Energy-Aware Coarse Grained Reconfigurable Architectures Using Dynamically Reconfigurable Isolation Cells

ROYAL INSTITUTE

OF TECHNOLOGY

OZAN ZEKİ BAĞ

Master's Degree Project
Stockholm, Sweden 2012

TRITA-ICT-EX-2012-249

Acknowledgements

I would like to thank my family, all of my friends and to Bella for the great support through not only for my thesis, but also for the whole time I spent in Sweden.

I appreciate Professor Ahmed Hemani to give this opportunity to work in such a novel project. Special thanks to my supervisor PhD candidate Syed Mohammad Asad Hassan Jafri for his patient guidance and valuable information that he shared with me and I will remember for the rest of my career. Moreover, thanks to PhD candidate Nasim Farahini for great advice.

Abstract

This thesis presents a self adaptive power management system to improve energy efficiency of coarse-grained reconfigurable architectures (CGRAs). CGRAs can host multiple applications on a single platform. Moreover, a single application may have multiple versions which have different degree of parallelism (fully serial, partially serial, fully parallel etc.). Selection of the optimum application version depends on runtime conditions such as resource availability on the platform. A traditional worst case design to satisfy its specifications results in undesirable power efficiency. Existing solutions to this problem offer costly hardware to mainly employ dynamic voltage and frequency scaling (DVFS). We propose exploiting reconfiguration of available resources on CGRA. Our solution makes use of dynamically reconfigurable isolation cells (DRICs) instead of dedicated hardware. We also introduce autonomous parallelism, voltage and frequency selection (APVFS) to realize DVFS functionality and to select the optimum version. Three applications are used for simulations, namely; matrix multiplication, finite impulse response filter (FIR) and fast Fourier transform (FFT). Results show that up to 72 % and 55 % power and energy can be saved respectively. Synthesis of the fabric shows considerable reduction in area overheads compared to existing designs employing DVFS.

Contents

List of Figures

List of Tables

List of Abbreviations

FPGA	Field Programmable Gate Array
CGRA	Coarse Grained Reconfigurable Architecture
DRRA	Dynamically Reconfigurable Resource Array
DVFS	Dynamic Voltage and Frequency Scaling
DRIC	Dynamically Reconfigurable Isolation Cell
RTM	Run Time Resource Manager
CGIR	Compact Generic Intermediate Representation
GRLS	Globally Ratio Synchronous Locally Synchronous
GnS	Globally-non-Synchronous
GALS	Globally Asynchronous Locally Synchronous
PMU	Power Management Unit
VCU	Voltage Control Unit
CGU	Clock Generation Unit
mDPU	morphable DataPath Unit
MAC	Multiply and ACcumulate
AGU	Address Generation Unit
RTL	Register Transfer Level
MANAS	MANuel ASsembly
PCB	Printed Circuit Board

Chapter 1

Introduction

The complexity of electronic systems has increase continuously since the introduction of first integrated circuits. Since 1965, Moore's law accurately formulated the increase in complexity. Moore's law states that; the number of transistors which can be placed on the same size of logic die doubles approximately every two years [1]. This concept is known as *technology scaling*. Today, few billions of transistors can be placed on a cm^2 die and in the future it will be even more [2]. When electronic circuits were first proposed by mid-20th-century, power consumption of the integrated circuits was lower and heat dissipation was easier to handle. The reason is, although a single transistor was larger and consuming more power compared to today's transistors, speed of the circuits was lower due to physical constraints on transistor size, lowering the switching activity and the total power consumption.

Power consumption and heat dissipation have become today's fundamental challenges to further increase the performance and computational power. This force is driving the industry to figure out new ways of power management. Some applications such as multimedia processing and telecommunication consume more power than the others due to handling vast amounts of data in a repetitive fashion. Reconfigurable hardware architectures provide the flexibility of adapting different applications on the same device and also possibilities of parallelization for both data handling and power consumption techniques. That is why, exploration of new power management methods in reconfigurable hardware architecture is of great interest.

Reconfigurable hardware devices allow data manipulation in different levels. Granularity is defined as the level of manipulation in terms of the number of bits that can be manipulated by the programmer. The most widely used reconfigurable architectures are Field Programmable Gate Arrays (FPGAs), enabling one single bit manipulation of data. Coarse Grained Reconfigurable Architectures (CGRAs) provide operator level manipulation of functional blocks, word level datapaths, and yet they are very area efficient compared to FPGAs.
Also, compared to fine-grained architectures, CGRAs enjoy massive reduction of configuration memory and configuration time, as well as considerable reduction in routing and placement allocation. All of these aspects also result in a potential reduction of the total

energy consumed per computation at the cost of a loss in flexibility compared to bit-level operations. To present the benefits that CGRAs enjoy over fine grained reconfigurable architectures, a lot of work has been done on improving the architecture, mapping applications, increasing reliability and optimizing power consumption [3] in CGRAs, which will be discussed in related work section.

CGRAs are composed of many computational resources such as algorithmic logic units (ALUs), multipliers, dividers etc. These resources result in excessive power consumption in CGRAs. With such resources available, modern CGRAs can accommodate multiple applications running simultaneously on a single platform. Potentially, different applications require different workloads and performance. Besides that, same outputs can be achieved with different versions of the same application (serial, parallel, partially-parallel designs etc.). The optimum version of the application at a given time instance is highly dependent on runtime conditions, such as available resources. Yet, this information cannot be predicted during compile time. A traditional worst case approach would result in lower performance and increased power consumption. Because, the platform would be to configured handle the biggest deviations.

This thesis work addresses power management challenges in CGRAs. Proposed methods achieve Dynamic Voltage and Frequency Scaling (DVFS) with late-binding. This concept will be referred as late-binded dynamic voltage and frequency scaling with support for multiple versions (LDV) from here on, which can be interpreted as a power intelligence as well. Proposed LDV methods make use of a compile time generated profile to select optimum voltage, frequency and the available version of the application and dynamically adjust these parameters during run- time. To realize LDV concept, there are challenging factors such as voltage switching overheads, synchronization overheads between the partitions working with different frequencies and etc.

In this thesis work, main focus will be reducing the synchronization overhead, originating from power hungry buffers, used to ensure safe communication between blocks running at different frequencies [4].

Proposed method makes use of dynamically reconfigurable isolation cells (DRICs) to ensure safe communication between different frequency islands on the chip, rather than using

dedicated buffers. DRICs reduce the synchronization overhead by reconfiguring one of the existing resources to act as an isolation cell, eliminating the need for additional buffers (implemented in ASIC). To overcome memory requirements, we present an autonomous parallelism, version and frequency selection algorithm (APVFS). APVFS has the role of run-time behavior observation over the system and share this information with the mapper.

Depending on the available resources, potential parallelization options and specified deadlines, mapper can chose the optimum voltage-frequency-application version triplet. This method requires only a single profile to be stored.

To host the applications and proposed LDV method, CGRA architecture called Dynamically Reconfigurable Resource Array (DRRA) that was developed in KTH is used. Please refer to Chapter 3 for the detailed explanation of this fabric. We have synthesized the architecture using 65 nm multi Vdd technology. Running many practical applications (FFT, FIR, Matrix multiplication) shows a significant reduction in power and energy consumption ,72%, 55% reduction respectively.

1.1 Related work

Both optimum configuration selection and DVFS overhead handling (using reconfiguration) are of interest for this thesis work, therefore, recent work from both areas are reviewed. DVFS has been an area of extensive research in recent years for system on chip design[6]. Existing research focuses on improving DVFS algorithms to reduce the energy consumption (mostly software based) and minimizing the overheads of DVFS architecture itself. Due to scope of this thesis, only the most prominent works focusing on minimizing DVFS overheads using reconfiguration are reviewed. Liang et al [7] proposed the use of reconfigurable links to reduce the overheads imposed by the FIFOs in the synchronizers for NoCs. Amir et al [6] presented a similar structure for the network on chips. The reconfigurable links are able to bypass the FIFOs if two cells are operating at same frequency. Both of these two methods require dedicated reconfigurable buffers and the reconfiguration is only used to minimize the timing overhead. Yang et al [7] proposed a method to apply dynamic voltage scaling for reconfiguration in reconfigurable hardware. Wrap processor [8] monitors the program at runtime and creates appropriate hardware structure for computation intensive parts. The fast execution of certain parts creates idle slacks. The voltage is later scaled to take advantage of these idle slacks. We use dynamic mapping of the isolation cells inspired from this method.

The difference is that we focus on reconfiguration to reduce synchronization overheads. We rely on a faster (parallel) implementation for generation of time.

A technique capable of taking advantage of the reconfiguration features of modern reconfigurable architectures is missing. In CGRA domain, only software based solutions have been presented [9], [10] without any reference to the underlying architecture involved. Nollet [11] presented a good review of runtime resource managers (RTMs). Traditionally, the RTMs were provided with only one implementation per application considering the worst case [12]. Abbas [13] explored the possibility of dynamically shifting between 2 versions of an application, at runtime. Couvreur [14] presented a two phase method for optimal application mapping, in terms of energy. The work was later improved in [15] by providing multiple criteria for selection of optimal versions. However, this method incurs prohibitive memory requirements and therefore limit the versions that can be stored. Moreover, this method relies on profiling all versions with all frequencies causing excessive memory requirements and slowing the runtime mapping selection of mapping. Asad et al [16] presented a method to significantly reduce the storage overheads of two phase method by storing multiple configurations as a compact customizable representation. In this paper we use the Compact Generic Intermediate Representation (CGIR) inspired from this work and present criteria for runtime unraveling of the code.

1.2. Contributions

This thesis work has three major contributions;

- The proposed method significantly reduces the memory requirements and the performance of runtime resource managers,
- Implementation of low latency DVFS using rationally related frequencies,
- APVFS algorithm to autonomously select the application version with high energy efficieny,
- An enabling control and management layer which serves to realize the above concepts.

Chapter 2

Globally Ratio Synchronous Locally Synchronous Design (GRLS)

Fully synchronous design style has advantages such as simplicity in the design flow, time to market, availability of reliable design and test tools. However, with technology scaling, the need for alternative solutions has increased. Despite this need, asynchronous design tools did not develop as much as synchronous design tools [17]. *Globally-non- Synchronous* (GnS) design style offers the transition in the industry between fully synchronous and full asynchronous designs. In GnS scheme, sub modules of the a design stay synchronous working at the same clock frequency. However, different modules can work in different clock frequencies. The communication between the cells with different operating frequencies is usually handled using custom built synchronizers, if there is data transfer needed. This thesis work offers a reconfigurable solution for the communication rather than a custom built solution.

2.1. GALS vs. mesochronous (Two branches of GnS)

Globally Asynchronous Locally Synchronous (GALS) and *mesochronous* design styles constitute two main branches of GnS design with mesochronous design being a subset of GALS design [21-23]. In mesochronous design, all the modules in a particular design, work at the same frequency, whereas the phase difference between the modules is unknown. In GALS systems, all the modules can run at different frequencies, also no assumption is made on phase alignment. Mesochronous designs may take the advantage of relatively simpler synchronizers for phase alignment, however they cannot perform DVFS functionality due to modules working at the same frequency. GALS systems can achieve high efficiency of DVFS functionality where more complex communication interfaces are needed. This comparison is a tradeoff between the two systems. It is really hard to claim that one design style is superior over the other one because the choice depends on parameters such as low area, low cost, higher power efficiency, physical layer characteristics of the underlying hardware etc. which may well differ from each other.

Globally Ratio Synchronous Locally Synchronous (GRLS) design was recently introduced to build high performance multiclock systems [24-25]. This concept requires the sub modules of the design to have rationally related clock frequencies and no phase alignment assumption is taken. Since different communicating applications are mapped close to each other, for this work, we assume the clocks to be phase aligned. Having rationally related operating frequencies for different modules provides low latency while taking the advantage of DVFS functionality. Therefore, GRLS design offers an intermediate solution between GALS and mesochronous systems within GnS approach.

In our energy aware power management system solution for CGRA systems, synchronizers for data communication are realized by dynamically reconfiguring available resources and DVFS is realized by inserting power management units (PMUs) implemented in ASIC. In that sense, this thesis work is a good representation of hardware software co-design style. Observe figure 2.1 for GRLS design below.

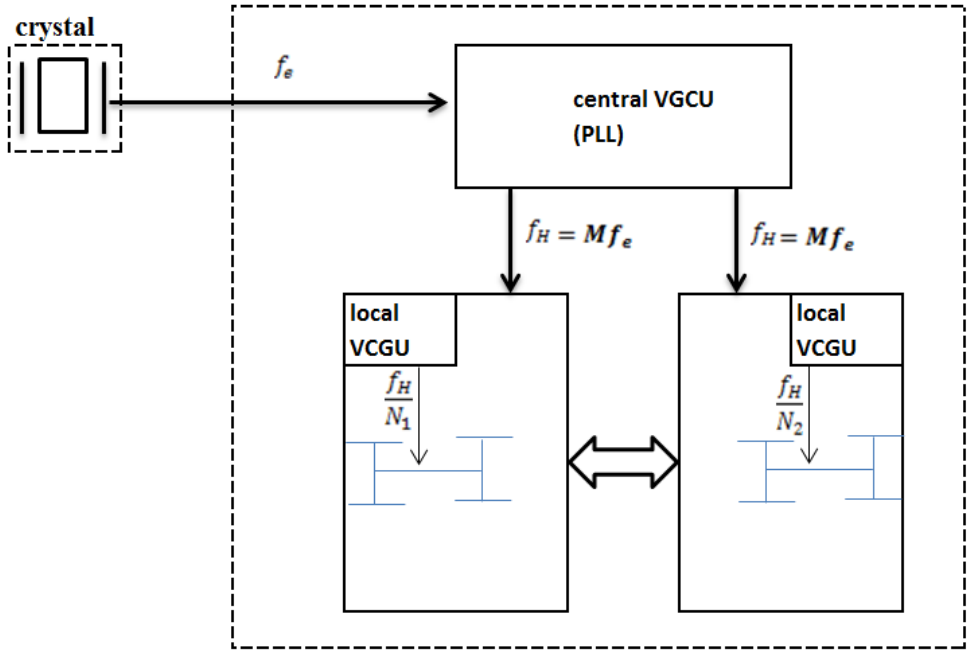

Figure 2.1: GRLS design

In the example design, crystal oscillator feeds the chip with f_e, central VCGU unit sets the desired frequency and local VGCU units further divide or multiplying the frequency of the sub modules. N_1 and N_2 have a common multiple so that the frequency is rationally related for the different partitions on the chip.

Chapter 3

Dynamically Reconfigurable Resource Array (DRRA)

DRRA is a Coarse Grained Reconfigurable Architecture designed in KTH for multiple, radio and multimedia applications with the focus of digital signal processing. There are three main concepts of DRRA, namely;

- Minimized data travel via sliding window concept,
- Reconfigurable and programmable architecture for reuse and different applications,
- Tolerance for manufacturing variations.

DRRA has the key principle that it deploys small, simple, agile pools of resources for computation, storage and interconnection. These pools can be customized during runtime depending on the application needs.

The pools are logically and physically a cluster of resources. Once the data in fed into the fabric, it stays in the memory the logic is moved. This "move the logic, not the data" approach eliminates the infrastructure overheads and performance inefficiencies which are caused by shared resources such as storage elements and interconnects.

DRRA fabric has a function library employing several useful functions and templates. Programmer can use the pre-defined instruction set to create a readable configware. Automatic code generator converts this configware into corresponding sequencer program.

Manufacturing variations in deep sub-micron geometries might create a yield problem and also might have the probability of hardware failure. If a process variations compensation system is implemented, DRRA with its regular topology and its principle of pool of resources, may avoid this problem by isolating failed and out-of-range performance resources. Furthermore, with its reconfigurable and programmable architecture, DRRA can easily cope with changes of specifications by modifying higher level software programmes.

In physical layer, the resource pools offered by DRRA translates into morphable datapath units (mDPUs), register files, switchboxes and sequencers. All the resources are integrated as a regular and seamlessly connected fabric on the logic die, as demonstrated in figure 3.1. The fundamental resource in the DRRA is *DRRA cell*. DRRA cells can be allocated as many as needed, under the restriction of the specifications. The physical positioning of the DRRA cells are realized as rows and columns. Figure 3.1 depicts a 2x7 fabric. One single DRRA cell is consists of one mDPU, one registerfile, one sequencer and two switchboxes. Top right square in the figure illustrates a single DRRA cell and components within. Different DRRA cells are connected by a seamless, sliding window, circuit-switched interconnect fabric. Please refer to switchbox component part for further information on this concept in section 3.3.

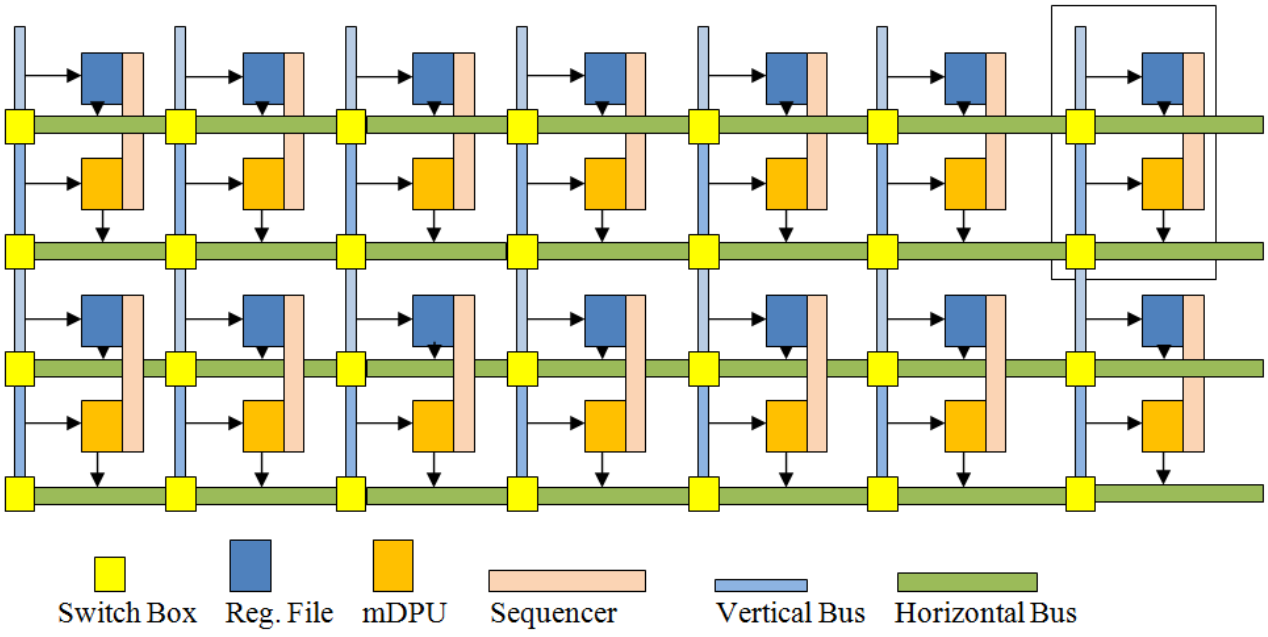

Figure 3.1: DRRA Fabric

3.1. mDPU

The computational unit in a DRRA cell is the mDPU. This unit has four 16-bits signed inputs and two 16-bits signed outputs. mDPU inputs may take complex values. Two input pairs can be used to represent two complex numbers, and the output pair can present the complex output. The mode of operation of the mDPU is set via configware, such as complex addition, multiplication, *multiply and accumulate* (MAC operation) etc. The programmer may choose one of the modes, please refer to Appendix A for mDPU instruction details. Once the mode of the mDPU instruction is loaded into the sequencer and executed, the mDPU will work in the same mode until it is changed later in the application dynamically or until it is configured for some other application.

3.2. Registerfile

The register file in a DRRA cell has 64 words depth and 16 bits word length. Two read and two write ports are implemented. The *address generation unit* (AGU) in register file is capable of performing circular buffered, vectorised and bit reversed addressing which are common practices in digital signal processing. This feature has a great advantage while handling applications such as Fast Fourier Transform (FFT). Read and write operations in register file can be increasing or decreasing from the starting address. Loops can be formed in configware for read and write operations. The read and write operations can be set with an initial delay and can be executed at a specific time during the application. Any combination of the two read and two write ports might be used simultaneously providing parallelism opportunities. Register file operations are specified via configware produced by the programmer. Parameters such as start address, end address, linear addressing, initial delay etc. can be set. Please see Appendix B for the details.

3.3. Switchbox

Switchbox is the component providing connections in DRRA c. Switchboxes are located in the intersections of the horizontal and vertical busses in the DRRA fabric as depicted in figure 3.1, where vertical busses represent the input lines, *vlanes,* and horizontal busses represent the output busses, *hb-mDPU* and *hb-reg.*

The input and outputs of the switchboxes are connected to vertical and horizontal busses, so that switchbox can connect different components. To see available instructions and parameters, please refer to Appendix C switchbox connect instruction details.

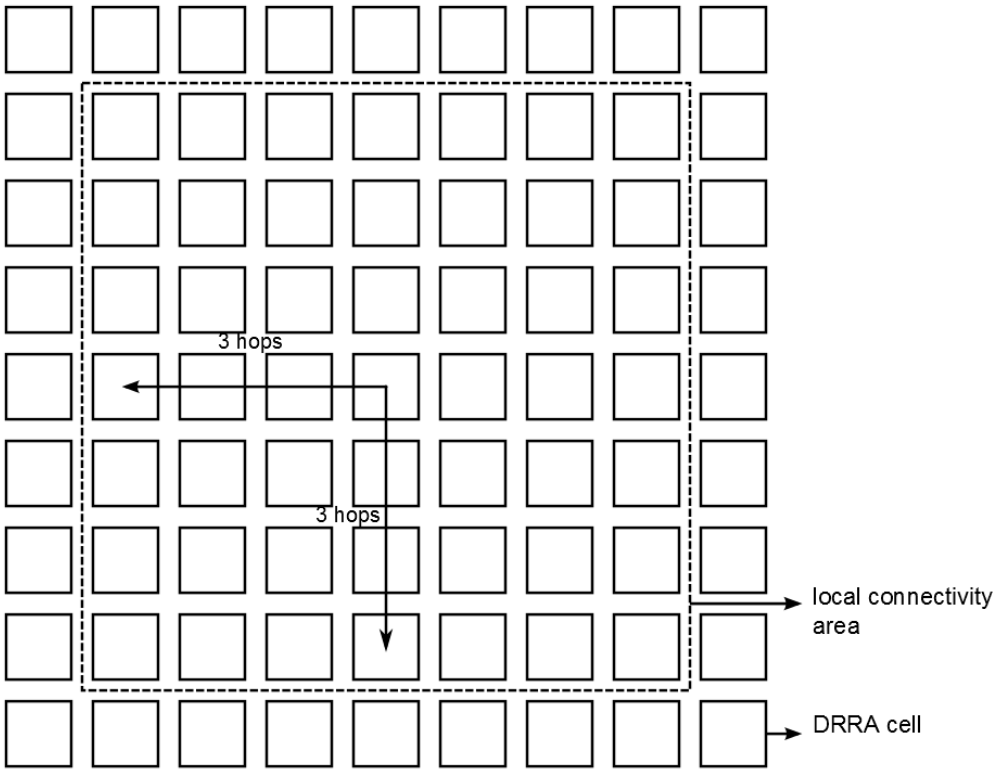

Figure 3.2: Switchbox connectivity area

As depicted in figure 3.2., DRRA fabric offers a seamless sliding window, for inter-cell communication. Components in a single DRRA cell may be connected to other DRRA cells' components 3 hops away columnwise and rowwise. This architecutre enables a 7x7 grid of cells communication without any extra interface rather than switchboxes. The advantage of this system is reduced communication overhead delay and reduced power consumption for communication within the local connectivity area.

3.4. Sequencer

Each DRRA cell as depicted in figure 3.1 has a sequencer block controlling the mDPU, register file and switch boxes in the cell. Sequencer can provide conditional or counter based looping and branching options, with the availability of 256 words program memory. All the instructions decoded from configware are executed in the sequencer. Therefore, programming the DRRA actually means programming the functionalities in the sequencer. The configware includes the instructions for mDPU, register file and the switch boxes.

After the execution, sequencer will send the corresponding instructions to the specific components. The programmer may only manipulate components within a specific DRRA cell. Therefore, different configware is needed for different DRRA cells.

However, hierarchical control signals are implemented in sequencer instructions, providing communication between different sequencers so that more complex tasks might be accomplished.

3.5. DRRA Design Flow and Application Setup

Designing a project on DRRA can be represented as a two step process. Firstly, the physical design step and secondly application mapping. According to physical specifications such as area, speed, power consumption, the physical designer may allocate any number of DRRA cells. According to the functional specifications, software designer can design the algorithms for the applications with the available instructions and DRRA library to produce a configware. This configware is an assembly like sequence providing instructions the sequencers. After the configware is produced by the programmer, a hardware compiler configures the DRRA according to this configware. At this point, the design is at *register transfer level* (RTL) and if there are any violations on the design specifications, the programmer or the physical designer may modify the configware or physical allocation.

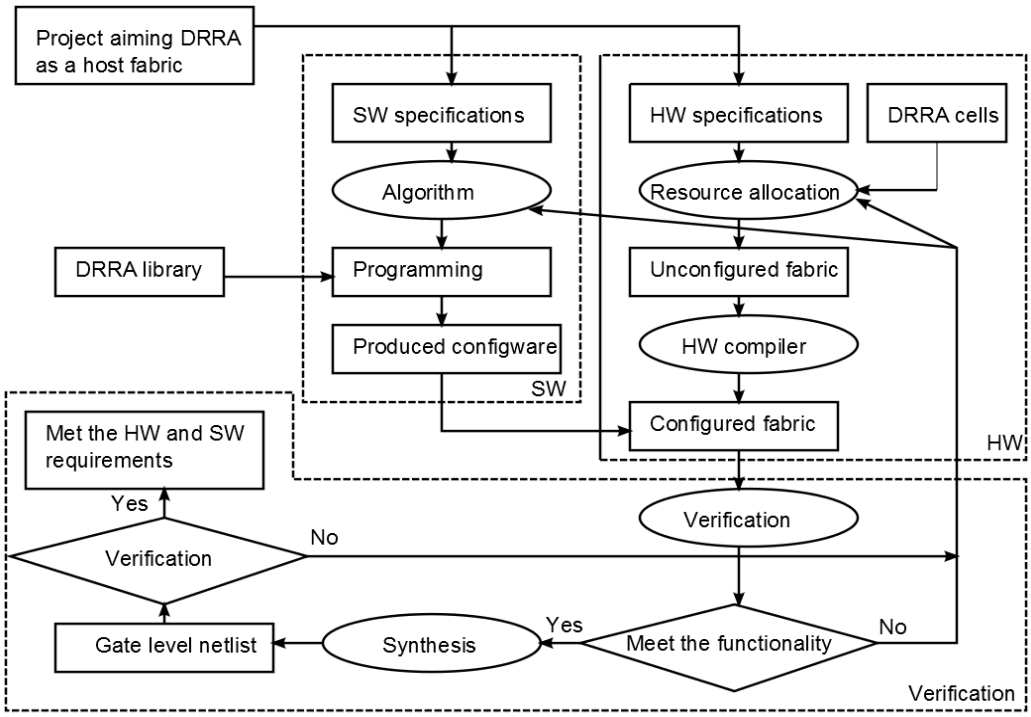

Figure 3.3: DRRA design flow

After the adjustment, the design is simulated again to verify to the specifications as well as functionality. If RTL level verification is successful, then gate level synthesis takes place. Gate level simulation is done afterwards for further verification and if there are any violations, modifications takes place again. Please see the design flow chart above.

3.6. The MANAS Tool

MANual ASsembly (MANAS) is a software tool developed in KTH for translating configware produced by the programmer into VHDL testbenches. In this section, details of the folders involved in MANAS tool will be revealed. Please refer to project folder tree structure in section 3.6. to see the folders involved in MANAS tool under configware.

- bin, the executable bin file;
- cfg, includes global constants needed to generate final test benches;
- seq, includes sequencer configwares in the form of seq*N*.txt files where *N* denotes the DRRA cell number ;
- output, contains the final test benches in 4 versions, namely;
    - tb_test_non_mem RTL.vhd;
    - tb_test_mem_RTL.vhd;
    - tb_test_non_mem gatelevel.vhd;
    - tb_test_mem_gatelevel.vhd.

  Four testbenches cover the cases of RTL or gate level and with or without SRAM. RTL level testbenches are used before the synthesis of the fabric. Gate level versions are used to test the design after the synthesis. Versions with SRAM support provide testing of the memory block if implemented. All four testbenches essentially the same functionality by inserting the instructions into the sequencers;
- instructions, includes the templates of the sequencer instructions;
- vhdl_sections, contains the templates of the four possible output test benches.

After providing seq*N*.txt files in seq folder, bin file is executed to automatically generate the testbenches. No argument is needed for the executable file but root authentication may be required. Depending on the memory requirements of the system, sample_data.vhd containing initial information on SRAM content can be provided as well. In this thesis, no SRAM is used for the applications, instead register files within DRRA cells are used as memory elements which will be explained in applications section in detail.

3.7. Design Folder Tree Structure

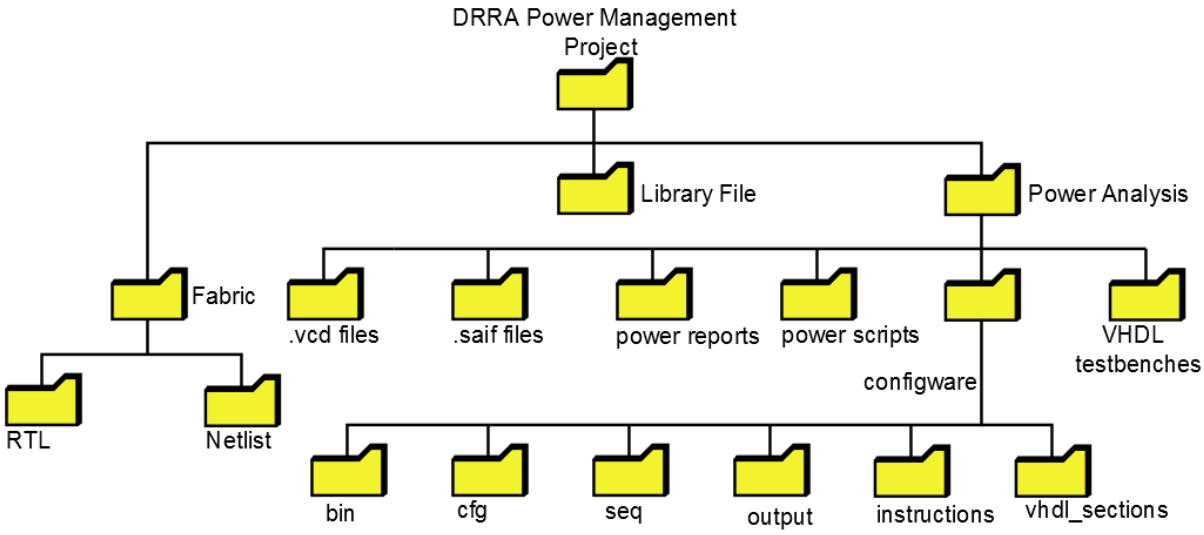

Figure 3.4: Design folder tree structure

The folder structure above contains all the files used and generated throughout the project. Library file contains TSMC 65nm multi Vdd library for the synthesis. Configware contains the MANAS tool and applications within the project. Power scripts contains Synopsys Design Vision scripts for both logic synthesis and power reporting. Value change dump (.vcd) and switching activity interchange format (.saif) contains the switching activity for the generated logic level netlists after Modelsim simulation of the different applications used for power analysis in this thesis work. For details, please refer to applications section. Power reports contains the final results.

3.8. Design Process Steps

This section of the thesis explains the process steps to reach the power results which are mentioned in the results section. This section includes DRRA resource allocation, DRRA DVFS implementation, synthesis and power results extraction respectively.

3.8.1. DRRA resource allocation

- Source code for the DRRA fabric is included within Fabric folder, please observe folder tree structure in figure 3.4 for this section.
- A new project is created under Modelsim,
- Within the RTL source code, drra_types_n_constants_std_logic.vhd file contains the constant used for the whole project. Among those constants, COLUMS and ROWS integer parameters represent how many rows and columns will be formed. To have the appropriate number of DRRA cells, these parameters are adjusted. 2x7 for example would results in a 14 cells architecture.
- Once the cell number is determined, the fabric becomes ready to be programmed in RTL level.

3.8.2. DRRA DVFS Implementation

- Under RTL folder, cgu_ozan.vhd file contains RTL code DVFS implementation.
- To implement DVFS circuitry, top level DRRA fabric needs to be modified as well.
- Extra signals introduced to implement DVFS functionality can be found under signal declaration in the top level commented as power signals.

3.8.3. Synthesis

- Synthesis script for Synopsys Design Compiler is included in Appendix G, important steps will be mentioned here.
- Firstly the paths for the synthesis libraries are included in the script.
- The paths are for the RTL codes involved in the design are set for analysis.
- Then the design is elaborated.
- Clock is created and frequency is set.
- False path rst_n is set because, critical path does not depend on reset signal.
- Power domain is created and supply voltage set.
- The synthesis takes place with compile command.
- After synthesis is completed, logic level netlist is stored for power analysis and standard delay format (.sdf) file is stored containing delay information of the standard cells.

- Please note that, for the design to be successfully tested, the inputs and outputs ports of the RTL modules in the design should be in std_logic type. Otherwise generated netlist and .sdf file will have unexpected information.

3.8.4. Power Results Extraction

- Once .sdf file and logic level netlist are generated with design compiler, they are added to the project under Modelsim. .sdf file can be added under Simulate tab in Modelsim. Once start simulate is hit, SDF tab can be seen. It is important that Disable SDF Warnings and Reduce SDF Errors to Warning checkboxes are checked.
- Testbenches produced by MANAS tool are added to the project to test the applications. All the files are compiled and simulated without optimization. Simulation time scale is adjusted to picoseconds under simulation options in modelsim.
- Value chain dump (.vcd) file contains the information on the switching activity of the design under test for a specified time period. This file is essential for accurate power estimation. A sample set of commands under Modelsim console can be used to store .vcd files:
    - run xxx ns
    - vcd file PATH/filename.vcd
    - vcd add -r tb_name/*
    - run xxx ns
    - quit –f
- After .vcd files are stored for the applications with different levels of parallelism for different operating voltages and frequencies, Design Compiler is used again to extract power results. To achieve that, .vcd files must be converted to switching activity interchange format (.saif) file for Design Compiler. Refer to Appendix G.
- Finally the netlist and .saif files are read in power is reported with report_power command.

Chapter 4

Design and Implementation

A power management system has been built on top of DRRA by introducing a globally ratio synchronous locally synchronous (GRLS) wrapper around every cell. Please see figure 4.1. below. The wrapper is used to ensure safe communication between nodes and to enable dynamic frequency and voltage scaling (DVFS) Please see Appendix E for DVFS functionality source code. The access point to provide the power services is given by the *Power Management Unit* (PMU), which uses *Voltage Control Unit* (VCU) and Clock Generation Unit (CGU) to control the voltage and the clock frequency, respectively, in each node. To support such fine-grained power islands, the NoC infrastructure adopts globally ratiochronous locally synchronous (GRLS) clocking (Chabloz and Hemani, 2010). In particular, all clocks on the chip run at frequencies which are submultiples of a certain *fH*. This restriction achieves a significant simplification in the implementation of synchronizes with low latency and overhead. Synchronizing registers (4 flip-flops per data line) are used between two different clock regions to reduce metastability, as suggested in (Chabloz and Hemani, 2010). Note that *Vs* and *fs* in the figure selected voltage and selected operating frequency respectively.

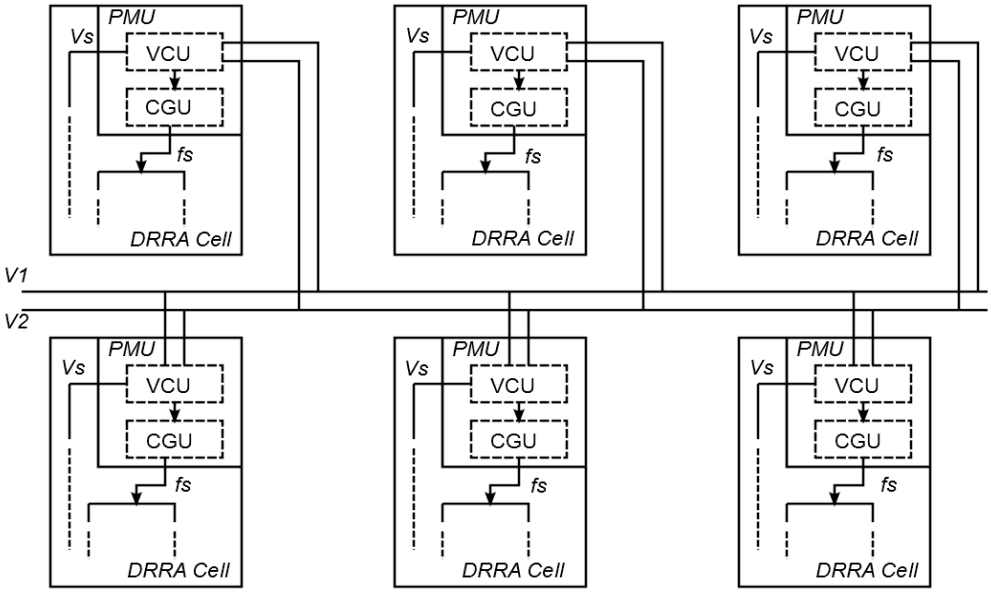

Figure 4.1: DVFS infrastructure in DRRA

4.1. Clock Generation Unit

For frequency scaling, we have used GRLS *Frequency Regulation System* (FRS) inspired from [20]. GRLS requires the frequencies of all local clocks to be *rationally-related*. Two or more frequencies are said to be rationally related if they are all submultiples of a common number, *Nc*. The hardware used for the frequency regulation system is shown in Figure 4.2 below. Once CGU receives selected clock from voltage control unit, it sets the output frequency with components counter, comparator and toggle FF. Note that comparator includes additional logic for duty cycle management which is not included in the simplified block diagram and it supports 1 to 15 division values of the input clock.

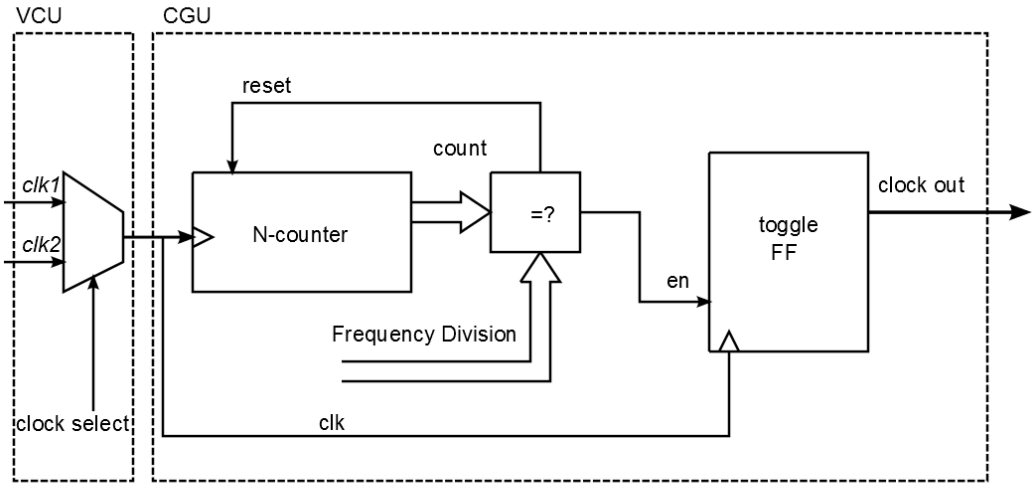

Figure 4.2: Clock generation unit (CGU)

4.2. Voltage Control Unit

The limitations of fixed supply voltage systems are recognized in literature and per-module voltage scaling has been proposed as an efficient alternative to address these limitations [25]. GRLS employs quantized supply voltage levels, i.e. multiple global supply voltages are generated on-chip or off-chip in a *Central Voltage Control Unit* (CVCU) and distributed throughout the chip using parallel supply voltage distribution grids.

In every module, a *Local Voltage Control Unit* (LVCU) is inserted containing PMOS power switches and the logic necessary to drive them. The power switches select one of the global supply voltages as the local supply voltage for the module. This allows quantized voltage regulation. In the figure, central VCU, which can be placed off-chip as well, powers the distribution grid with quantized voltage levels. Depending on the system requirements, *n* rails can be formed and distributed. For this work, 2 operating voltages were set as 1.32V and 1.1V. As depicted in the figure, local VCUs select *V1* and *V2* as their local operating voltages. In physical layer, this grid is realized as parallel multiple voltage rails with PMOS power switches beneath.

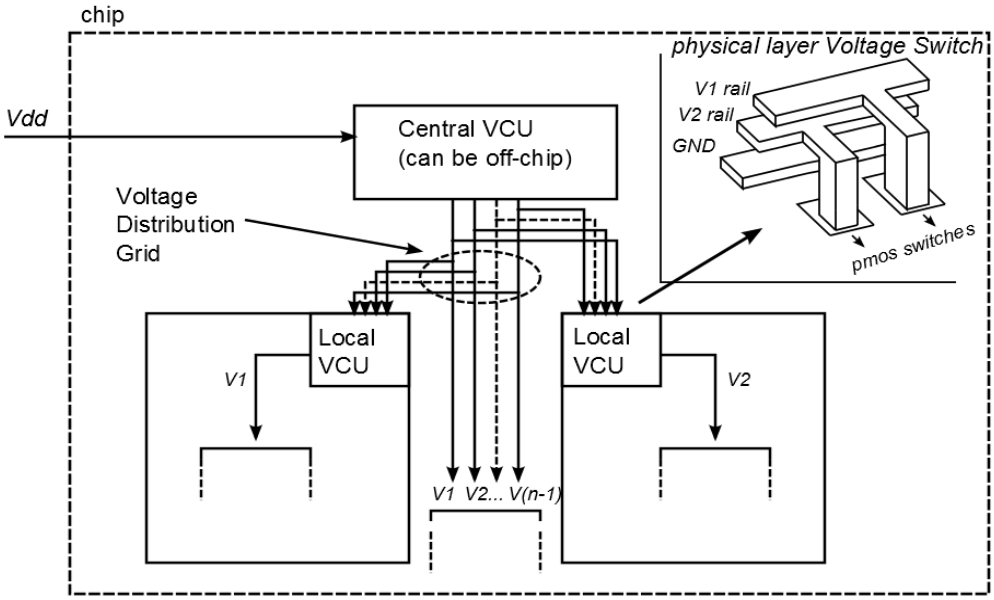

Figure 4.3: Voltage control unit (VCU)

4.3. Power Management Intelligence

Figure 4.4 demonstrates the power intelligence system. The control unit for the system resides in Leon3 processor as a high level software control layer for DRRA fabric. The loader in the figure has the role of interfacing AHB bus and DVFS circuit within each DRRA cell. DVFS circuit itself is a passive element in the design. Meaning that it waits inputs from the processor to change the frequency and voltage levels for particular DRRA cells.

Power intelligence scheme has four main functions, namely;

- Loading the configware from the configuration memory;
- Adjusting the configware to generate DRICs;
- Forming different frequency regions;
- Performing per cell DVFS algorithm.

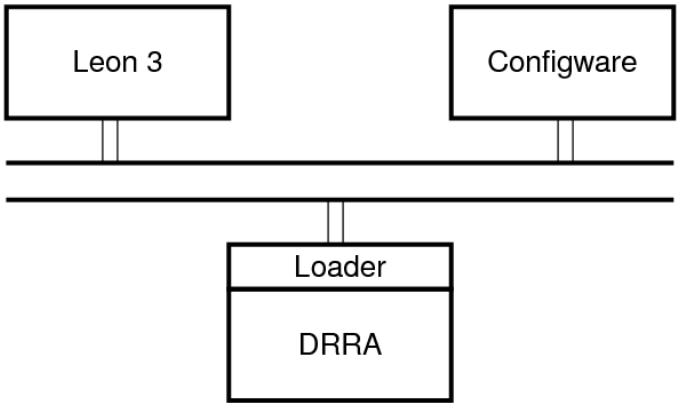

Figure 4.4: Per-cell DVFS for best effort power management with run-time performance monitoring

4.3.1. Dynamically Reconfigurable Isolation Cells (DRICs)

Dynamically reconfigurable isolation cells can be formed during runtime. Rather than implementing power hungry buffers to interface different frequency and voltage islands, a single DRRA cells can be programmed with an automatically generated configware depending on the initial application. At runtime, a soft binary of DRIC is stored. A DRIC can be placed anywhere on the fabric. DRICs are formed before the formation of different voltage regions. When it is detected that some regions have different operating frequencies, DRIC command is executed by the Leon3 processor.

4.3.2. Registerfile Instruction Generation

Once the DRICs are generated according to islands on the fabric with different operating frequencies, they need to be programmed for GRLS communication interface. Algorithm 1 below represents the regulation algorithm to determine the instructions.

A quick observation on algorithm 1 below will reveal that, depending on the different rationally related frequencies, send and wait sequence for data transfer will repeat itself, please see the example below. Function 1 finds the minimum period that send and wait statements will repeat as a sequence.

In a sense, the control code on Leaon3 processor translates into a data flow control after these the function 1 and algorithm 1are run, therefore, the translation to data flow control is a two step process. Calculating the minimum number of instruction with function 1 is particularly important as a first step because, number of instructions to be loaded into the DRICs determines how much the device will stay in idle state for programming.

In the generated sequence, a zero translate into a wait statement and one determines when the transmitter should send data. Zeros and ones are stored as "seq" and configware is generated according to this sequence using a look up table. If the sequence code is zero, a delay instruction generated and placed in the DRIC, if the sequence code is one then a refi (register file read write) instruction is generated and placed. For refi and delay instruction details please see Appendix D and E respectively. Each of these instructions are executed once and the sequence continues.

$$P = \min\left(N_r, N_t/HCF, N_r/HCF\right) \quad (1)$$

```
1.  Data: N_r, N_t
2.  Result: Array containing minimum sequence that needs to be stored
3.  initialization;
4.  P = min (N_r, N_t/HCF, N_r/HCF);
5.  if N_r ≤ N_t then
6.      send = 1;
7.  else
8.      for i from 0 to P do
9.          if c > N_r − N_t then
10.             send = 1;
11.             c = c − (N_r − N_t);
12.         else
13.             send = 0;
14.             c = c + N_t;
15.         end
16.         sendseq[i] = send;
17.     end;
18. end;
```

Algorithm 1: Regulation algorithm

Example sequence: To illustrate the algorithm, consider for example that $Nr = 20$ and $Nt = 8$ as frequency division values and hence their highest common factor, $HCF = 4$. The minimum sequence length calculated by the algorithm 1 will be 5. Now the algorithm 2 will fill the sequence data structure with 1010010100.... As a result, as shown in Figure 4.5, the net configware will contain 2 refi instructions and 3 delay instruction.

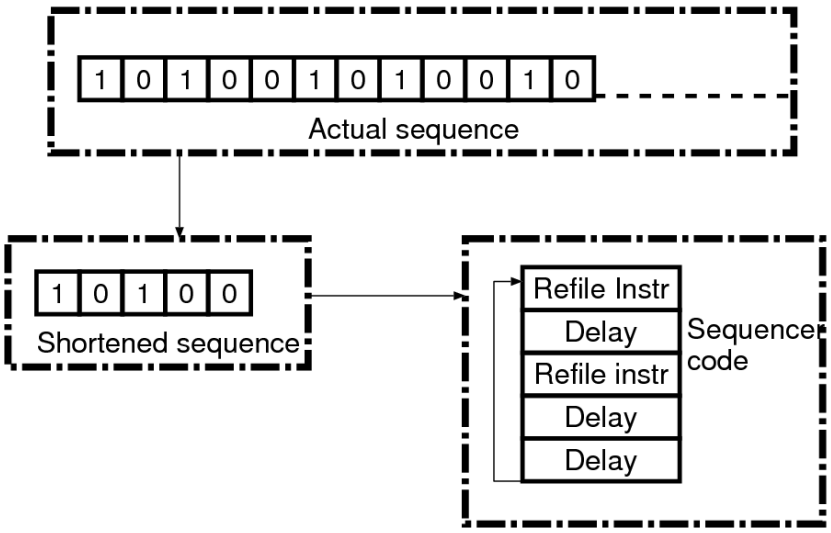

Figure 4.5: Generation of sequencer code from regulation algorithm

Forming DRICs according to frequency division values of different frequency islands and generating the sequence is not enough to form the GRLS interface. Interconnections between transmitter cell and DRIC and also between DRIC and receiver cell should be placed as well using switchbox instructions. In DDRA fabric, the interconnection code is placed in the destination sequencer. Therefore, to connect the transmitter side island to DRIC, the switchbox instruction is placed in DRIC cell itself. This is done automatically by Leon3 processor control after the DRICs are formed. To connect the DRIC to receiver side island, the switchbox instruction is placed in the receiver side sequencer. DRRA fabric does not have a word addressable sequencer, for that reason, whenever a DRIC is generated between islands working with different operating frequencies, the configware needs to be rewritten.

4.3.3. Multicasting Support

It is likely that DRRA cells executing a common application will be forced to reduce their frequency and operating voltage simultaneously. Therefore, multicasting is performed in our design to configure multiple cells in a single cycle. RowMultiC proposed by [27] is used to realize this scheme, please observe Figure 4.6.

The biggest advantage of this scheme is that the number of wires is significantly reduced. To realize this scheme, address decoder of the components need to be adjusted. In multicasting, each sequencer is assigned a unique ID depending on its row and column position in the fabric. Incoming address is compared with this unique ID to program the sequencers. To address a sequencer, 1 is placed for row and column indexes

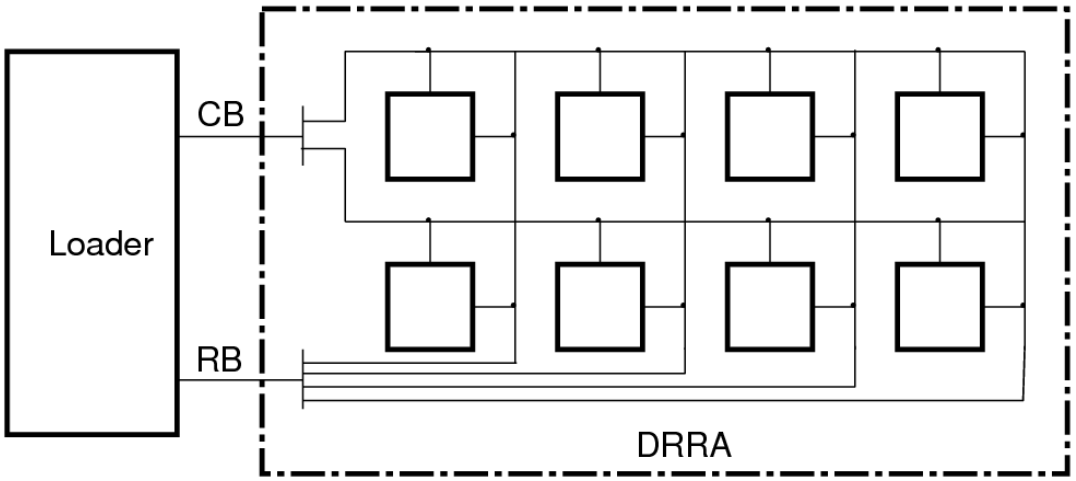

Figure 4.6: Multicast configuration

Multiple row and column indexes can be set to 1 realizing the multicasting scheme by addressing multiple sequencers at the same time. Overall overhead for implementing this scheme would be $(r + c)n$ bits where r represents the number of rows and c represents number of columns. Note that in the figure above, RB stands for row bus and CB stands for column bus.

4.3.4. Autonomous Parallelism Voltage and Frequency Selection (APVFS)

Best effort per-cell DVFS concept is applied on existing DRRA fabric. Depending on the runtime conditions, APVFS algorithm selects the best voltage/frequency/version triplet for the specific DRRA cells to reduce power.

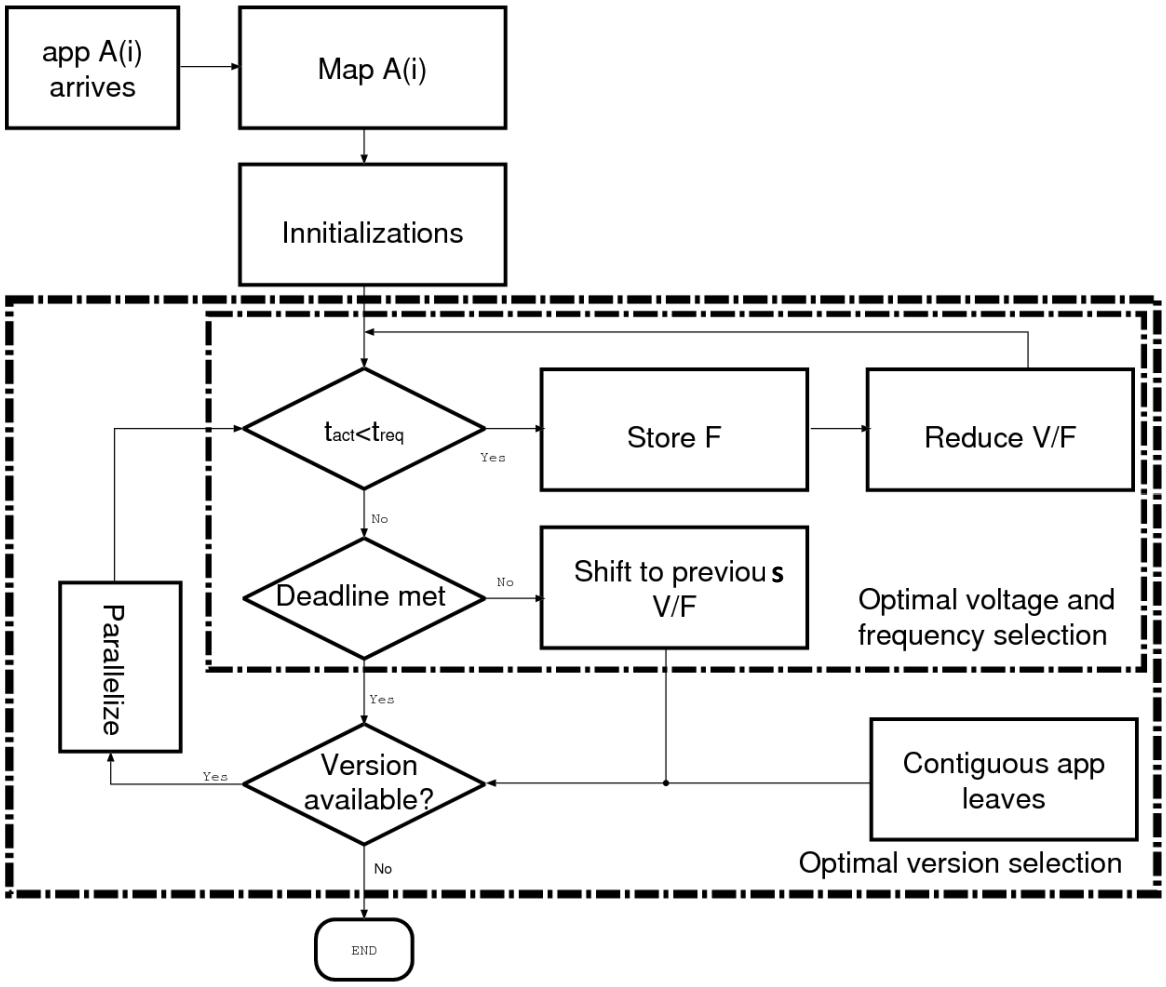

Figure 4.7: APVFS algorithm

This adaptive power management using distributed DVFS with runtime performance monitoring is depicted in figure 4.7. This process consists of 3 steps. Firstly initialization of voltage and frequency is applied and the latency constraint is set. Secondly runtime information is gathered in terms of workload and latency. Finally, if the latency is lower than the constraint, DVFS is applied to the switch as well as available parallel version selection.

Chapter 5

Results

In this section on the thesis, potential reduction in power and energy consumption is formalized. After that results are presented.

5.1. Formal Evaluations

Equation 1 below denoted the total energy of a system, running *A* number of applications simultaneously, $E_0(i)$ being the energy consumed by i^{th} application.

$$E_t = \sum_{i=1}^{A} E_0(\mathrm{i}) \qquad (1)$$

Total energy for a system employing DVFS functionality, total energy would be given by;

$$E_{dvfstotal} = \sum_{i=1}^{A} E_{dvfs}(i) \qquad (2)$$

Initially, if we assume that all the applications are running at the same frequency, the relation between E_0 and E_{dvfs} would be;

$$E_{dvfs} = E_0 + E_{tx} + E_{rx} + E_{sync} \qquad (3)$$

Where E_{tx} , E_{rx} , E_{sync} denotes data transmission, data reception and synchronization energy respectively. E_{tx} in this equation is a major components due to power hungry buffers. But yet, this thesis offers reconfigurable isolation cells and regions are formed whenever only they are needed. Therefore, the outcome of equation number is 2 is expected to be less than equation number one as a results of small overheads.

Dynamic power consumption is of an application is given by the commonly used equation number 4.

$$P_{dynamic} = C_{eff} + f_{operating} + V_{dd}^2 \quad (4)$$

Where C_{eff} , $f_{operating}$, V_{dd} denotes effective charged capacitance, operating frequency and operating voltage respectively. $E_0(f, V_{dd})$ then would be the dynamic power of each application as a function of frequency and voltage. Therefore, it is feasible to lower the frequency and voltage whenever it is possible for less dynamic power.

5.2. Simulations

First of all, 1.1V and 1.32V are selected as quantized voltage levels for this thesis to apply dynamic voltage scaling. Then, Sysnopsys Design Compiler is used to synthesize the DRRA fabric using 65 nm multi-Vdd technology library. Please see Appendix G for sample synthesis and power analysis script. Table 5.1 below reveals that, DRRA fabric can run up to 1.4 GHz frequency at 1.32V operating voltage and up to 1 GHz frequency at 1.1V operating voltage without any timing violation. Allowable selected frequencies for GRLS communication scheme are 1400, 1000, 500, 333, 250 and 125 MHz to employ per-cell DVFS with runtime performance monitoring in Figure 4.6.

Voltage (V)	Frequency (MHz)	Timing Constraint
1.32	1400	violated
1.32	**1200**	**met**
1.1	1200	violated
1.1	1100	violated
1.1	2400	violated
1.1	**1000**	**met**
1.1	500	met
1.1	333	met
1.1	250	met
1.1	125	met

Table 5.1: Voltage frequency pairs

To reveal energy and power savings of our proposed system, three common digital signal processing application algorithms are mapped on DRRA fabric, namely; matrix multiplication (MM), Fast Fourier Transform (FFT), and Finite Impulse Response (FIR).

Three versions of matrix multiplications are available (fully serial, partially parallel and fully parallel). Please refer to Appendix F for example fully serial matrix multiplication configware. To reveal the power and energy consumption measurements, switching activity files were recorded and analysis was performed using Synopsys Design Compiler. Below Table 5.2 illustrates the power and energy results of applied APVFS algorithm. Note that *parpar* application version stands for partially parallel.

Voltage	**Frequency (MHz)**	**Application**	**Time (picoseconds)**	**Power W**	**Energy**
Matrix multiplication enters to platform					
V1	1200	MM serial	48480	13.56	657388.8
V2	1000	MM serial	49980	10.55	527289.0
V2	500	MM serial	122500	4.85	594125.0
V2	500	MM parpar	69650	4.13	287654.5
V2	333	MM parpar	102660	2.92	299767.2
V2	333	MM parallel	78450	3.7	290265.0
V2	250	MM parallel	103940	2.37	246337.8
V2	333	MM parallel	7850	3.7	290265.0
FIR enters to platform					
V1	1200	FIR serial	104690	13.8	1444722.0
V2	1000	FIR serial	128930	11.05	1424676.5
V2	500	FIR serial	258840	5.59	1446915.5
V2	1000	FIR serial	128930	11.05	1424676.5
FFT enters to platform					
V1	1200	FFT serial	61400	13.5	828900.0
V2	1000	FFT serial	74970	11.13	834416.1
V2	500	FFT serial	178730	5.58	997313.4
V2	1000	FFT serial	74970	11.13	834416.1

Table 5.2: Power and energy simulation results

Initially 1.4 GHz and 1.32V were assigned to the chip knowing that this pair is not meeting the deadline from Table 5.1. Then APVFS algorithm was applied to iterate for the best voltage/frequency/version triplet.

Third column in the table represents the application and available versions. After an application is run, the time execution time was monitored to see if the timing deadline is violated or not. If the deadline was not violated, next frequency/voltage pair was applied to system according to Table 5.1. After that, if the deadline is not met with the adjusted frequency/voltage pair, available parallel version of the application was applied. This iteration continues until the most parallel operating with the smallest possible frequency/voltage pair is selected. Below Table 5.3 represents the reductions in net power and energy in terms of percentages.

Iteration	Operation	Power Reduction	Energy reduction
Matrix multiplication enters to platform			
1	reduce V&F	0.00	0.00
2	reduce F	22.20	19.79
3	reduce F	64.23	9.62
4	change version	69.54	56.24
5	reduce F	78.47	54.40
6	change version	72.71	55.85
7	**reduce F**	**82.52**	**62.53**
8	increase F	72.71	55.85
FIR enters to platform			
9	reduce V&F	36.04	17.46
10	reduce F	46.09	18.42
11	**reduce F**	**66.05**	**17.36**
12	increase F	46.09	18.42
FFT enters to platform			
13	reduce V&F	30.86	13.21
14	reduce F	36.66	13.02
15	**reduce F**	**50.24**	**7.46**
16	increase F	36.66	13.02

Table 5.3: Reduction in power and energy

Rows depicted bold style in Table 5.3 represent the points where the deadlines for the specific application were violated. As a results, in the next iteration previous frequency value is set. It can be also be observed in the table during the iteration just before the deadline is violated, upto 72% power reduction and upto 55% energy reduction is achieved with the proposed method. Below Figures 5.2.1 and 5.2.2 depict the reduction energy and power for MM, FIR and FFT applications. Please observe that after applying APVFS algorithm to DRRA fabric, energy and power values iterates towards the minimum possible value.

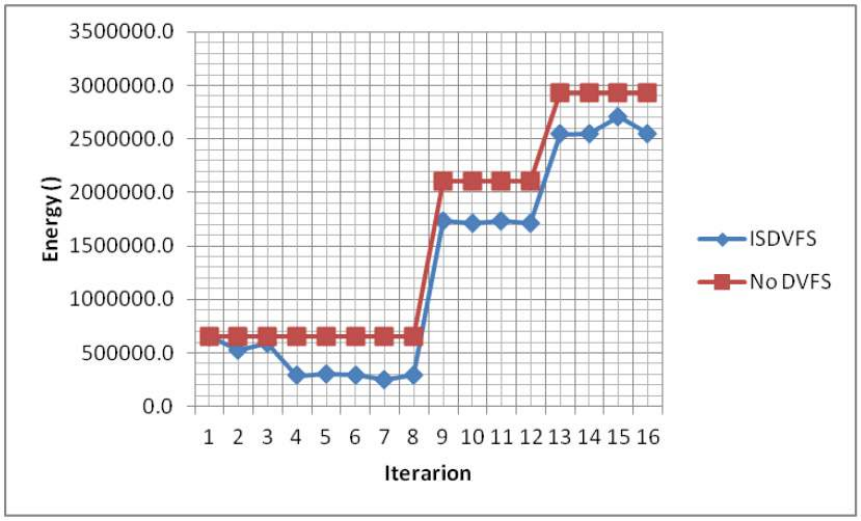

Figure 5.1: Energy reduction graphical representation

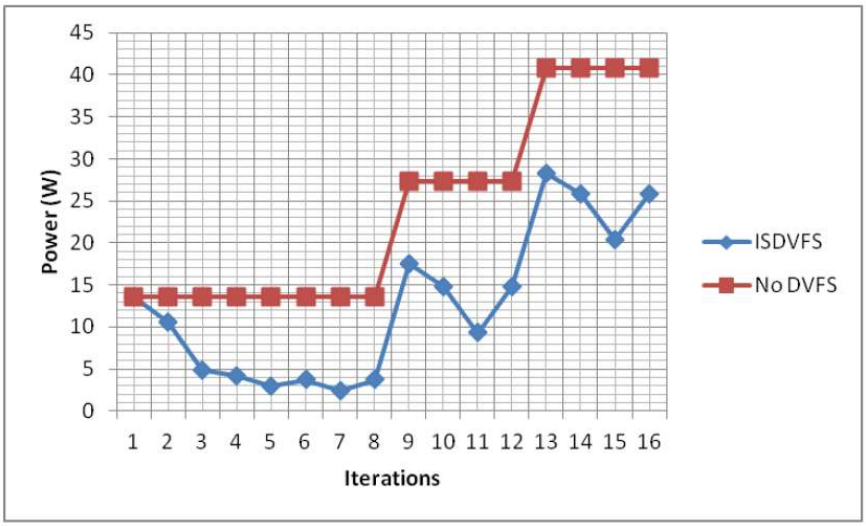

Figure 5.2.2. Power reduction graphical representation

5.2.1. Overhead analysis

In this section of the thesis, area and power values of traditional DVFS (TDVFS) approach and DVFS with reconfigurable isolations cells (ISDVFS) are compared according to varying number of cells and regions.

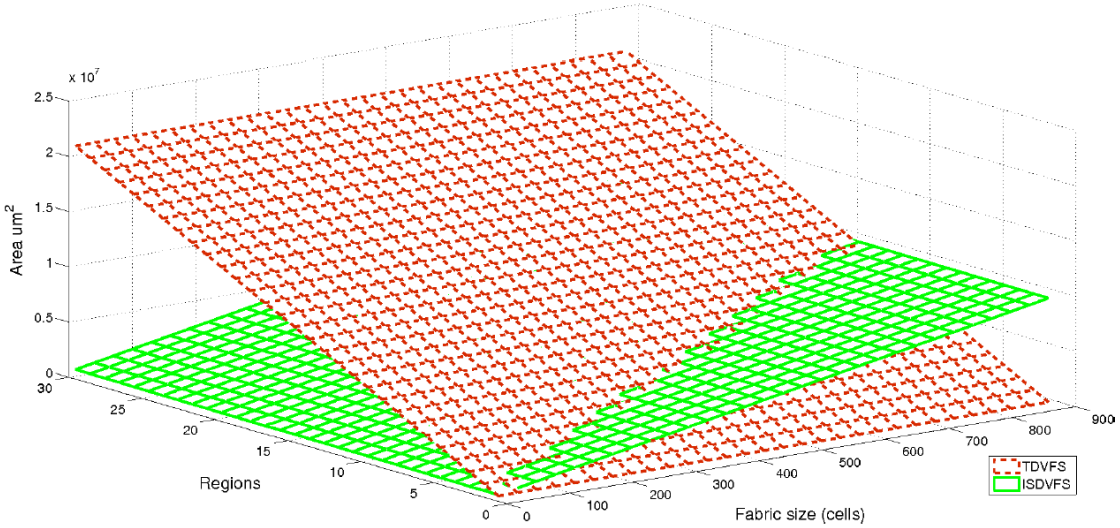

Figure 5.3: Area overhead comparison

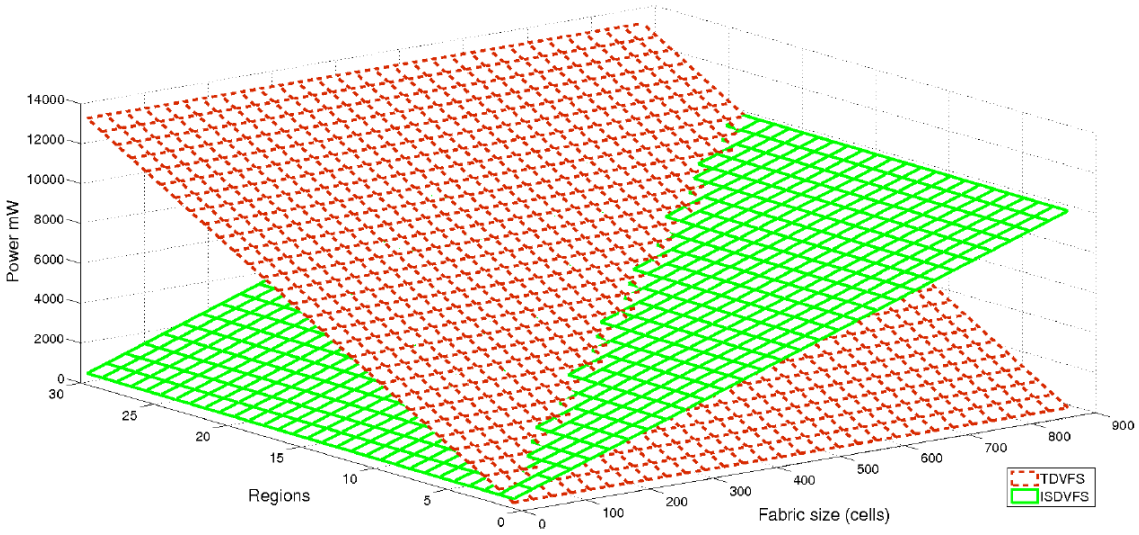

Figure 5.4: Power overhead comparison

It can be observed in above Figures 5.2.3 and 5.2.4 that the area and power overhead increases for TDVFS with the addition of cells.

However, increasing the number of cells has a negligible affect on the ISDVFS overheads. Number of regions is the more determining factor for the ISDVFS overheads. Number of cells compared to number of regions to employ a particular application is considered to be much smaller. In other words, many cells can be used within a few number of regions depending on the application specification. This reveals that, our reconfigurable isolation cells approach offers significant reduction in terms of area and power overheads over the TDVFS approach.

Chapter 6

Conclusion and Future Work

In this thesis work, a power management architecture and its implementation for Coarse Grained Reconfigurable Architectures (CGRAs) was presented. Firstly, background and related work analysis was performed followed by GRLS communication scheme. Secondly, DRRA fabric details were presented as a host for proposed solution and applications. Finally design and implementation details were presented followed by the results.

It was revealed that power and area overheads compared to conventional DVFS methods were considerably decreased by Dynamically Reconfigurable Isolation Cells (DRICs). Autonomous parallelism, voltage and frequency selection (APVFS) algorithm was proposed to test the benefits of our proposed solution, recalling that APVFS algorithm will dynamically check for applications deadlines and will adjust the voltage/frequency/version triplet selection accordingly. To employ APVFS algorithm, matrix multiplication, FIR and FFT applications were used, with matrix multiplication having three available versions in different levels of parallelism.

Overall, simulations results showed that up to 72% reduction in power and up to 55% reduction in energy can be achieved with ISDVFS scheme over completely synchronous design style. It was also presented that ISDVFS offers negligible overheads over the traditional DVFS approach.

As a future work, APVFS algorithm can be further investigated with other applications to reveal the efficiency. Furthermore, DRRA fabric may also be manufactured and physical chip can be used on a dedicated printed circuit board (PCB) as a reconfigurable platform to have more accurate measurements. It is also worth mentioning that producing configware for DRRA fabric requires a lot of attention of the programmer to carefully choose the parameters involved in the instructions. A compiler might be designed to further simplify the process of producing configware.

Appendix

A. mDPU (MDPU) instruction

Field name	Length in bits	Description
instr_code	4	Value 2 indicates it is an mDPU instruction
mdpumode	4	Specifies the mode of operation
count	10	Reserved bits, set value 0 for mDPU instruction
saturation	4	It specifies which are used for saturation
outb	2	It specifies where to send data from output port A
outb	2	It specifies where to send data from output port B
acccount	8	Specifies the number of cycles after which the accumulation counter is cleared
dpuunused	2	Reserved bit, set value 0 for mDPU instruction

Example: As a reference for analyzing configware codes given in Appendix F and G, a single line code for mDPU instruction will be explained here.

MDPU mdpuinstr 2 0 0 3 3 11 0;

Above example line of configware code uses " mdpuinstr" as reserved word to indicate that it is an mDPU instruction. It specifies the mode of operation as 2 corresponding to;

out0 = in0 * in1

out1 = (in0 * in1) + acc

Please recall from section 3.1 that mDPU has four inputs and two outputs. In this particular example code, in2 and in3 are not in use, which can be useful depending on the application to be programmed. After the mode of operation, count and saturation values are set to 0 meaning that they are not being used. Following that, outa and outb values are set to 3, realizing that outputs will be streamed to the left and right output busses in DRRA fabric at the same time. It could be the case that programmer wants to send the data to one direction only, then either 1 or 2 would be set for only right and only left respectively. 11 in the example code denotes to clear "acc" (accumulator) after 11 cycles and finally unused bit is set to 0.

B. Registerfile (REFI) instruction

Part 1

Field name	Length in bits	Description
instr_code	4	It specifies the type of instruction
porttype	2	It specifies (a) Which port to use in register file, as there are two ports available (b) It specifies the RD/WR
ivalid	1	Unused
mode	1	It specifies the addressing mode to be used
startadrs	6	It specifies the starting address in register file
endadrss	6	It specifies the end address
incrdcr	1	It specifies if the address should be incremented or decremented
incr_dcr_value	5	It is the offset value.
initial delay	6	It delays the execution by N cycles
output control	2	It specifies which port should be enabled for sending the data
instruction complete	1	It specifies if the instruction has a 2^{nd} part or not
infinite loop	1	If enabled the instruction will repeat itself infinite times

Part 2

Field name	Length in bits	Description
instr_code	4	It specifies the type of instruction.
i_rpt_ty	2	Set value 0 to disable repetition, set value 3 to enable repetition for both parts
repetition delay	6	Delay between repeating the instructions
number of repetitions	6	How many times this instruction should be repeated.
rept_incr_dcr	1	It specifies if the address should be incremented or decremented
rept_incr_dcr_value	5	It is the offset value
middle delay	6	Specifies the delay between two successive address generations
range counter	6	Set the range to count in terms of cycles

C. Switchbox connect (CONNECT) instruction

Field name	Description
instr_code	It specifies the type of instruction.
source	The keywords could either be "REFI" for register files, or "DPU" for mDPUs.
source index	Specifies the connection object address.
port selection	Keyword for port type. Port type could either be "OUT" or "IN".
port index	Specifies which port the object is using for connection. For "OUT", it could either be 0 or 1. For "IN", it could be 0, 1, 2, 3.
to	Keyword "TO".
destination	The keywords could either be "REFI" for registerfiles, or "DPU" for mDPUs.
destination index	Specifies the connection object address.
port selection	Keyword for port type. Port type could either be "OUT" or "IN".
port index	Specifies which port the object is using for connection. For "OUT", it could either be 0 or 1. For "IN", it could be 0, 1, 2, 3.

D. Delay (DELAY) instruction

Field name	Length in bits	Description
instr_code	4	It specifies the type of instruction.
number of cycles	7	Specifies N number of cycles.
unused bits	25	Unused, set value 0.

E. DVFS RTL code

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.all;
3.  use IEEE.std_logic_unsigned.all ;
4.  use work.drra_types_n_constants.all;
5.  use IEEE.numeric_std.all;
6.  USE IEEE.STD_LOGIC_ARITH.ALL;

7.  entity vcgu is
8.          generic (cell_col : integer range 0 to COLUMNS - 1  := 0;
9.          cell_row : integer range 0 to ROWS - 1  := 1);
10.         port (clk1, clk2: in std_logic;
11.         reset: in std_logic;
12.         freq_div: in std_logic_vector (3 downto 0) := "0010" ;
13.         clk_select: in std_logic_vector(1 downto 0) := "01";
14.         clk_out: out std_logic:= '0';
15.         rowbus_pmu : in std_logic := '1' ;
16.         colbus_pmu : in std_logic := '1' );
17. end vcgu;
18. architecture behave of vcgu is
19.         signal counter_en, clk_outs1, clk_outs2: std_logic;
20.         signal counter1, counter2: std_logic_vector (3 downto 0) := "0000";
21.         begin
22.         process (clk1,colbus_pmu,rowbus_pmu,clk_select)
23.         begin
24.         if (colbus_pmu = '1' and rowbus_pmu = '1') then
25.                 case clk_select is
26.                 when "00" =>
27.                 counter_en <= '0';
28.                 when "01" =>
29.                 counter_en <= '1';
30.                 when others  =>
31.                 counter_en <= '1';
32.                 end case;
33.         end if;
34.         end process;
35.         process (clk1,reset, freq_div)
36.         begin
37.                 if(reset= '0') then
38.                 counter1 <="0000";
39.                 clk_outs1 <= '0';
40.                 elsif rising_edge(clk1) and (counter_en = '0') and (freq_div /= "0000") then
41.                 counter1 <= counter1 + "0001";
42.                 if (counter1 = freq_div  - "0010") then
43.                 clk_outs1 <= '1';
44.                 elsif (counter1 = freq_div - "0001") then
45.                 counter1  <= "0000";
46.                 clk_outs1 <= '0';
47.                 end if;
48.                 end if;
49.                 end process;
```

```
50.         process (clk2, reset, freq_div)
51.         begin
52.                 if(reset= '0') then
53.                 counter2 <="0000";
54.                 clk_outs2 <= '0';
55.                 elsif rising_edge(clk2) and (counter_en = '1') and (freq_div /= "0000") then
56.                 counter2 <= counter2 + "0001";
57.                 if (counter2 = freq_div - "0010") then
58.                 clk_outs2 <= '1';
59.                 elsif (counter2 = freq_div - "0001") then
60.                 counter2  <= "0000";
61.                 clk_outs2 <= '0';
62.                 end if;
63.                 end if;
64.         end process;
65.         process (clk_outs1, clk_outs2, clk1, freq_div )
66.         begin
67.                 if (freq_div = "0000") then
68.                 clk_out<=clk1;
69.                 else
70.                 if(clk_outs1 = '1') or (clk_outs2 = '1') then
71.                 clk_out <= '1';
72.                 elsif (clk_outs1 = '0') or (clk_outs2 = '0') then
73.                 clk_out <= '0';
74.                 else
75.                 clk_out <= clk1;
76.                 end if;
77.                 end if;
78.         end process;
79. end behave;
```

F. Fully serial matrix multiplication configware

Sequencer 0

```
1.  USE SEQ 0
2.  INTRP 0;
3.  INTRP 0;

4.  CONNECT REFI 0 OUT 0 TO LATA 0 IN 0;
5.  CONNECT REFI 0 OUT 1 TO LATA 0 IN 1;

6.  /* read from port A Address: [0, 0] */

7.  RFILE1 rfileinstr 0 0 0 0 3 1 1 1 3 0 0;
8.  RFILE2 rfileinstr 3 0 4 1 0 0 0;

9.  /* read from port B Address: [0, 0] */

10. RFILE1 rfileinstr 1 0 0 4 20 1 1 0 3 1 0;


11. /*--------------------------------execution instruction--------------------------*/
12. /*  (y0, y1) = [x2 + (x0 * x1), x2 - (x0 * x1)] */

13. MDPU mdpuinstr 2 0 0 3 3 0 0;
```

Sequencer 1

```
1.  USE SEQ 1
2.  INTRP 0;
3.  INTRP 0;

4.  CONNECT LATA 0 OUT 0 TO REFI 1 IN 0;
5.  CONNECT LATA 1 OUT 0 TO REFI 1 IN 1;
6.  CONNECT REFI 1 OUT 0 TO LATA 1 IN 0;
7.  CONNECT REFI 1 OUT 1 TO LATA 1 IN 1;


8.  /* write from port A Address: [0, 0] */
9.  RFILE1 rfileinstr 2 0 0 0 15 1 1 3 3 1 0;

10. /* write from port A Address: [0, 0] */
11. RFILE1 rfileinstr 3 0 0 17 20 1 1 6 3 0 0;
12. RFILE2 rfileinstr 3 0 4 1 0 3 0;

13. /* read from port B Address: [0, 0] */
14. RFILE1 rfileinstr 0 0 0 0 15 1 1 0 3 0 0;
15. RFILE2 rfileinstr 3 0 0 1 0 3 0;

16. /* read from port B Address: [0, 0] */
17. RFILE1 rfileinstr 1 0 0 16 19 1 1 1 3 0 0;
18. RFILE2 rfileinstr 3 0 4 1 0 3 0;


19. /*--------------------------------execution instructions--------------------------*/
20. /*  (y0, y1) = [x2 + (x0 * x1), x2 - (x0 * x1)] */

21. MDPU mdpuinstr 12 0 0 3 3 0 0;
```

G. Synthesis and Power Analysis Script

Synthesis

```
set designer "Syed Mohammad Asad Hassan Jafri & Ozan Zeki Bag"
set company "KTH"
set SynopsysHome [getenv "SYNOPSYS"]
set search_path "../tcbn90g_110a"

set target_library "/home/ozan/Desktop/power_results/lowpower65nm/tcbn65lpbc.db
/home/ozan/Desktop/power_results/lowpower65nm/tcbn65lpbc1d1.db
/home/ozan/Desktop/power_results/lowpower65nm/tcbn65lpbc1d11d1.db
/home/ozan/Desktop/power_results/lowpower65nm/tcbn65lpbc1d11d32.db
/home/ozan/Desktop/power_results/lowpower65nm/tcbn65lpbc1d321d1.db
/home/ozan/Desktop/power_results/lowpower65nm/tcbn65lpbc1d321d32.db"

set link_path "$search_path"
set link_library "* $target_library "
remove_design -designs
power_preserve_rtl_hier_names
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/drrafabric/misc.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/drra_types_n_constants_std_logic.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/drrafabric/agu.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/cgu_ozan.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/regfile_stdlog.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/config_swb_stdlog.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/drrafabric/TristateMux_unregistered.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/samplemanas/vhdl_sections/MANAS_PACKAGE.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/drrafabric/switchbox.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/drrafabric/fifo.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/txt_util.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/mDPU_new_stdlog.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/toplevel_regfile_stdlog.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/sequencer_stdlog.vhd
analyze -library WORK -format vhdl /home/ozan/Desktop/implementation/grlib-gpl-1.1.0-b4108/designs/leon3mp/drra_fabric_stdlogic.vhd
elaborate FABRIC -architecture RTL -library DEFAULT

create_clock -name "clk" -period 0.41 -waveform { 0.0 0.20  }  { clk  }
set_dont_touch_network [ find clock clk]
set_fix_hold clk
set_false_path -from [get_port rst_n]
```

30. create_power_domain PD_SODIUM
31. create_supply_port VDD1
32. create_supply_port VSS
33. create_supply_net VN1 -domain PD_SODIUM
34. connect_supply_net VN1 -ports {VDD1}
35. create_supply_net GN -domain PD_SODIUM
36. connect_supply_net GN -ports {VSS}
37. set_domain_supply_net PD_SODIUM -primary_power_net VN1 -primary_ground_net GN
38. set_voltage 1.32 -object_list VN1
39. set_voltage 0.0 -object_list GN
40. set_operating_conditions BCCOM
41. compile -map_effort medium -incremental_mapping

42. write -hierarchy -format verilog -output /home/ozan//Desktop/power_results/sdf_files/v1_2400_14cells_std.v
43. write_sdf /home/ozan//Desktop/power_results/sdf_files/v1_2400_14cells_std.sdf

Report Power

1. vcd2saif -i myvcdfile.vcd -o mysaiffile
2. read_file -format verilog {/home/ozan/Desktop/power_results/netlists/v1_2400_14cells.v}
3. elaborate FABRIC -architecture RTL -library DEFAULT
4. read_saif -input /home/ozan/Desktop/power_results/saif_files/v1_2400.saif -instance_name tb_mode7
5. report_power -analysis_effort low > /home/ozan/Desktop/power_results/results/v2_1500MHz.rep

References

[1] Moore, G. E.; , "Cramming more components onto integrated circuits," Electronics Magazine , vol. 38, no. 8, Apr 1965

[2] Bohr, M.; , "The new era of scaling in an SoC world," Solid-State Circuits Conference - Digest of Technical Papers, 2009

[3] Z. ul Abdin and B. Svensson, "Evolution in architectures and programming methodologies of coarse-grained reconfigurable computing," *Microprocess. Microsyst.*, vol. 33, no. 3, pp. 161–178, May 2009. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2008.10.003

[4] W. J. Dally and J. W. Poulton, *Digital System Engineering*, W. J. Dally and J. W. Poulton, Eds. Cambridge University Press, 1998.

[5] M. A. Shami and A. Hemani, "Morphable dpu: Smart and efficient data path for signal processing applications," in *SiPS*, 2009, pp. 167–172.

[6] R. Airoldi, F. Garzia, and J. Nurmi, "Improving reconfigurable hardware energy efficiency and robustness via dvfs-scaled homogeneous mpsoc," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 286–289.

[7] L. Guang, E. Nigussie, and H. Tenhunen, "Run-time communication bypassing for energy-efficient, low-latency per-core dvfs on networkon- chip," in *SOC Conference (SOCC), 2010 IEEE International*, sept. 2010, pp. 481 –486.

[8] R. Lysecky, "Low-power warp processor for power efficient highperformance embedded systems," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, april 2007, pp. 1 –6.

[9] J. Kim, S. Yoo, and C.-M. Kyung, "Program phase and runtime distribution-aware online dvfs for combined vdd/vbb scaling," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 417 –422.

[10] G. Rauwerda and G. Smit, "Implementation of a flexible rake receiver in heterogeneous reconfigurable hardware," in *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, dec. 2004, pp. 437 – 440.

[11] V. Nollet and D. Verkestt, “A quick safari through the mpsoc run-time management jungle,” in *Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on*, oct. 2007, pp. 41 –46.

[12] M. Galanis, G. Dimitroulakos, and C. Goutis, “Mapping dsp applications on processor/coarse-grain reconfigurable array architectures,” in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, may 2006, p. 4 pp.

[13] N. Abbas, “Runtime parallelisation switching for MPEG4 encoder on MPSoC,” Master’s thesis, Royal Institute of Technology, 2007.

[14] C. Ykman-Couvreur, E. Brockmeyer, V. Nollet, T. Marescaux, F. Catthoor, and H. Corporaal, “Design-time application exploration for mp-soc customized run-time management,” in *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, nov. 2005, pp. 66 –69.

[15] C. Ykman-Couvreur, V. Nollet, T. Marescaux, E. Brockmeyer, F. Catthoor, and H. Corporaal, “Pareto-based application specification for mp-soc customized run-time management,” in *Embedded Computer Systems: Architectures, Modeling and Simulation, 2006. IC-SAMOS 2006. International Conference on*, july 2006, pp. 78 – 84.

[16] S. Jafri, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, “Compact generic intermediate representation (cgir) to enable late binding in

[20] J.-M. Chabloz and A. Hemani, “Distributed dvfs using rationally-related frequencies and discrete voltage levels,” in *ISLPED*, 2010, pp. 247–252.

[21] Messerschmitt, D.G.; , "Synchronization in digital system design," Selected Areas in Communications, IEEE Journal on , vol.8, no.8, pp.1404-1419, Oct 1990

[22] Mu, F.; Svensson, C.; , "Self-tested self-synchronization circuit for mesochronous clocking ," Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on , vol.48, no.2, pp.129-140, Feb 2001

[23] Muttersbach, J.; Villiger, T.; Fichtner, W.; , "Practical design of globallyasynchronous locally-synchronous systems," Advanced Research in Asynchronous Circuits and Systems, 2000. (ASYNC 2000) Proceedings. Sixth International Symposium on , vol., no., pp.52-59, 2000

[24] J. M. Chabloz and A. Hemani, ”A Flexible Interface for Rationally- Related Frequencies,” ICCD 2009

[25] J. M. Chabloz, ”Distributed DVFS with Rationally-Related Frequencies and Quantized Voltage Levels,” ISLPED 2010

[26] A. Hemani, DRRA Summary, 2009.

[27] V. Tunbunheng, M. Suzuki, and H. Amano, “Romultic: fast and simple configuration data multicasting scheme for coarse grain reconfigurable devices,” in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, dec. 2005, pp. 129 –136.