used PEs. If the maximum resource usage is smaller than the given resource constraint (the number of PEs in a column), we increase the unroll factor while it meets the resource constraint.1 After generating a CDFG with proper unroll factor, we perform HLS to get the schedule and binding information of one column of PEs. Since we have multiple columns, we can run next iterations on the next columns while the first column is running the first iteration. In this manner, we can implement loop pipelining with the reconfigurable array architecture.

HEURISTIC ALGORITHM

In the proposed Modified design flow for CGPA mapping application, we present a fast heuristic mapping algorithm considering Steiner points[10]. It is based on a mixture of two algorithms: List Scheduling and Quantum-inspired Evolutionary Algorithm (QEA).

List Scheduling

Over the CDFG[11], list scheduling runs with the given resource constraint in order to sorts the vertices from the sink to the source. If a vertex has a longer path to the sink, then it gets a higher priority. Here it assume mesh connectivity of 16 (4 × 4) PEs in an array. In this, four PEs in each column perform the operations in one iteration of a loop in cycle1, cycle2, cycle3, and so on. Since data transfer between neighbors PEs takes one cycle, the data is actually transferred to the PEs in the next cycle.

Here one important thing to be considered is that as the traversed paths are implemented, the number of remaining PEs will be decreased, which in turn decreases the chance to find a solution in a later phase. Therefore, we have to take care in ordering the edges to be routed. In list scheduling, the order of edges is fixed by the sorting of the vertices.

Quantum-Inspired Evolutionary Algorithm (QEA)

The QEA is an evolutionary algorithm that is identified to be very efficient when compared to other evolutionary algorithms. A Quantum-inspired Evolutionary Algorithm (QEA)[1], which is based on the concept and principles of quantum computing, such as a quantum bit and superposition of states. Like other evolutionary algorithms, QEA is also characterized by the representation of the individual, evaluation function. However, instead of binary, numeric, or symbolic representation, QEA uses a Q-bit, defined as the smallest unit of information, for the probabilistic representation and a Q-bit individual as a string of Q-bits. A Q-gate is introduced as a variation operator to drive the individuals towards to get better solutions. Thus the results based on QEA perform well.

It represents the solution space by encoding each solution have a set of Q-bits. A Q-bit is the smallest unit of information in QEA, which is defined with a pair of numbers (α, β) where $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ which gives the probability of the Q-bit to be found in the '0' state and $|\beta|^2$ gives the probability of the Q-bit to be found in the '1' state. A Q-bit may be in the '1' state, or in the '0' state, or in a linear superposition of the two and this is how QEA maintains many potential solutions in a compact way, thereby enabling much faster design space exploration than any other currently known evolutionary algorithm. This concepts is used for Q-bit and superposition of states originate from quantum computing and that is why the algorithm is called QEA.

Q-Bit Encoding

To solve the problem of mapping an application onto the CGRA with QEA, we use a 2- D array (I x J) of Q- bits for each operation in the CDFG [10]. In the Q- bit array, each row represents binding and each column represents scheduling. Since an operation should be scheduled at only one control step and mapped to only one PE, we adopt one-hot encoding technique. Thus only one entry in the array should be evaluated to '1' and all the others should be evaluated to '0'.

Q-Bit Generation and Evaluation

In QEA, the number of possible cases for evaluating α and β is carried and based on the evaluation results the best case is chosen for Q-bit generation. After Q-bit generation, the number of possible cases in a generation is observed. From the array of Q-bits generated, choose only one Q-bit with highest probability among the Q-bits that satisfy controls on data dependency, and then set its state to '1'.

The QEA starts from the list scheduling result as a seed and attempts which helps to reduce the total latency. Since the QEA starts with a relatively good initial solution, it tends to reach a better solution faster.

Routing Path Finding

When the schedule and binding of each vertex are determined, it tries to find the routing path among the vertices with unused remaining PEs to see if these schedule and binding these results violate the interconnect restriction. In this routing phase, we consider the order of edges to be routed as priority.

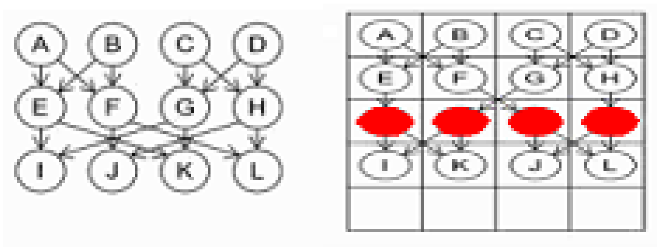
The priority used for the ordering is determined as follows.

- Edges located in the critical path get higher priority.
- Among the edges located in the critical path, edges that have shorter distance get higher priority based on the concept of steiner tree algorithm .

According to the above priority, we make a list of edges and find a shortest path for each edge. In this routing phase, we consider a Steiner tree for multiple wires from single source.

EXPERIMENTAL SETUP

Both the existing and the modified 4 x 4 Processing Elements (PEs) in CGRA targeted for application mapping for uses butterfly addition example shown in Figure 5.



(a) **(b)**

Figure 5: Butterfly Addition Operation (a) Data Flow Graph (DFG) (b) Routing with Steiner Tree

Figure 6. Shows the experimental setup for existing 4 x 4 PEs array. Inputs In1, In2, In3, In4 are 16-bit wide, and outputs are A1, B5, B6, A2 with 16-bit wide.

According to butterfly addition example outputs A1, B5, B6 and A2 of existing 4 x 4 PEs are as follows:

$$A1= (in1 + in2) + (in3 + in4)$$

$$B5= (in1 – in2) + (in3 – in4)$$

$$B6 = (in1 + in2) – (in3 + in4)$$

$$A2 = (in1 – in2) – (in3 – in4)$$

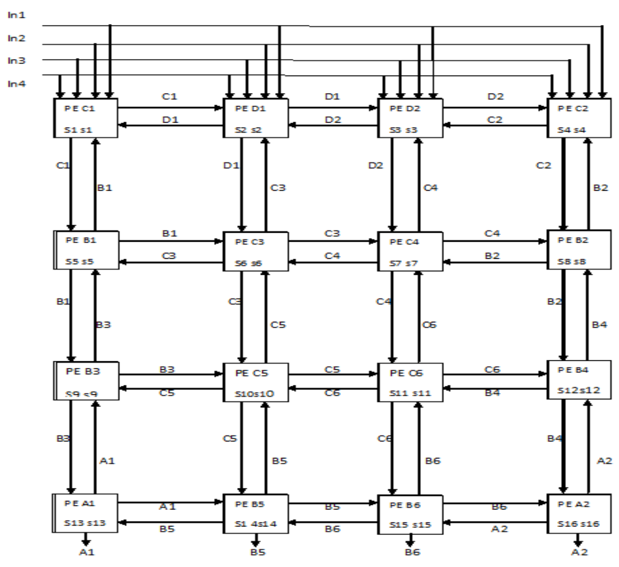


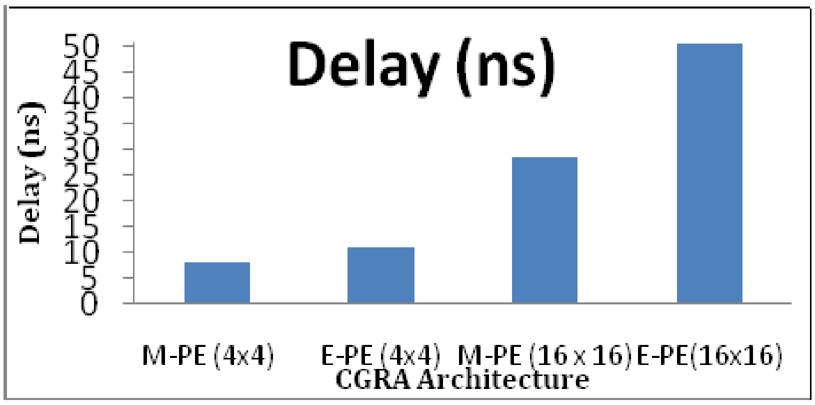
Figure 6: Experimental Setup for Modified 4 X 4 PEs Array

RESULTS

Both the existing and modified CGRA circuits were described in Verilog HDL and simulated using ModelSim. Synthesis was done in Xilinx9.1i targeted for Virtex FPGA device. Power analysis was done in Xilinx 9.1i Xpower analyser.

Table 1: Simulation Results for Integer Arithmetic

Parameters	Modified PEs (M-PE)		Existing 4x4 PEs (E-PE)	
	(4x4)	(16 x 16)	(4x4)	(16x16)
Delay (ns)	8.106	28.52	10.750	50.750
Power (mW)	140.50	320.6	149.55	449.55



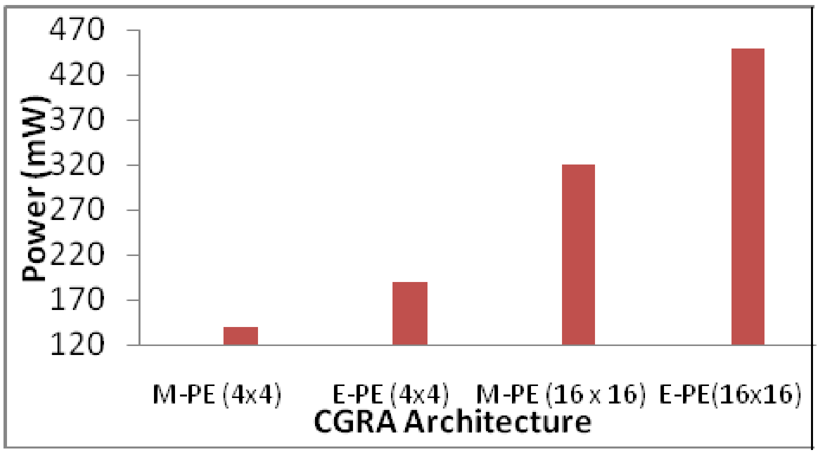


Figure 7: Comparison Graph: (Delay & Power Analysis)

Synthesis Report of Modified 4 x 4 PEs

Device Utilization Summary

Selected Device	:	xa3v5000-4fg900-4
Number of Slices	:	266 out of 33280
Number of 4 input LUTs	:	224 out of 66560
Number of IOs	:	131
Number of bonded IOBs	:	130 out of 6332

Delay Summary Report

Speed Grade	:	-4
Delay	:	8.106 ns
Total	:	8.106 ns (5.347 ns logic, 2.759 ns route)
		(66.0% logic, 34.0% route)

Synthesis Report of Existing 4 x 4 PEs

Device Utilization Summary

Selected Device	:	xa3v5000-4fg900-4
Number of Slices	:	688 out of 33280 (2%)
Number of 4 input LUTs	:	1159 out of 66560 (1%)
Number of IOs	:	259
Number of bonded IOBs	:	258 out of 633 (40%)

Delay Summary Report

Speed Grade	:	-4
Delay	:	10.750 ns
Total	:	10.750 ns (7.515ns logic, 3.235ns route)
		(69.9% logic, 30.1% route)

CONCLUSIONS

In this paper we have presented a fast heuristic approach considering Steiner points for existing and modified PEs array for CGRA Integer application that achieves extreme performance improvement, finds optimal solutions within a few seconds. The design are described in verilog HDL and simulated in ModelSim 6.3g_p1 and synthesized on Xilinx Virtex FPGA and power analysis was done in Xpower analysis tool. From table.1 it is shows that the proposed modified PE based CGRA have 25% reduction in delay and 20% reduction in power for a sample of (4 x 4) and (16 x 16) PEs, which results in enhanced throughput and low power architecture of the application targeted to CGRA. The percentage of reduction in delay and power would be much higher as the size of the Processing Elements (PEs) array grows, which is a noted savings for higher end applications that is need for low power and high performance. The Proposed architecture can be targeted for multimedia applications using test bench for higher efficiency & low power designs.

REFERENCES

1. Ganghee Lee, Kiyoung Choi, *Senior Member, IEEE, and Nikil D. Dutt, Fellow, "*Mapping Multi-Domain Applications onto Coarse-Grained Reconfigurable Architectures", *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems, vo*l. 30, no. 5, pp. 637 – 650, May 2011.

2. H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. M. C. Filho, "Morphosys: An integrated reconfigurable system for data parallel and computation-intensive applications," *IEEE Transactions on Computes.*, vol. 49, no. 5, pp. 465 – 481, May 2000.

3. PACT XPP Technologies [Online]. Available: http://www.pactxpp.com.

4. B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, “ADRES: An architecture with tightly coupled VLIW processorand coarse-grained reconfigurable matrix,” in *Proceedings of Field Programmable Logic Array (FPLA)*, 2003, pp. 61–70.

5. H. Park, K. Fan, S. A. Mahlke, T. Oh, H. Kim, and H. Kim, “Edge centric modulo scheduling for coarse-grained reconfigurable architectures,” in *Proceedings of PACT*, Oct. 2008, pp. 166–176.

6. J. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek, “A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architecture,” IEEE Transactions on Very Large Scale Integrated Systems, vol. 17, no. 11, pp. 1565–1578, Jun. 2008.

7. Y. Ahn, K. Han, G. Lee, H. Song, J. Yoo, and K. Choi, “SoCDAL: System-on-chip design accelerator,” ACM Transactions on Design Automation in Electronic Systems, vol. 13, no. 1, pp. 171–176, Jan. 2008.

8. Y. Kim, M. Kiemb, C. Park, J. Jung, and K. Choi, “Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain specific optimization,” in Proceedings *of DATE*, vol. 1, no. 1, pp. 12-17, Mar. 2005.

9. The SUIF Compiler System [Online]. Available: http://suif.stanford.edu

10. Yoonjin Kim, Ilhyun Park and Kiyoung Choi*,”* Low Power Reconfiguration Technique for Coarse-Grained Reconfigurable Architecture” IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 17, NO. 5, MAY 2009.

11. Ganghee Lee, Seokhyun Lee, and Kiyoung Choi “Automatic Mapping of Application to Coarse-Grained Reconfigurable Architecture based on High-Level Synthesis Techniques”, 978-1-4244-2599-0/08/$25.00 ©2008 IEEE.

AUTHOR’S DETAILS

J. Selvakumar received his B.E degree in Electrical & Electronics from the Jerusalem College of Engineering, Madras University, India in 1999, M. E. degree in VLSI Design from ANNA University, Chennai in 2003. Currently, he is an Assistant Professor (Senior Grade) in the Department of Electronics and Communication Engineering at S.R.M. University, Kattankulathur, Chennai, India, working here since 2003. Currently pursuing Ph D in the field of Reconfigurable Digital Filter Bank.

His research interests include VLSI Signal Processing, Analog VLSI Design, and Efficient VLSI Filter Architecture. He has published four International Journal papers, presented four International Conference papers and one National conference paper to his publication list.

R. Phani VIdyadhar received his B-Tech degree in Electrical & Electronics Engineering from the Prasad V Potluri Siddhartha Institute of Technology, JNTU University, India in 2011. Currently he is doing his Post Graduation M.Tech VLSI Design in S.R.M University Kattankulathur, Chennai, India. Currently working on the project FPGA Implementation of Reconfigurable Architecture for Logical Operations and succeeded in giving satisfactory results.