**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**

***IJCSMC, Vol. 3, Issue. 4, April 2014, pg.212 – 218***

**RESEARCH ARTICLE**

# Design of Coarse Grain Architecture for DSP Application

**Kushal R. Kalmegh** is pursuing M. Tech (CE) in G. H. Raisoni College of Engineering, Nagpur, India (e-mail: kushalkalmegh3@gmail.com)
**Prof. Vaishali Tehre** is Assistant Professor with the Department of Electronics & Telecommunication, G. H. Raisoni College of Engineering, Nagpur, India 440016 (e-mail: vaishali.tehre@raisoni.net)

***Abstract— CGRAs consist of an array of a large number of function units (FUs) interconnected by a mesh style network. In stream-based applications, such as telecommunications, data encryptions, and signal processing are the workloads in many electronic systems. In these applications, the real-time constraints often have stringent energy and performance requirements. The application-specific integrated circuits (ASICs) become inevitably a customized solution to meet these ever-increasing demands for highly repetitive parallel computations. In this project proposing a coarse grain architecture and mapping of some DSP Algorithm.***

***Keywords— CGRAs, FUs, GPS, PE***

## I. Introduction

CGRAs consist of an array of a large number of function units (FUs) interconnected by a mesh style network. In stream-based applications, such as telecommunications, data encryptions and signal processing are the workloads in many electronic systems. In these applications, the real-time constraints often have stringent energy and performance requirements. The general purpose solutions, such as general purpose processors (GPPs), are widely used in conventional data-path oriented applications due to their flexibility and ease of use. The application-specific integrated circuits (ASICs) become inevitably a customized solution to meet these ever-increasing demands for highly repetitive parallel computations. Reconfigurable architectures (RAs) have long been proposed as a way to achieve a balance between flexibility as of GPP and ASICs. The hardware-based RA implementation is able to explore the spatial parallelism of the computing tasks in targeted applications, by avoiding the instruction fetching, decoding, and executing with the software implementations, which results in performance gain and an energy over general purpose processors. On the other hand, RAs maintains the post fabric flexibility either offline or on the fly, to accommodate to protocol updates new system requirements or new system requirements that is not feasible in ASIC implementations. The flexibility provided by RAs can improve fault tolerance and the system to be reliable.

## II. Key Features

In a typical Cell architecture, a set of cell units are organized in a tiled structure. Each cell block consists of four neighboring processing elements (PEs) along with the control and data switching fabrics. For the intra and inter cell communications, a three-level layered interconnection network is designed. A serial peripheral interface (SPI) is designed for an efficient way to load or reconfigure instruction codes into active cell units. Now reconfiguring the instruction memories, the data flow can be dynamically changed to accommodate to different application demands. Some important features of the Cell architecture are as follows.

*A.Coarse-grained granularity*:
Cell is designed to generate coarse-grained configurable system targeted for computation intensive applications. The processing elements works on 16-bit input signals and generate a 36-bit output signal, which avoids high overhead and has better performance compared with fine-grained architectures.

*B.Flexibility:*

Due to the rich computing and resources in communication, versatile computing styles are feasible to be mapped onto the Cell architecture, including SIMD, MIMD, and 2D systolic array structures. This also expands the overall range of applications to be implemented.

*C. Dynamic reconfiguration*:
By loading new instruction codes into the configuration memory, new operations can be executed on the desired PEs without any interruption with others PEs. The number of PEs involved in the application is also adjustable for different system requirements.

*D. Deep pipeline and parallelism*:
Two levels of pipeline are achieved— one is the instruction level pipeline (ILP) in a single processor element and other is the task level pipeline (TLP) among multiple cells. The parallelism data can also be explored to concurrently execute multiple data streams, which in combine ensures a very high computing capacity

## III. CELL UNIT AND PROCESSING ELEMENT

The reconfigurable cell units are the fundamental components in Cell, which are aligned in a 2D mesh structure as shown in Figure 1. The cell consists of four identical PEs connected with a crossbar switch. The PE is consists of an arithmetic and logic unit, I/O muxes, instruction register, instruction controllers, data memories and instruction memories as shown in Figure 1. It can be configured to perform the functions like logic, shift, and arithmetic etc. The arithmetic unit takes two 16-bit vectors as inputs for basic arithmetic functions to generate a 36-bit output without any loss of precision during multiply-accumulate operations. The PE also includes some shift and logic operators, usually found in targeted data streaming applications. The basic operations supported by Cell processor are listed in Table 1. Through the programmable on-chip connections, multiple PEs can be chained together to implement more complex algorithms.
An up to 4-stage pipeline structure is developed in each processor, as shown in Figure 1.

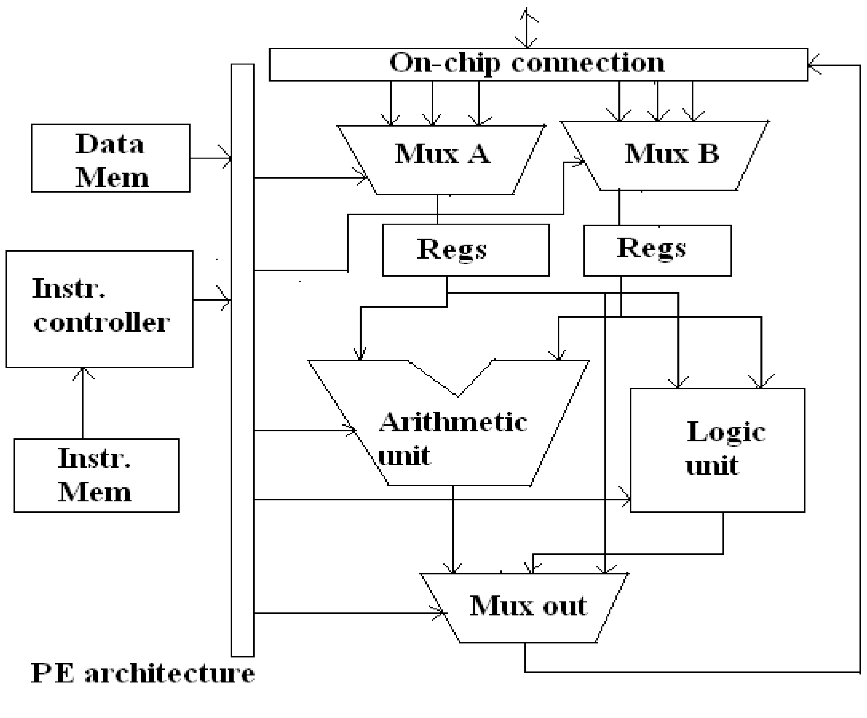


Fig. 1. Processing element architecture

The stage inputs data from the on-chip connection is selected by Src select stage and calculated by other PEs or itself and stores the data into its local register banks. For basic multiply-add and other logic operations, the execution stages (*Exe1* and *Exe2* ) occupy two clock cycles. The *Des* select stage selects the output result and sends it back to the on-chip interconnections. The pipeline stages are not fixed in this Cell, as present in traditional pipelined parallel processor. To allow fast passing through input data or intermediate results to the next operating unit, bypass path can be selected in every stage except for *Src* select to reduce the processing delay if required. The traditional pipelined parallel processor, decoding stage is replaced by an instruction controller, which generates all scheduling and control signals in parallel with the 4 pipeline stages. In an instruction memory, pre-stored instruction code is loaded into the instruction controller on a cycle-by-cycle basis for provide both functionality and data path control for a specific algorithm.

In flexible 4-stage pipeline structure avoids the deep instruction pipelines of fetching operation, decoding operation, and registering read/write and ALU operations in conventional general purpose processors. In Addition, the instruction code can be dynamically reconfigured in various modes to adapt to different application requirements. Therefore, Cell is able to provide comparable energy efficiency as an ASIC while maintains dynamic programmability as a DSP.

The instruction code is designed in a 8-bit frame format. Program counter control (PC control) section is used to indicate execution time, next one instruction address, and valid memory ranges for the mapped application. The configuration of data flow and computing units are specify the data path and operation control signals, while for synchronization scheduling among multiple computing units I/O delays are used. To configure the on-chip communication network, a Network-on-Chip (NoC) control signal is designed.

New instruction will not to be loaded and executed until the previous current one expires. The instructions are accessed in a cycle by cycle manner that supports periodical execution of a set of operations. In our current design, a 8-bit instruction memory block is attached to each PE. In a cell, four PEs placed in the east, west, north and south directions forma quad structure with a fully connected crossbar unit which is located at the center, as shown in Figure 2. Arbitrary non blocking connections among PEs in the same cell are supports by the crossbar network. Instruction memories are connected to each PE and are chained in a linear array fashion by serial peripheral interface (SPI) for configurations. The controls of data exchange are also provided by the instruction code in a cycle by cycle manner.

## *A. Fully Connected Crossbar Intracell Interconnection.*

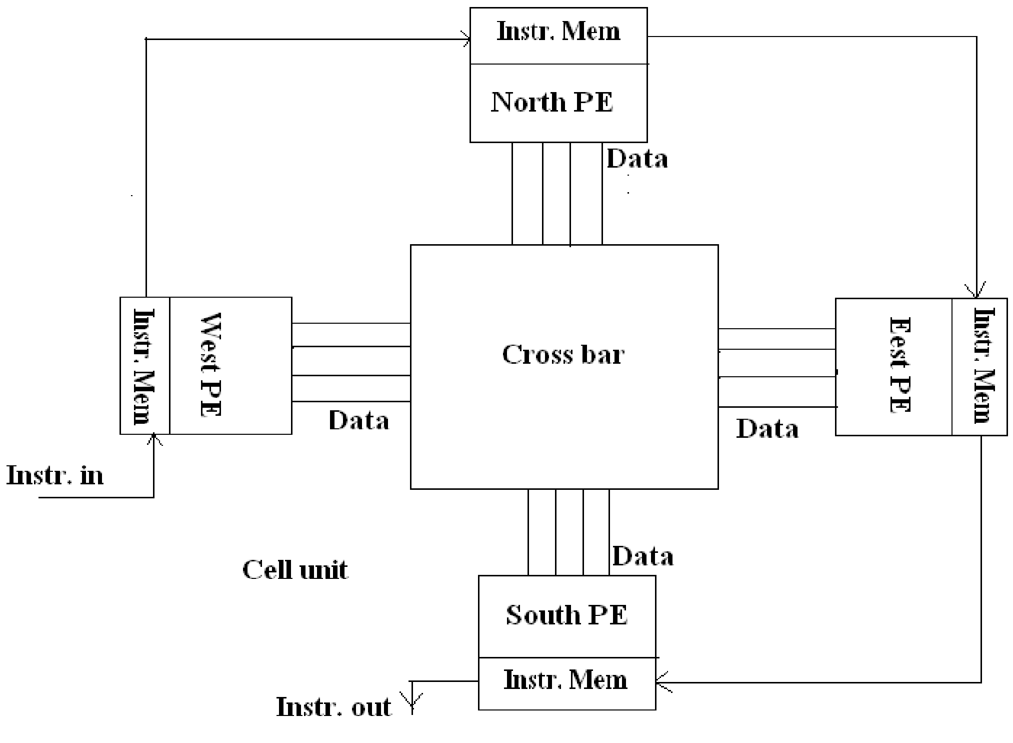


Fig. 2. CGRA (cell) architecture

For the intra cell communications, a centralized shared register memory (SRM) block is designed. But it was desert due to its high area and power costs and complex memory access controls. In the current design, the PEs and instruction memories are placed at the four edges in a complete cell. A non blocking data exchange connection is able to provide by fully connected crossbar switch box. As compared to the SRM implementation, the control logics are greatly simplified in the crossbar connection, which in turn results in a better area performance and timing.

## *B. Comparison of Power/Energy Consumption with FPGA and ASICs.*

In this section, we compare the power and energy consumption performance with other computing platforms, including GPP, FPGA and ASIC. Due to similar algorithm mapping structures and the same simulating frequency. In CGRA, Cell is about 2.7 ~

5.4 less power efficient than ASICs. On the other hand, in CGRA Cell outperforms FPGA by a factor of 2.7 ~ 4.8 in terms of power consumption. A more meaningful figure is depicted in Figure(8) that compares the average energy efficiency among the evaluated to FPGA result. Among the evaluated platforms, the ASICs are the most energy efficient. However, this performance gain is achieved at a cost of no post fabrication flexibility and high engineering design cost. This result clearly shows that the coarse-grained architecture is able to fill the energy efficiency gap between the fine-grained FPGA and logic specific ASIC architectures.

## C. *Figures and Tables*

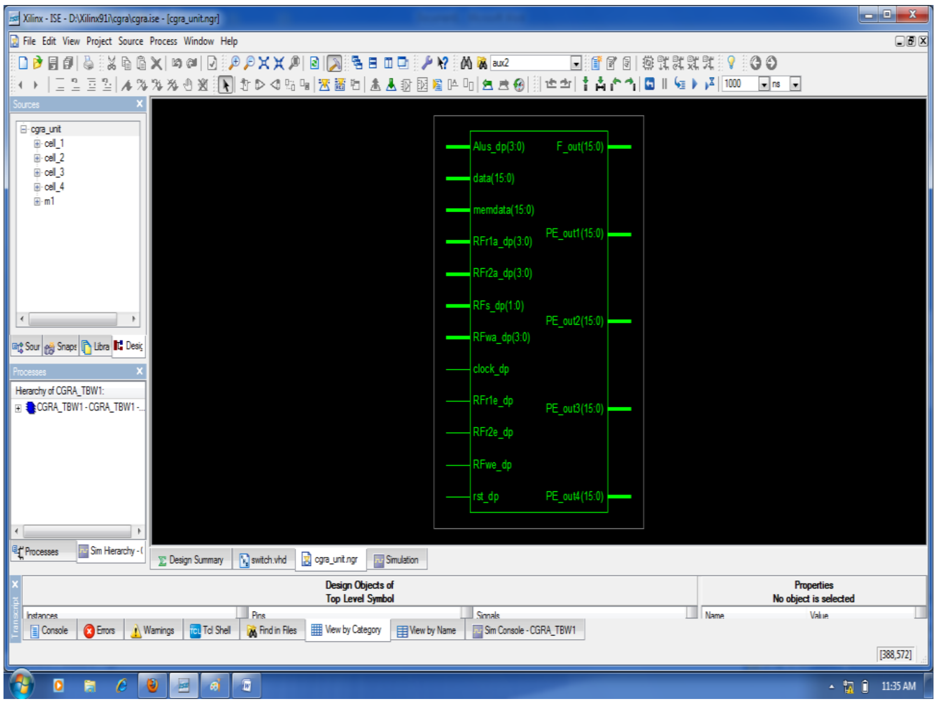


Fig. 3. RTL Schematic of CGRA (cell) architecture

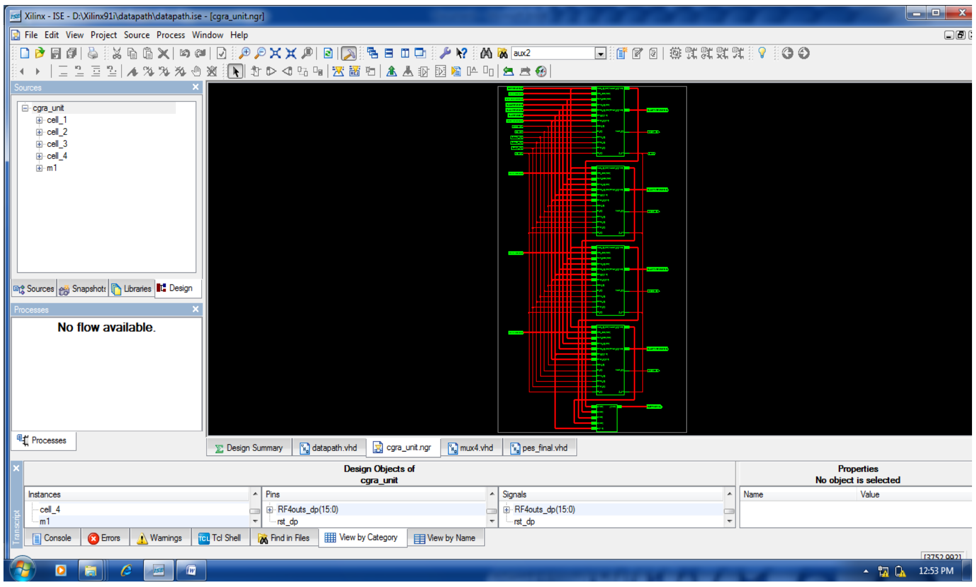


Fig. 4. Technology Schematic CGRA (cell) architecture

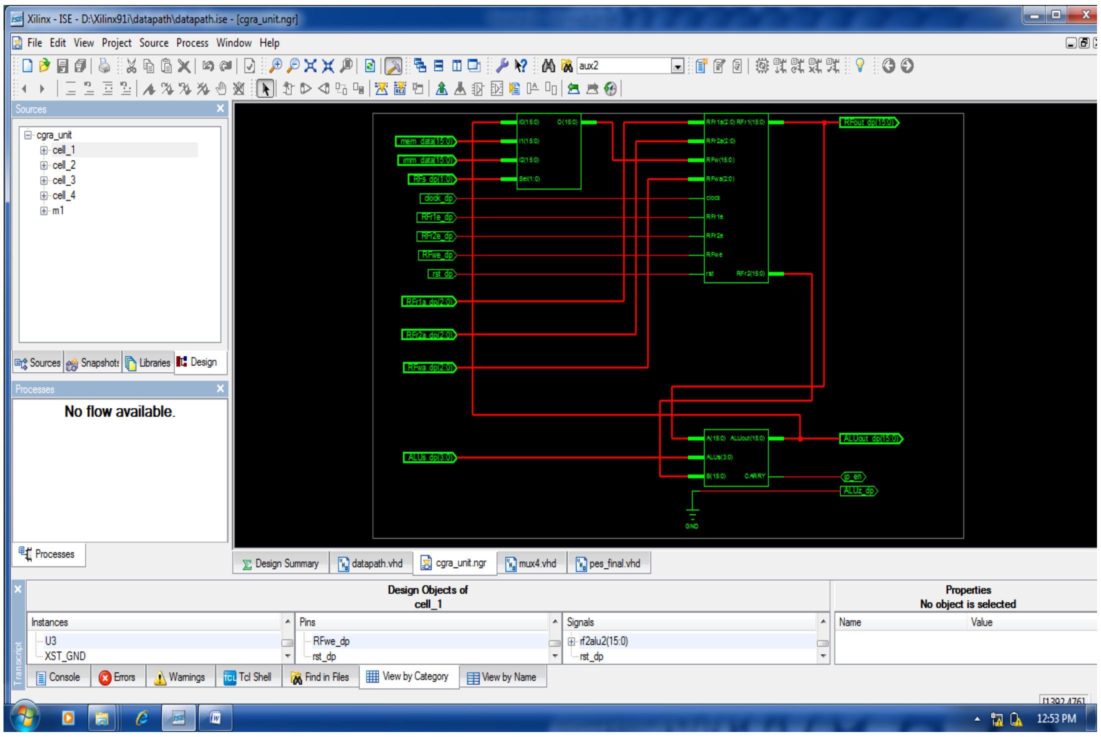


Fig. 5. Technology Schematic of processing element

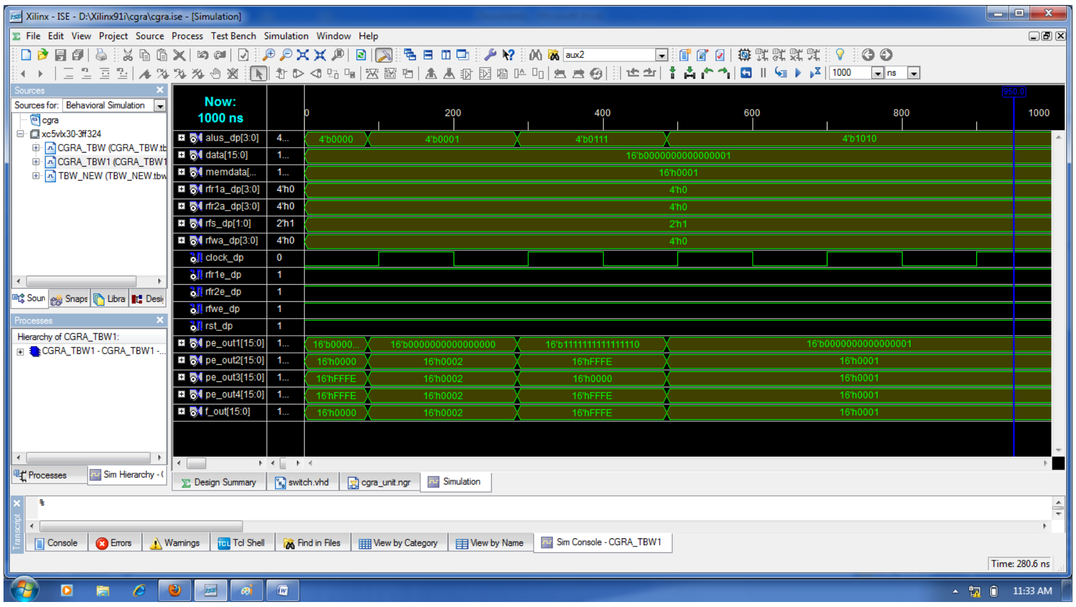
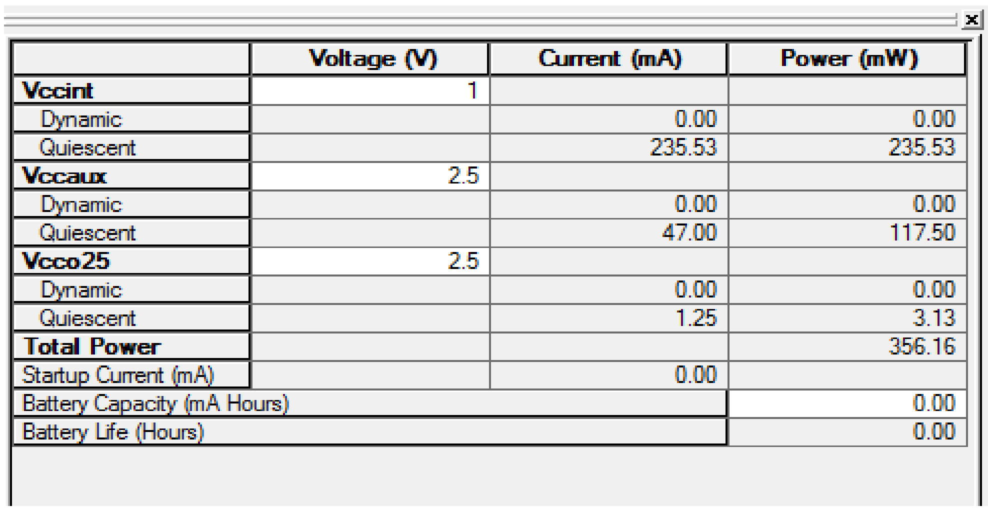
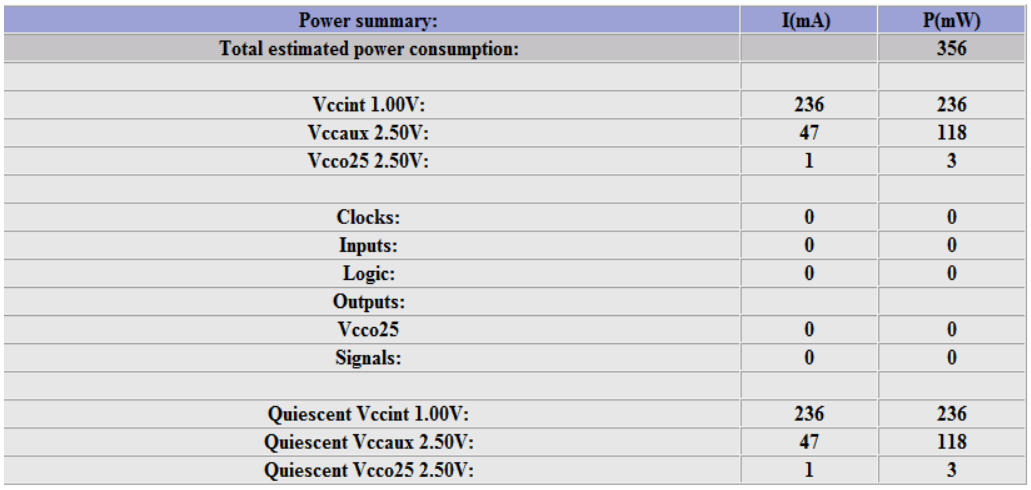


Fig. 6. Simulate waveform



| | Voltage (V) | Current (mA) | Power (mW) |
|---|---|---|---|
| **Vccint** | 1 | | |
| Dynamic | | 0.00 | 0.00 |
| Quiescent | | 235.53 | 235.53 |
| **Vccaux** | 2.5 | | |
| Dynamic | | 0.00 | 0.00 |
| Quiescent | | 47.00 | 117.50 |
| **Vcco25** | 2.5 | | |
| Dynamic | | 0.00 | 0.00 |
| Quiescent | | 1.25 | 3.13 |
| **Total Power** | | | 356.16 |
| Startup Current (mA) | | 0.00 | |
| Battery Capacity (mA Hours) | | | 0.00 |
| Battery Life (Hours) | | | 0.00 |

Fig. 7. Voltage & current parameters



| Power summary: | I(mA) | P(mW) |
|---|---|---|
| Total estimated power consumption: | | 356 |
| | | |
| Vccint 1.00V: | 236 | 236 |
| Vccaux 2.50V: | 47 | 118 |
| Vcco25 2.50V: | 1 | 3 |
| | | |
| Clocks: | 0 | 0 |
| Inputs: | 0 | 0 |
| Logic: | 0 | 0 |
| Outputs: | | |
| Vcco25 | 0 | 0 |
| Signals: | 0 | 0 |
| | | |
| Quiescent Vccint 1.00V: | 236 | 236 |
| Quiescent Vccaux 2.50V: | 47 | 118 |
| Quiescent Vcco25 2.50V: | 1 | 3 |

Fig. 8. Various power summary of evaluated CGRA systems

## REFERENCES

[1] R. Ferreira, J. Vendramini, and M. Nacif. "Dynamic reconfigurable multicast interconnections by using radix-4 multistage networks in fpga." In IEEE 9th International Conference on Industrial Informatics, INDIN, 2011.

[2] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, and Y. Paek. "A graph drawing based spatial mapping algorithm for coarse-grained reconfigurable architectures." IEEE Trans. Very Large Scale Integr. Syst.,17:15651578,November 2009.

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

[4] A. DeHon, Y. Markovsky, E. Caspi, et al., "Stream computations organized for reconfigurable execution," Microprocessors and Microsystems, vol.30,no.6,pp.334– 354,2006.

[5] 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 Design, Automation and Test in Europe (DATE '05), vol. 1, pp. 12– 17, 2005.

[6] SmartCell: An Energy Efficient Coarse-Grained Reconfigurable Architecture for Stream-Based Applications Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, MA 01609, USA Correspondence should be addressed to Xinming Huang, xhuang@ece.wpi.edu Received 2 February 2009; Accepted 15 April 2009.

[7] T. Marshall, L. Stansfield, J. Vuillemin, and B. Hutchings, "A reconfigurable arithmetic array for multimedia applications," in Proceedings of the ACM/SIGDA 7th International Symposium on Field Programmable Gate Arrays, pp. 135–143, 1999.

[8] C. Fisher, K. Rennie, G. Xing, et al., "Emulator for exploring RaPiD configurable computing architectures," in Proceedings of the 11th International Conference on Field-Programmable Logic and Applications, pp. 17–26, 2001

[9] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, "Architecture exploration of the ADRES coarse-grained reconfigurable array," in Springer Reconfigurable Computing: Architectures, Tools and Applications, pp. 1–13, 2007.

[10] S. Khawam, T. Arslan, and F. Westall, "Synthesizable reconfigurable array targeting distributed arithmetic for systemon-chip applications," in Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04), pp 2051–2058, 2004.

[11] B. Khailany, W. J. Dally, U. J. Kapasi, et al., "Imagine: media processing with streams," IEEE Micro, vol. 21, no. 2, pp. 35– 46, 2001.

[12] H. Singh, M.-H. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh, and E. C. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," IEEE Transactions on Computers, vol. 49, no. 5, pp. 465–481, 2000.

[13] M. B. Taylor, J. Kim, J.Miller, et al., "The RAWmicroprocessor: a computational fabric for software circuits and generalpurpose programs," IEEE Micro, vol. 22, no. 2, pp. 25–35, 2002,.

[14] M. B. Taylor, J. Kim, J.Miller, et al., "The RAWmicroprocessor: a computational fabric for software circuits and generalpurpose programs," IEEE Micro, vol. 22, no. 2, pp. 25–35, 2002.

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

[16] A. DeHon, Y. Markovsky, E. Caspi, et al., "Stream computations organized for reconfigurable execution," Microprocessors and Microsystems, vol. 30, no. 6, pp. 334–354, 2006.

[17] 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 Design, Automation and Test in Europe (DATE '05), vol. 1, pp. 12–17, 2005

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