Memory Aware DVFS framework for SPARC based LEON3 Processor

By
Zohaib Najam
NUST201362497MCEME35513F

Supervisor
Dr. Umar Shahbaz Khan
Department of Mechatronics Engineering

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Mechatronics Engineering (MS-78)

In
College of Electrical and Mechanical Engineering,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(August 2016)

Memory Aware DVFS framework for SPARC based LEON3 Processor

By

Zohaib Najam

NUST201362497MCEME35513F

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Mechatronics Engineering (MS-78)

Thesis Supervisor

Dr. Umar Shahbaz Khan

Thesis Superviser's Signature: ________________

In

College of Electrical and Mechanical Engineering,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(August 2016)

Declaration

I hereby declare and certify that this research work titled "**Memory Aware DVFS Framework for SPARC based LEON3 Processor**" is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST CEME or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST CEME or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Zohaib Najam

NUST201362497MCEME35513F

Signature: ______________

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the University.

Author Name: **Zohaib Najam**

NUST201362497MCEME35513F

Signature: ____________________

Superviser Signature: ____________________

Copyright Statement

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST CEME or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST CEME or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Zohaib Najam**

NUST201362497MCEME35513F

Signature: ________________

Acknowledgment

I am thankful to my beloved parents who raised me and supported me throughout my life in good and bad times. I would also like to express special thanks to my supervisor Dr. Umar Shahbaz Khan for his sincere supervision throughout my thesis and also for Advanced Embedded System course which he has taught me.
I can safely say that I have in depth understanding of this subject. I would also like to thank Dr. Muhammad Yasir Qadri, Dr. Khurram Kamal and Dr. Mohsin Islam Tiwana to assess and evaluate my Thesis Dissertation, and express my special thanks to Dr. Mubashir Saleem for his guidance as PG coordinator.
I am also thankful to Mr. Zahid and Ayaz for his support and cooperation. Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Abstract

This research has proposed hardware modification in softcore LEON3 processor based on SPARC V8 Architecture. LEON3 is an open source softcore processor described in VHDL hardware description language, the softcore nature of LEON3 allows high level of reconfiguration and customization of hardware design so the major contribution of this dissertation is to modify the hardware design to include support for dynamic voltage frequency scaling and a technique to change the system operating frequency at run time during the execution of standard benchmark applications. In addition, this research has also proposed a technique to acquire performance statistics of the processor such as cache hit/miss ratio and number of cycles consumed by the application which in turn can be exploited to reconfigure the operating frequency to improve performance of the system. The proposed techniques have been tested on a Xilinx prototyping board XUPV5 and performance statistics have been validated by comparative analysis with LEON3 simulator TSIM.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

The research work in this dissertation has been presented in three parts. First part includes support for performance monitoring unit in LEON architecture to report dynamic events such as cache hit/miss ratio, number of executed instructions and number of cycles consumed by a particular application. A performance monitoring unit is very important to debug complex hardware design but open source GPL distribution of LEON3 does not support this feature. Second part includes support for dynamic voltage frequency scaling to dynamically reconfigure processor's operating frequency which will have certain implications on performance and power consumption. The last part consist of integration of first and second part i.e to reconfigure operating frequency driven and controlled by real-time performance statistics.

1.1 Problem Statement and Contribution

In battery operated electronic devices such as smart phones, tablets and laptops better battery time and fast processing has been the need of consumers and hence becomes a challenging milestone for developers and design engineers. Dynamic voltage frequency scaling has been considered as one of the most efficient technique to reduce power consumption specifically for battery operated devices. Over the years many sophisticated DVFS algorithms and heuristics have been introduced to address power issues, however most of the DVFS algorithms have been developed using direct relation of frequency and power consumption as discussed in equation 2.2.

$$P_{static} = I_{leakage} \times V_{DD}. \tag{1.1}$$

$$P_{dynamic} = C_L \times V_{DD}^2 \times f_p. \tag{1.2}$$

Recent research also suggests that reducing the CPU frequency may not always reduce the power consumption due to processes involved in memory accesses. The contribution of this dissertation is to introduce memory aware DVFS algorithm for LEON architecture, this algorithm has been implemented on android operating system but since LEON architecture does not include support for dynamic voltage frequency scaling and performance monitoring unit, the proposed framework was not implemented prior to this dissertation. Moreover including support for PMU and DVFS are the other contributions of this dissertation.

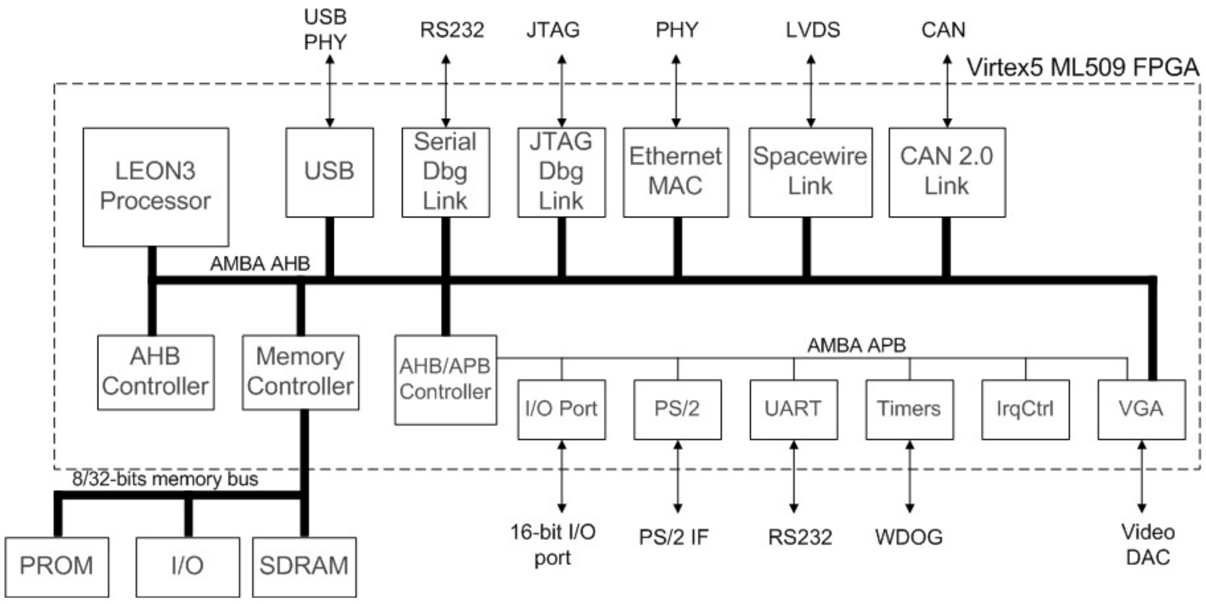


Figure 1.1: Architectural Diagram of LEON3 showing AMBA bus connections

1.2 LEON3 Architecture

LEON3 is a 32-bit soft multicore processor based on SPARC V8 architecture written in synthesizable VHDL language[1]. The complete LEON3 system is highly configurable via VHDL generics defined in the configuration file or xconfig GUI tool made available by its developers. The source code is available under various distrbutions from Gaisler Research. LEON architecture has been designed for embedded and spacecraft applications which requires stringent and precise real-time computing. Most of the source code is available under the GNU General Public License (GPL) and has various template designs to implement on field-programmable gate array (FPGA) prototype boards. The source code under GPL distribution does not include clock gating, asymmetric multiprocessing and floating point unit which have been made available under commercial license. The LEON3 system has been developed around advanced microcontroller bus architecture (AMBA). The AMBA bus consist of two buses, the advanced peripheral bus (APB) and the advanced high-performance bus (AHB). The system uses the AHB bus to connect LEON3 processor to memory controller, debug support unit and other devices such as Ethernet, serial and JTAG debug links. By default, minimal LEON3 system requires processor to act as master on the bus, memory controller and APB bridge acts as slave and uses the bus to communicate with the processor. The memory controller provides access to three types of memories: SRAM, PROM and SDRAM and allows FPGA to communicate with on-board memory standards. The APB bridge is connected to both buses, acts as master on AHB while act as slave on the APB bus. The bus architecture of LEON3 has made each IP core to be mapped at a particular address space, this is useful to access on-chip registers of each attached IP core from application program, such as UART scalar register, multiprocessor status register and GPIO data register. New modules can be easily added to the system using AMBA AHB/APB buses and it can be verified using GRMON. A block diagram of LEON3 architecture is shown in Fig. 1.1.

1.3 Space Applications

The major applications of the proposed enhancement in LEON3 include hard real time systems such as space applications. Researchers have also developed various space applications based on LEON3 processors. Performance estimation for spacecraft applications was suggested by [2] using LEON3 FT processor with the computation of only significant events related to space applications i.e. cycles per instruction, time per instruction, time per cycle and instructions per task. Space related applications are also tested with on board FPGA based LEON FT processor platform by [3] for H2Sat mission and evaluate the reliability and effectiveness of a reconfigurable processor i.e. LEON3 over ordinary satellite command system.
Performance comparison between LEON based processors and other processors for various space craft application and mission is provided in [4]. Frequency based performance evaluation was done by [5] for space applications with LEON based platform.

1.4 Summary

In this chapter LEON3 state of the art soft core processor based on SPARC V-8 architecture has been briefly discussed, as research work under this dissertation has been carried out on this architecture. In addition hardware modification of LEON3 and contribution of this research has also been summarized in this chapter, the rest of the chapters are summarized as follows.
Chapter 2 talks about the literature review to discuss related work, research originality and novelty of this dissertation. In Chapter 3 we will discuss the modified and Enhanced LEON3 architecture in terms of dynamic voltage frequency scaling (DVFS) and real time performance statistics. Chapter 4 will include the implementation results, validation of those results and overheads induced due to work contribution in this dissertation. The last chapter is conclusion and it highlights research novelty and its potential in the field of Embedded Systems and Computer Architecture, it also covers how this research can be extended in future if appropriate measures are taken.

Chapter 2

Literature Review

The research work of this dissertation has three parts. PMU part has been implemented by various researchers in internationally published articles, [6] has introduced a real-time performance monitoring unit for LEON3 and made comparative analysis with LEON3 emulator TSIM but this work was limited to single core LEON3 processor. Real-time hardware performance statistics are considered efficient metrics to reconfigure hardware design, memory aware DVFS framework is an instance of such reconfiguration. This algorithm has been implemented on android operating system and the results suggests improvement in performance and energy consumption than situation of simply selecting lowest supported frequency [7].

2.1 Background of DVFS

The increase in transistor density in hardware prototyping environments has enabled the implementation of complex systems on a single chip at the expense of high device cost and power consumption. Over the last decade the manufacturers of Field Programmable Gate Arrays (FPGA) such as Actel, Altera and Xilinx have improved their devices in terms of logic density, therefore the deployment of reconfigurable hardware in embedded systems to optimize both performance and power consumption has been one of the important area of research [8]. [9] highlight the challenges of reconfigurable processor architectures and need of a performance monitoring scheme to implement complex hardware designs on FPGA's. In [10] an efficient power saving technique is presented in which partial dynamic reconfiguration (PDR) has been deployed in FPGA to reconfigure part of the hardware design on the fly without interrupting rest of the system design. DVFS (Dynamic Voltage Frequency Scaling) is also one of the widely used power reduction techniques for general purpose processors and is generally implemented in the kernel to change the frequency and voltage of a microprocessor in real time driven by various supported algorithms [11]. The major contribution of this paper is to include support for DVFS in LEON3 architecture driven by general-purpose I/O port of LEON3. In [12] similar framework for 32-bit PowerPC system on chip has been proposed to address battery powered applications.
Over the years researchers have greatly explored DVFS for power saving and proposed efficient algorithms which have proven better results as well. [5] have analyzed the performance of LEON 3FT processor at different operating frequencies and the end result shows improved execution time in contrast to processors among LEON's predecessors. The efficiency of DVFS can be further enhanced by combining dynamic parallelism with DVFS [13]. DVFS is a technique which allows voltage and frequency of a processor to be

adjusted on the fly in order to manage CPU energy consumption and processing capacity. When high performance is the priority frequency can be increased to a higher level (Over clocking) similarly frequency can be decreased to a lower level (CPU throttling) in order to extend battery time and reduce power dissipation. However, recent research also suggests that reducing the CPU frequency does not necessarily reduce the power consumption [7]. This is mainly observed for memory bound applications where optimal power consumption occurs at a frequency other than the lowest one supported by the processor because such applications mainly rely on memory access and its latency which are independent of CPU frequency [7].

GPL (general public license) distribution of LEON3 does not support DVFS so the contribution of this paper involves architectural modification to include support for DVFS in LEON3. The work presented in this paper is based on integration of GPL distribution of LEON3 with Xilinx commercially available soft IP DCM_ADV. DCM_ADV is the Xilinx primitive commercially available soft IP that provides all the clock synthesis capabilities of the original DCM as well as access to the dynamic clock reconfiguration feature. The dynamic clock reconfigurable feature enables synthesis of a new frequency adjusted clock without reloading the bitstream into FPGA.

2.1.1 DVFS Algorithms

DVFS is a widely used technique to leverage the trade-off between power consumption and throughput and it has been implemented in various architectures[14]. Modern processors such as Intel XScale, AMD Athlon and Transmeta Crusoe are equipped with the DVFS feature[15]. DVFS has been considered as the most efficient way to reduce power consumption and dissipation as it reduces both static (leakage) and dynamic power [16].

$$P_{static} = I_{leakage} \times V_{DD}. \tag{2.1}$$

$$P_{dynamic} = C_L \times V_{DD}^2 \times f_p. \tag{2.2}$$

Over the years various algorithms and heuristics have been proposed in the implementation of DVFS. Artificial Intelligence (AI) driven algorithms have certainly shown its significance in high performance computations, probabilistic models and statistical reasoning[17]. In [15] Kong and Choi have proposed a DVFS technique for multicore sytems which considered energy-delay product (EDP) of running application to decide optimal settings of frequency and voltage of processor cores dynamically. Another approach estimates the subsequent workload and deadlines in advance and DVFS is performed by operating system (O.S) to reduce power consumption [18]. In another DVFS technique, the proposed algorithm require either application or compiler support in order to perform DVFS [19]. DVFS technique which has proven better results involves processor run-time statistics and online learning algorithm to determine voltage and frequency at a given time with a goal of minimizing energy consumption. This algorithm has proven better results in terms of power saving with a slight degradation in performance for both memory bound and CPU bound applications [19, 20]. In [16] temperature aware dynamic scaling framework is proposed, DVFS technique is based on chip temperature in order to reduce both static and dynamic power. In [21] dynamic run-time events such as memory access counts and cache hit/miss ratio obtained from performance monitoring unit (PMU) are used as heuristics to dynamically scale voltage and frequency, similar to this approach, Y.Liang proposed a novel memory aware DVFS algorithm based on correlation between critical speed and memory access rate of the benchmark applications [22]. This approach has also proven better results in terms of energy saving when compared to Linux built-in On-demand DVFS [22]. Some other DVFS algorithms consider cache miss and instruction per cycle rate (IPC) as a fine heuristics to drive voltage scaling[23, 24]. Related work suggests that DVFS algorithms requires real-time feedback of dynamic events but GPL distribution of LEON3 does not include real-time performance monitoring unit which is the contribution of [6].

2.2 Background of PMU

Most modern processors are equipped with PMS to aggregate fine-grained information regarding its hardware design. Requirement of performance statistics in reconfigurable computing has motivated many researchers to provide non-intrusive and real-time monitoring of performance statistics with minimal resource usage. Related work has been done for LEON3 platform by [25]. The authors provide architectural modification to include PMS under single, dual and quad-core LEON3 processors but their proposed scheme needs to include a specific software infrastructure to collect performance statistics that includes perf_event of Linux and require kernel mode for execution. Improving LEON3 with the aid of PMS for specific spacecraft applications and worst-case execution time (WCET) analysis is explained by [6], but this work was limited to a single core implementation only. [26] compute accurate cycles per instruction (CPI) components by proposing hardware performance counter architecture and tested using SPEC CPU 2000 benchmark suit. The authors compared their results against the one implemented in IBM POWER5.
Ideal architecture for profiling require three features as explained in [27] including concurrency, non-intrusiveness and flexibility. It requires concurrent monitoring of counters and in this paper we have presented concurrent monitoring of events in hardware while providing flexibility in software to sample events at run-time. Secondly it requires the modification to be less intrusive in terms of resource utilization such that there should be no need for extra dedicated bus or new instructions for the hardware. The proposed work has been implemented in hardware with minimal resources and less intrusive as explained in section 4. Thirdly, hardware must allow software to control and access collected data with flexibility and scalability, as in our work software can fully control the performance monitoring mechanism by start profiling, stop profiling, resetting and collecting counters.

2.2.1 Power estimation and PMU

In recent years productivity of hardware counters is extended as it is considered as a key metric for power estimation. Emulation of workload distribution using hardware performance counters is presented by [28]. Reconfigurable computing is generally deployed to optimize resource utilization per application. The reconfigurable feature can only be exploited if software/hardware developer can monitor hardware performance metrics at run time. [9] discuss the challenges of reconfigurable processors and effectiveness of performance counters. [29] has exploited the correlation between power consumption and performance of processor and complete estimation of power consumption using performance counters without any dedicated power sensor hardware.
Common application focused by researchers using PMS is for workload optimization.

Performance-energy trade-off and linear correlation between performance and energy can be considered perfect case study for this enhanced architecture. Dynamic power estimation using HPC support is exploited by [30] via related hardware counters. Only four events i.e. [x,y,z,a] were used by [31] to estimate real time performance and thread scheduling.

2.2.2 Predictability and Energy optimization using Cache line locking

Recent studies show that cache line locking can be useful for improvement of predictability and performance-power ratio in real time embedded systems. [32] provides cache locking scheme for the improvement of predictability caused by increasing performance to power ratio for specific percentage of cache size locked. Other applications of PMS includes run time tuning and adjustment of cache configurations in order to save energy. Cache can be configured for a more predictable behavior and for worst case execution time (WCET) analysis using line locking. Line locking can be done both statically or dynamically. As the cache requirements vary for various applications, [33] proposed a self tuning cache to save energy for memory subsystems per application based on run time acquisition of performance counters. [34] address the problem of unpredictability provided by cache behavior using dynamic cache line locking. Authors provide useful implementation of line locking based on cache performance counters to evaluate the the bottlenecks of unpredictable cache behavior and worst case performance estimate for safe computation in hard real time systems.
[35] also worked on unpredictability management using line locking and explored its implications to avoid size related jitters and WCET analysis. However the applications of PMS are not only limited to power estimation and predictability analysis but can also be used in other techniques such as hardware support for bug detection in real time etc. [36].

Chapter 3

Design and Methodology

The design and methodology of this dissertation has three phases as discussed in the introduction section.

3.1 DVFS related modifications in hardware design of LEON3

LEON3/GRLIB includes multiple source distributions to target developers with different milestones. GPL is the GRLIB general public license (open source) distribution. This open source distribution has a clock generation unit that uses DCM (IP Core provided by Xilinx) to generate a fixed scale down processor clock typically 60 MHz. However, this DCM does not support dynamically reconfigurable frequency adjusted clock feature.
The system on chip (SoC) has been extended to include frequency reconfigurable feature using DCM_ADV (IP Core provided by Xilinx) which includes full access to all the features of original DCM as well as support for dynamic reconfiguration circuit to dynamically synthesize a new frequency adjusted clock. This functionality has been implemented by dynamically changing the multiply and divide attributes of DCM (CLKFX_MUL and CLKFX_DIV) using dynamic reconfigurable ports (DI and DADDR) introduced in DCM_ADV IP Core (equation 3.1).

$$Processor_{clk} = BoardFrequency \times \frac{CLKFX\_MUL}{CLKFX\_DIV}. \qquad (3.1)$$

The DCM induction in clock generation unit is intact and DCM_ADV primitive has been introduced, the system clock is switching between the clock generated by DCM and DCM_ADV. The clock generated by DCM is static whereas the clock generation using DCM_ADV is dynamic, moreover when generating a new frequency adjusted clock, LEON3 processor clock switches to static clock coming from DCM and when the new frequency adjusted clock is locked, processor clock switches back to clock coming from DCM_ADV. This is useful to ensure that processor clock never stops as stopping the processor clock may halt the pipeline until it awakes back (Fig. 3.1).

In order to run LEON3 processor on Virtex-5 board, maximum supported processor frequency is up to 80-90 MHz, DFS feature of DCM_ADV primitive has two modes of operation; low and high. , both modes have different valid output range of frequencies which is shown in Table 3.1. The modes can be dynamically changed in order to satisfy the needs of design engineers but in our case as LEON3 cannot run above 100 MHz so mode selection in our experimentation is low.

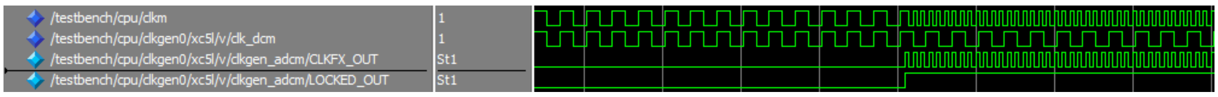

/testbench/cpu/clkm
/testbench/cpu/clkgen0/xc5l/v/clk_dcm
/testbench/cpu/clkgen0/xc5l/v/clkgen_adcm/CLKFX_OUT
/testbench/cpu/clkgen0/xc5l/v/clkgen_adcm/LOCKED_OUT
1
1
St1
St1

Figure 3.1: An illustration of Dynamic Frequency Scaling; System clock (clkm) never stops and switches between clocks generated by DCM (clk_dcm) and DCM_ADV (CLKFX_OUT), LOCKED_OUT shows the status of newly synthesized frequency

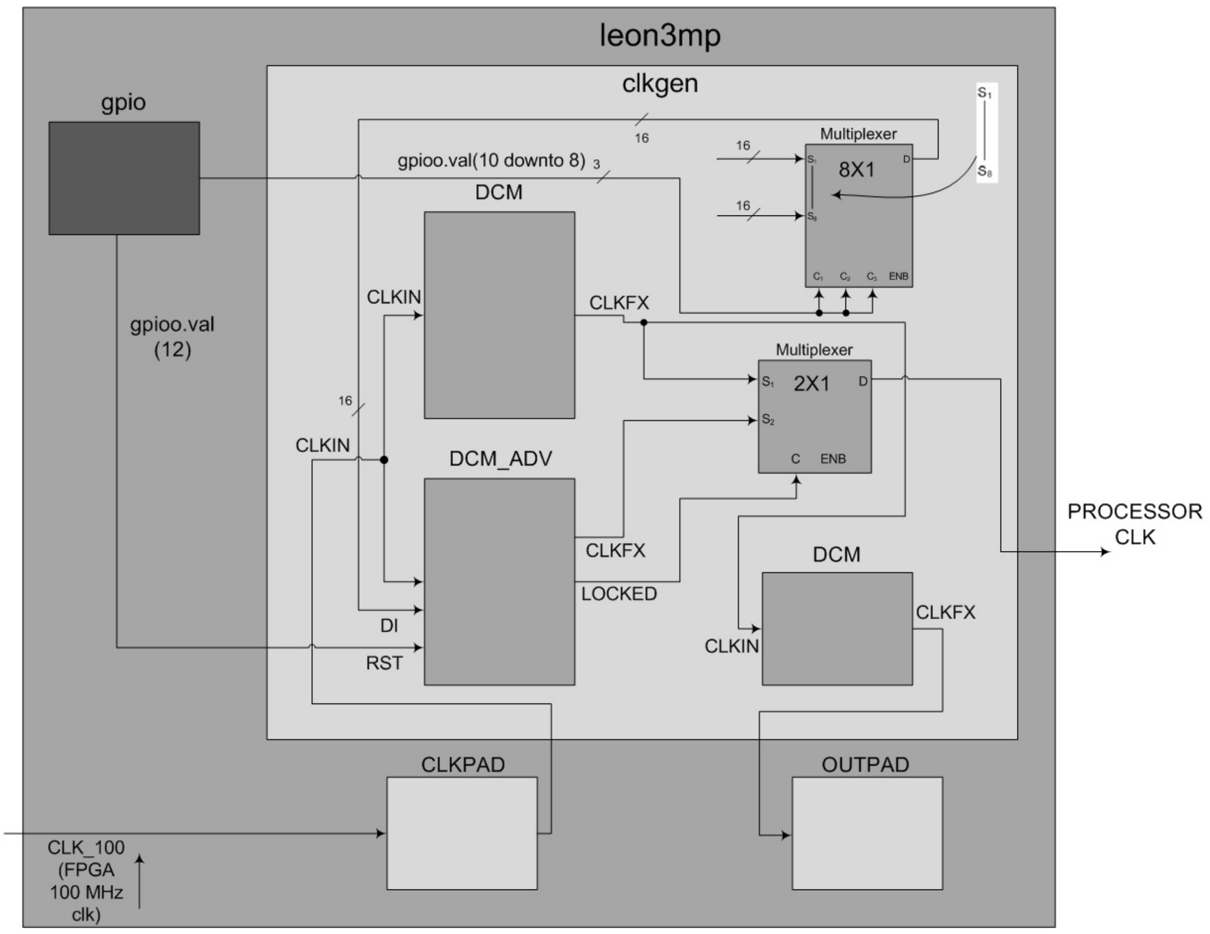


Figure 3.2: Architectural Modification of LEON3 which includes support for reconfigurable system clock frequency

Table 3.1: DFS modes of operation

DFS mode	Fin (Mhz)	Fout (Mhz)
Low	1.0 - 140	32.0 - 140
high	25 - 350	140 - 350

Table 3.2: Look-up table for system frequency

gpioo.val[10:8]	DI[15:0]	Frequency
000	0000011000001001	70MHz
001	0000011100001001	80MHz
010	0000011000010011	35MHz
011	0000100000010011	45MHz
100	0000101000010011	55MHz
101	0000110000010011	65MHz
110	0000111000010011	75MHz
111	0001000000010011	85MHz

3.1.1 Voltage/Frequency Control Unit

The reconfigurable ports which allow dynamic frequency scaling are programmed using GPIO peripheral attached to advanced peripheral bus (APB) because GPIO can be accessed (read/write) from application program using its direction and data register mapped at a particular address space in the architecture, so this modified architecture would allow the developers and design engineers to change the processor frequency on the fly during the execution of application programs and benchmarks. The modified architecture of LEON3 including reconfigurable feature is presented in the Fig. 3.2.

3.1.2 Implications of DVFS on UART

UART is one of the peripherals attached to the APB bus and is mapped at 0x80000100 in the address space. The interface is provided for serial communications and this UART has a programmable 12-bit scalar to generate the desired baud-rate, the number of scalar bits can be increased with VHDL generic s-bits. One appropriate formula to calculate the scalar value for a desired baud rate is as follows.

$$scaler\,value = \frac{system\_clock\_frequency}{(baud\_rate * 8) - 1}. \tag{3.2}$$

The above equation shows that system clock has a direct effect on scalar value so dynamically changing the system clock frequency without adjusting the scalar value accordingly can result in UART malfunctioning. In order to avoid this situation we have reconfigured the UART scalar register every time a new frequency adjusted clock is locked. The UART scalar register is mapped at 0x8000010C so it can be reconfigured from the application program.

Algorithm 1 Software procedural steps for DVFS

```
1: procedure DYNAMIC VOLTAGE FREQUENCY SCALING
2: Reset DCM_ADV by setting the 12th bit of GPIO data register
3: Re-program data port of DCM_ADV (DI) by writing a value from Table II to GPIO port 10 down-to 8
4: Release the reset of DCM_ADV by writing a 0 to corresponding GPIO port (bit 12)
5: Finally reconfigure the UART scalar register with a value calculated using equation 4
6: end procedure
```

3.2 PMU related modifications in hardware design of LEON3

The integer unit of LEON3 is composed of pipeline stages of instructions with various signals providing status of instruction. Cache subsystem provide various signals about cache behavior for analysis. These signals are used to collect hardware events. Two 32-bit registers were added for both Instruction and Data cache controller to poll cache stats and one 32-bit register was added in integer unit to poll cycles count. A simple HPC controller was added to the integer unit for the collection of counters. Tracing data access, timing and execution information of application automatically as a feedback from hardware is important for performance analysis and debugging. This information can be useful to decide optimal configuration of reconfigurable parameters. Architectural modification includes three sub systems: *event detection*, *event polling* and *collecting counters* as explained below.

3.2.1 Event Detection

Event detection is based on cache hit signals and some other events that are indicator of cache access. Instruction cache is accessed for every clock cycle except when pipeline stall signal is set or when pipeline nullifies the cache access. Once instruction cache access is discerned, hit signal in cache controller indicates instruction cache hit. Similarly in data cache controller module of the processor, hit will occur when cache access is registered followed by the indication that instruction is executing in memory stage with a hit signal. The flowchart of event detection phenomenon for I-Cache is explained in Fig. 3.3, while the flow chart for detection of HPC's of D-Cache is illustrated in Fig. 3.4.

3.2.2 Event Polling

Event polling includes addition of registers to count the number of occurrences of specific events. These counters include instruction cache access counter (IAC), instruction cache hit counter (IHC), data cache access counter (DAC), data cache hit counter (DHC) and cycles counter (CC). IAC, IHC and DAC, DHC all reside in the instruction cache controller and data cache controller respectively while CC resides in the integer unit.

3.2.3 Counter collection

Data from counters can be collected using some SPARC instructions or logic analyzer at real time. This phenomena can be controlled via HPC controller that resides in integer

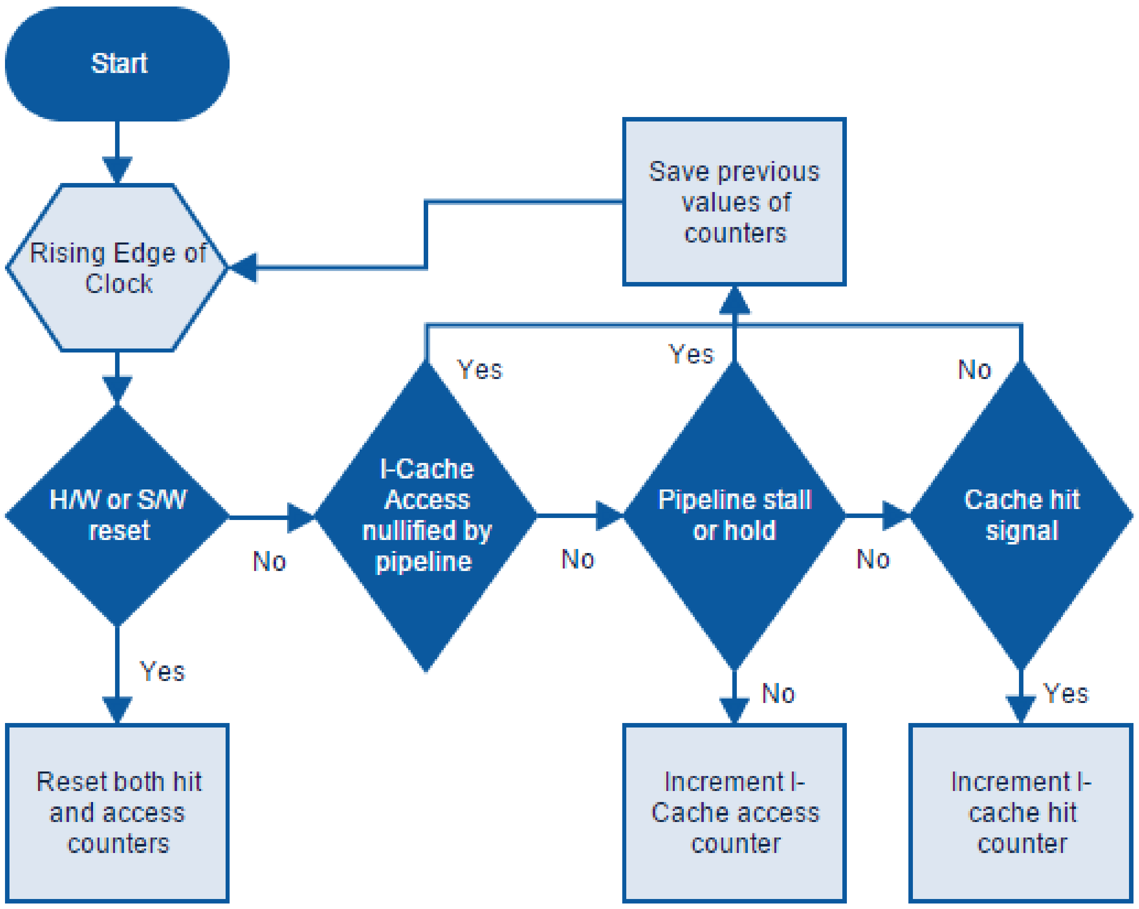

Start
Rising Edge of Clock
Save previous values of counters
Yes
Yes
No
H/W or S/W reset
No
I-Cache Access nullified by pipeline
No
Pipeline stall or hold
No
Cache hit signal
Yes
No
Yes
Reset both hit and access counters
Increment I-Cache access counter
Increment I-cache hit counter

Figure 3.3: Flow chart of I-Cache HPCs

unit. Application specific registers (ASRs) that are part of integer unit used for the collection of counters via application program as ASR's can be read/write using simple rdasr and wrasr instructions in C program. Writing to ASR25 decides which event to capture on ASR24 so application program is allowed to write on ASR25 with a value corresponding to a specific event such as *start_profiling()* to boot counter values, *stop_profiling()* to stop counter increments, *hpc_rst()* to reset all hardware performance counters and *hpc()* for collection of counters sequentially. Similarly *I_cache_disable()* can be used to disable the complete instruction cache, *D_cache_disable()* function can be used to disable data cache completely, *CL(size)* function for line locking of given cache size (separately for instruction and data cache) and *energy()* function for collection of energy consumption stats. These function were written in C assembler and decoded by HPC controller in order to control the complete working of performance counters.

HPCs are calculated under the system of event detection and event polling. To access these counters without any specified infrastructure, SPARC instructions can be used to fully control the functionality by writing only four bits of an ASR. User can access any specific counter by writing to ASR at run time. Accessing and resetting of these counters can be done using simple user-defined functions based on SPARC assembly instructions. Performance counter reset bit (PR) is used to reset all the counters dictated by software application. HPC controller is the brain of PMS, it allow applications to access specific counter and to reset them with minimal complexity and overhead on the architecture.

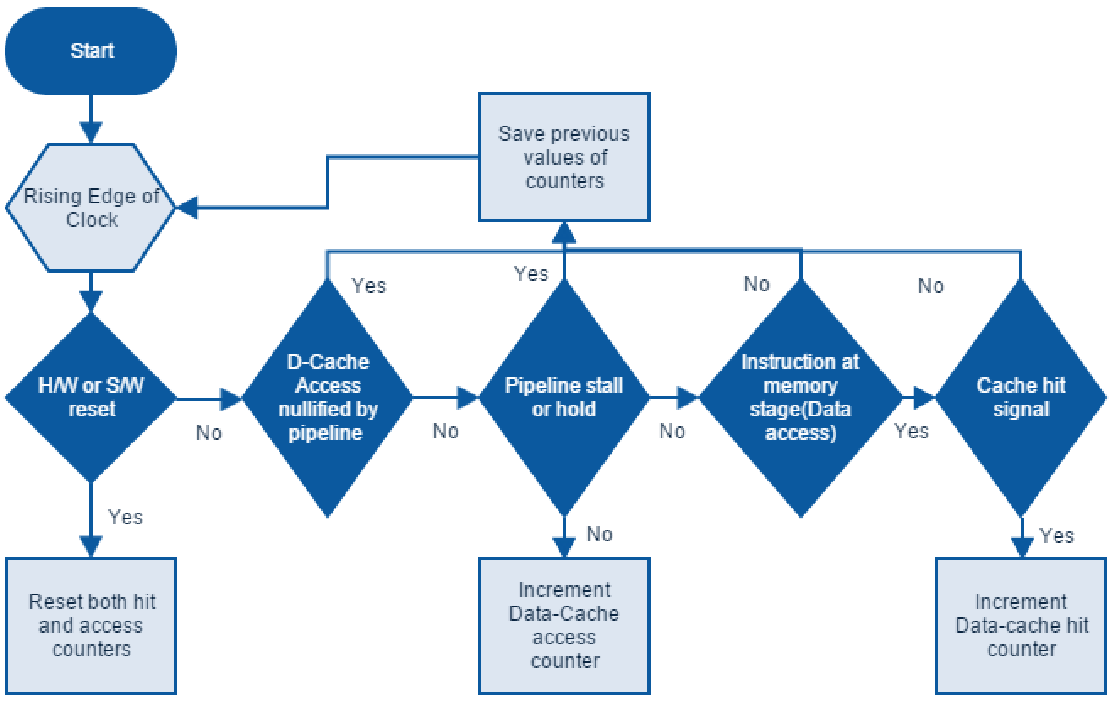

Start
Rising Edge of Clock
Save previous values of counters
H/W or S/W reset
D-Cache Access nullified by pipeline
Pipeline stall or hold
Instruction at memory stage(Data access)
Cache hit signal
Yes
No
Reset both hit and access counters
Increment Data-Cache access counter
Increment Data-cache hit counter

Figure 3.4: Flow chart of D-Cache HPCs

Five 32 bit registers were added to count the events concurrently, while all these counters are multiplexed by a simple 16x1 multiplexer. Selection of some specific counters or some specific actions on counter values can be performed by HPC controller. Hardware reset in addition to software reset is also included which can be controlled using FPGA switch.

Hardware counters can easily be implemented using some hardware signals reside in integer unit and cache controller module of LEON3 architecture. Flowchart to access hardware cache statistics are shown in Fig. 3.3 and Fig. 3.4. Flow chart illustrates that for every rising edge of clock, system determines whether there is H/W reset from board or a S/W reset from application program to reset counter values to zero. For instruction cache the access counter increments for every cycle except when there is a pipeline stall or when instruction cache access is nullified because of some exception, fault or pipeline bubble. Once this is discerned the cache hit signal will indicate the instruction cache hit and hit counter will be incremented.

Similarly data cache access counter will increment every time when there is no H/W or S/W reset, no pipeline stall, no nullified cycle and also the executing instruction must resides in the memory stage of pipeline. Now setting of cache hit signal will indicate that data cache was hit and data cache hit counter will be incremented.

3.2.4 Performance monitoring scheme and HPC controller

There is no need for any specific software infrastructure to collect these counters, integration of customized assembly instructions with benchmark's source code are enough to get the counter values or to reset them. Integer unit module in hardware design of LEON3 allows processor to communicate with the external world using watchpoint registers *(ASR24 - ASR31)*. The work presented in this paper make use of two ASR's to implement the desired functionality. ASR24 is used to read the counter values of specific

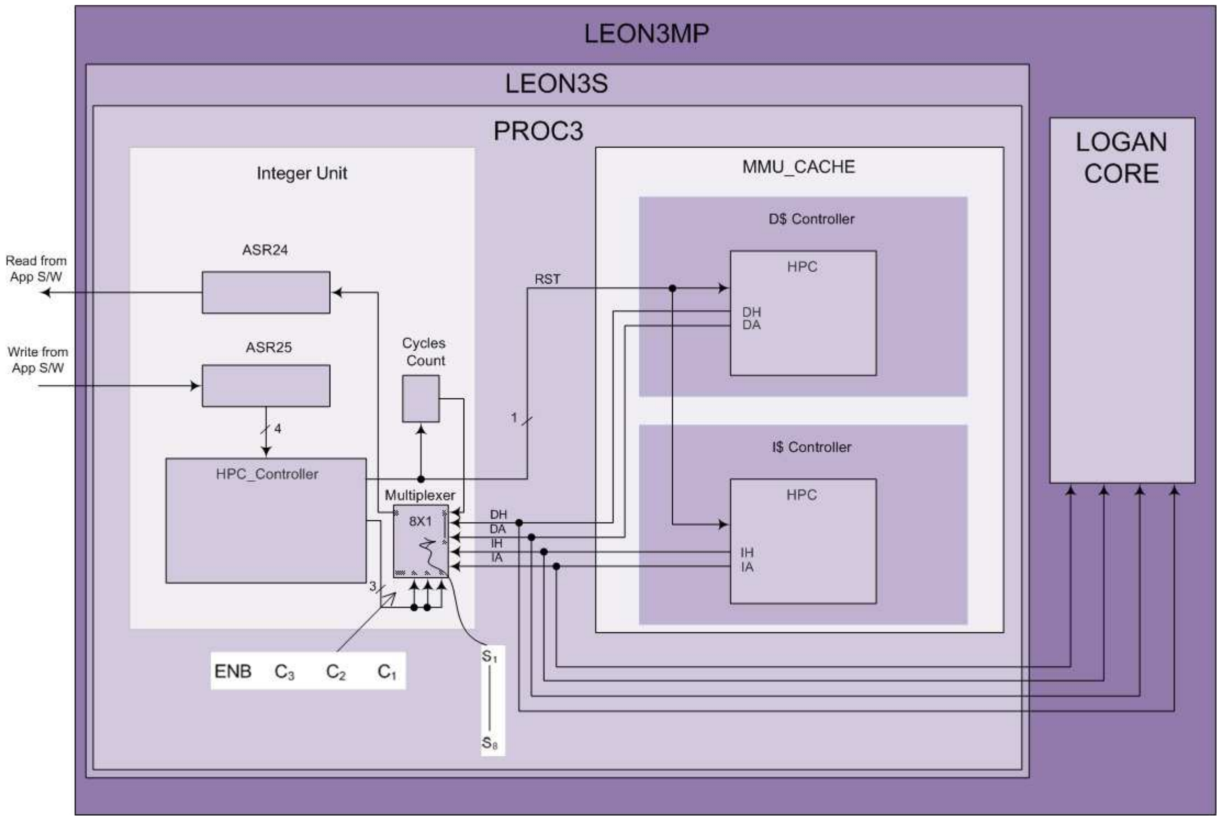


Figure 3.5: Modified architecture of LEON3 with HPC controller

event from application software driven by user specified logic as user can select these counters concurrently by communicating with HPC controller through only four bits of ASR25.

A real time environment running standard benchmark applications need to collect hardware performance statistics for that application so there must be an interface between application and hardware to read these counters when desired. Ability to reset these counters from application program makes it possible to extract performance information before switching to any other application, this in turn allows continuous reconfiguration of parameters at run-time. The complete modification and enhancement of LEON3 architecture with controller for HPCs is illustrated in Fig. 3.5. Performance counters which resides in cache controller modules are multiplexed with each other and controlled by ASR25 so that counter value of any monitored event is readily available at application level. HPC controller resides in integer unit and is provided with reset (RST) signal to reset the counter values in hardware, hence, profiling of events can be stopped by setting the reset signal for number of clock cycles and profiling can be started again by lowering the reset signal at application level using ASR's. GRMON debugger can be used for analysis of performance results as these counter values were also mapped to Logic Analyzer (LOGAN) core of LEON3 available in GPL distribution for run-time evaluation.

Chapter 4

Implementation and Results

4.1 Hardware Test Environment

Before the discussion of tests and their obtained results, a brief introduction of the test environment is presented. Fig. 4.1 shows the prototyping board used in our experimentation to verify the proposed functionality. It is the target device for implementation of LEON3 core. This technology supports dynamic reconfigurable clock synthesis feature as provided by DCM_ADV Xilinx primitive. The board has 100 MHz single ended crystal oscillator and is powered by 3.3V supply, it provides the source clock to dynamic reconfigurable circuit of DCM.

A part of test environment involves use of GRMON debug monitor, a debugging software tool intended to debug and examine the hardware configuration of LEON3 system at run-time. GRMON can be started from command prompt in Windows platform and support multiple interfaces such as JTAG, UART and Ethernet to connect to the target device. GRMON also allows downloading and execution of cross compiled application programs using simple load and run commands once attached to the target prototyping board running LEON3 processor. When GRMON connects to the target system it scans the system configuration which includes list of attached IP cores, system operating frequency and GRLIB build version. This can be done by reading plug and play information located at 0xfffff000 address on AHB bus, as this information also includes processor operating frequency so once processor operating frequency is dynamically changed during the execution of programs, exiting and then reconnecting GRMON can be used to validate that a new frequency has indeed been locked.

4.2 System Testing

In this section we first describe the complete development flow of this study followed by system configuration and finally the tests performed with their results.

4.2.1 Development flow

The development flow of our experimentation involves integration of embedded software applications and embedded hardware platform. Xilinx ISE design suite is used to generate hardware bitstream of LEON3 processor that can be downloaded on FPGA device and the execution of software application programs on target prototyping board is controlled

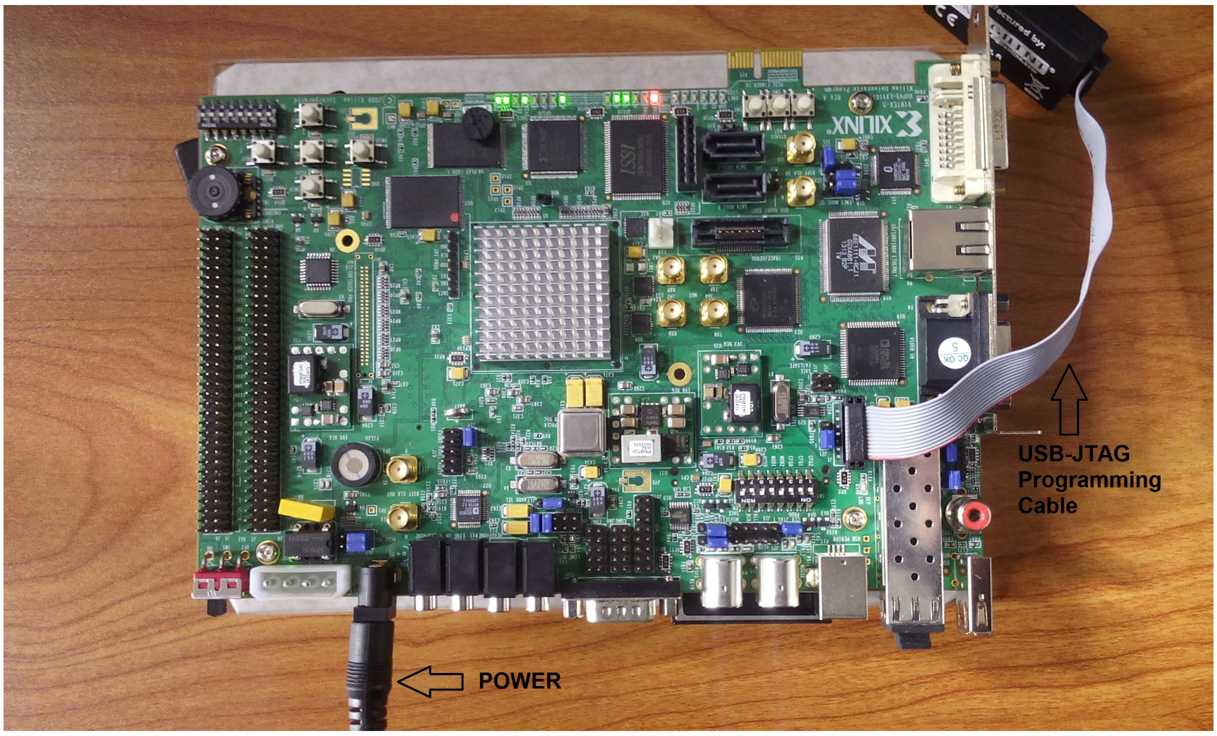
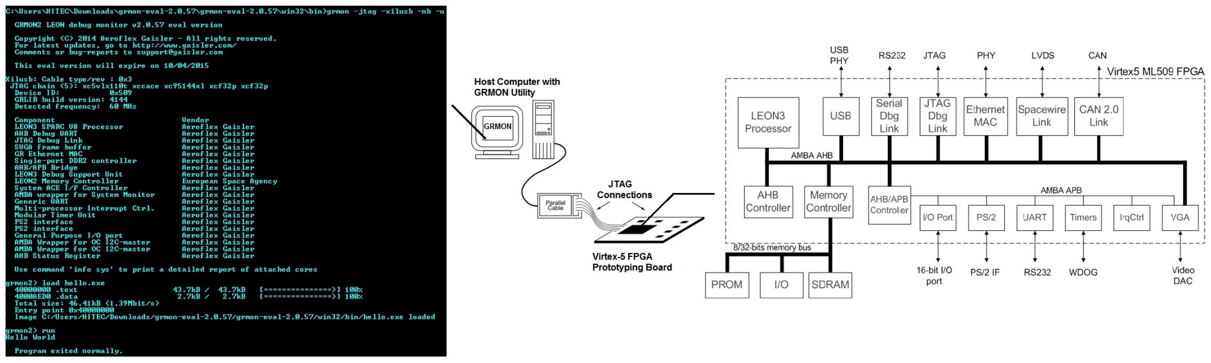


Figure 4.1: To verify implementation of DVFS and PMU in LEON3 architecture, Xilinx FPGA Virtex-5 ML509 prototyping board has been deployed in our experimentation

via GRMON debug monitor for LEON3 processor. The whole process is shown in Fig.4.2

4.2.2 System Configuration

LEON3 processor is initially configured via GUI made available by its developers to allow users an offline initial configuration of a complete LEON3 system which includes number of cores, operating frequency and cache size etc. Table 4.1 shows LEON3 platform and system configuration.



```
C:\Users\HITEC\Downloads\grmon-eval-2.0.57\grmon-eval-2.0.57\win32\bin>grmon -jtag -xilusb -nb -u

  GRMON2 LEON debug monitor v2.0.57 eval version

  Copyright (C) 2014 Aeroflex Gaisler - All rights reserved.
  For latest updates, go to http://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

  This eval version will expire on 10/04/2015

Xilusb: Cable type/rev : 0x3
 JTAG chain (5): xc5vlx110t xccace xc95144xl xcf32p xcf32p
  Device ID:           0x509
  GRLIB build version: 4144
  Detected frequency:  60 MHz

  Component                            Vendor
  LEON3 SPARC V8 Processor             Aeroflex Gaisler
  AHB Debug UART                       Aeroflex Gaisler
  JTAG Debug Link                      Aeroflex Gaisler
  SVGA frame buffer                    Aeroflex Gaisler
  GR Ethernet MAC                      Aeroflex Gaisler
  Single-port DDR2 controller          Aeroflex Gaisler
  AHB/APB Bridge                       Aeroflex Gaisler
  LEON3 Debug Support Unit             Aeroflex Gaisler
  LEON2 Memory Controller              European Space Agency
  System ACE I/F Controller            Aeroflex Gaisler
  AMBA wrapper for System Monitor      Aeroflex Gaisler
  Generic UART                         Aeroflex Gaisler
  Multi-processor Interrupt Ctrl.      Aeroflex Gaisler
  Modular Timer Unit                   Aeroflex Gaisler
  PS2 interface                        Aeroflex Gaisler
  PS2 interface                        Aeroflex Gaisler
  General Purpose I/O port             Aeroflex Gaisler
  AMBA Wrapper for OC I2C-master       Aeroflex Gaisler
  AMBA Wrapper for OC I2C-master       Aeroflex Gaisler
  AHB Status Register                  Aeroflex Gaisler

  Use command 'info sys' to print a detailed report of attached cores

grmon2> load hello.exe
  40000000 .text                      43.7kB /  43.7kB   [===============>] 100%
  4000AED0 .data                       2.7kB /   2.7kB   [===============>] 100%
  Total size: 46.41kB (1.39Mbit/s)
  Entry point 0x40000000
  Image C:/Users/HITEC/Downloads/grmon-eval-2.0.57/grmon-eval-2.0.57/win32/bin/hello.exe loaded

grmon2> run
Hello World

  Program exited normally.
```

Figure 4.2: Complete development flow of this work which includes implementation of enhanced LEON3 on target FPGA and debugging using GRMON

Table 4.1: LEON3 platform and system configuration

System Frequency	Reconfigurable (New feature)
PMU	Yes (New feature)
Processor Cores	4 (SMP)
I-Cache	8KB, 32bytes/line
D-Cache	4KB, 32bytes/line
Integer Unit	Yes
Target Technology	Virtex-5 FPGA
Simulation Environment	ModelSim
GRLIB Release	GPL 1.2.2-b4123

Table 4.2: Overhead Analysis and FPGA Resource Utilization

Type of Resource	LEON3	LEON3 + DVFS	Resource Increment
Slices	7251 out of 17280	7251 out of 17280	No Increment
LUTs	15,771 out of 69,120	15,792 out of 69,120	21 out of 69,120 (0.03%)
Block RAM	27 out of 148	27 out of 148	No Increment
Fan-out	3361	3373	12
DCM_ADVs	6 out of 12	7 out of 12	1 out of 12 (8.33%)
BUFGs	15 out of 32	18 out of 32	3 out of 32 (9.40%)
Power (W)	4.709	4.757	0.048

4.2.3 Experimental Results of DVFS

In this study standard representative benchmarks such as Dhrystone, Coremark and Stanford have been considered to analyze the implications of DVFS on execution time and power consumption of corresponding programs, these benchmark applications are tested at nine different frequencies supported by enhanced LEON3 architecture and the results are shown in Fig. 4.3, 4.4, 4.5. In order to compute power consumption of these programs product of number of cycles and power per cycle was measured. XPower Analyzer tool has been used to evaluate power per cycle at each supported frequency level. A brief comparison of FPGA resource utilization is also presented in Table 4.2 to analyze the overheads introduced by the proposed system due to architectural modifications.

Results suggests that the benchmarks included in this study are CPU bounded programs, for memory bounded programs minimum power consumption occurs at a frequency other than the lowest one supported by the architecture [7].

4.2.4 Experimental Results of PMU

As explained earlier, the results have been evaluated and compared using three different approaches for various benchmarks. Instruction cache hit rate as a performance stastistic is evaluated for stanford and coremark bench suits with Modelsim, TSIM and hardware as illustrated in Fig. 4.6. Similarly I cache results are explained in Fig. 4.7 for those benchmarks.

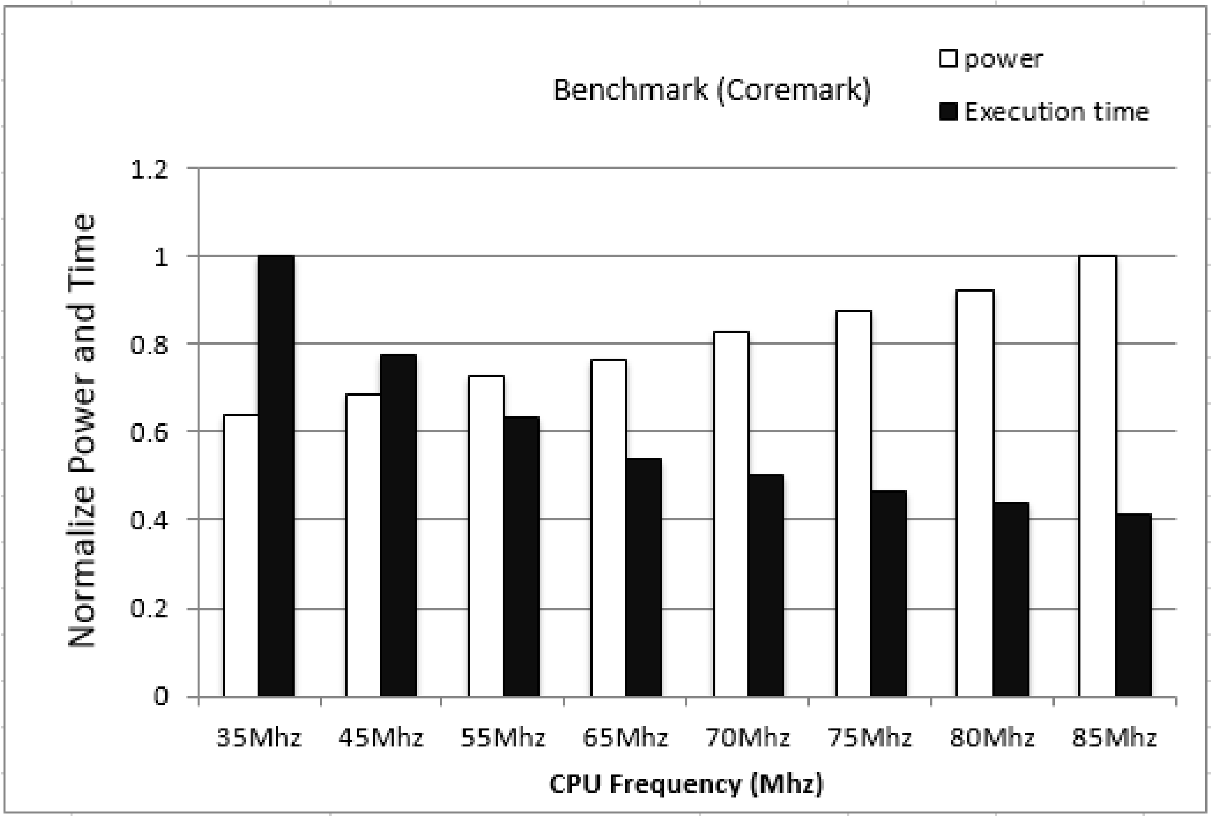


Figure 4.3: Normalized execution time and power consumption for Coremark benchmark application

Comparison with TSIM

Results have been validated and compared with TSIM. TSIM is an instruction level, cycle accurate, configurable LEON emulator provided by Aeroflex Gaisler. It can emulate a single LEON processor non-intrusively. System configurations deployed in our test environment are same as default TSIM configurations. The perf command of TSIM provide number of cycles, instructions executed, CPI and cache miss/hit ratio for both I-cache and D-cache. The effectiveness of TSIM is limited as it cannot be used for real time analysis. Simulation results of PMS are obtained using Modelsim and for real time hardware results GRMON application software is used to load and execute programs with customized assembly instructions in order to get the counter values of dynamic hardware events

Performance statistics of various standard benchmarks have been observed over TSIM for comparative analysis with the results obtained in this study. Results section suggest the precision achieved in this study as the error percentage lies within the tolerance lelvel of TSIM as established in figure 4.8 and 4.9. Application programs considered in this paper are small programs such as tree, bubble, perm etc extracted from Stanford benchmark so that performance statistics can be obtained in simulation using Modelsim.

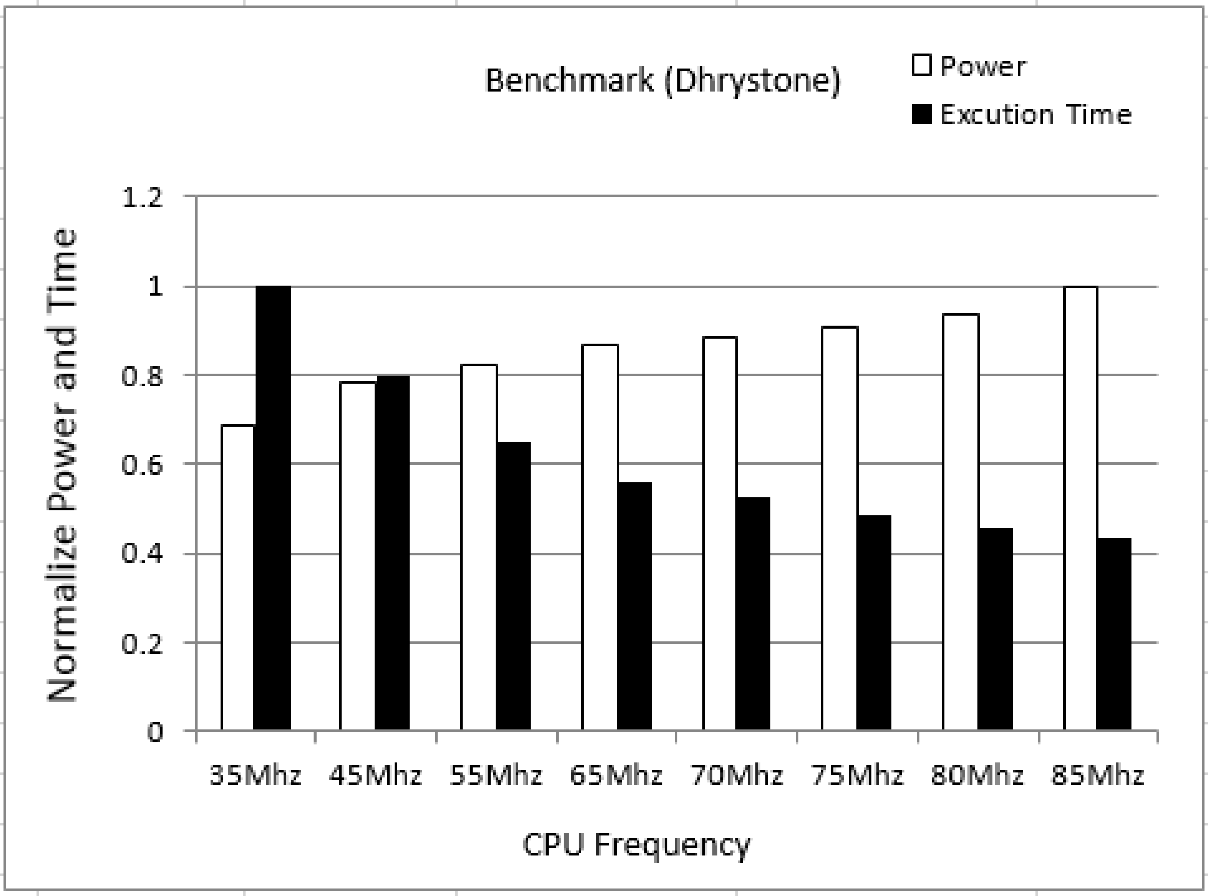


Figure 4.4: Normalized execution time and power consumption for Dhrystone benchmark application

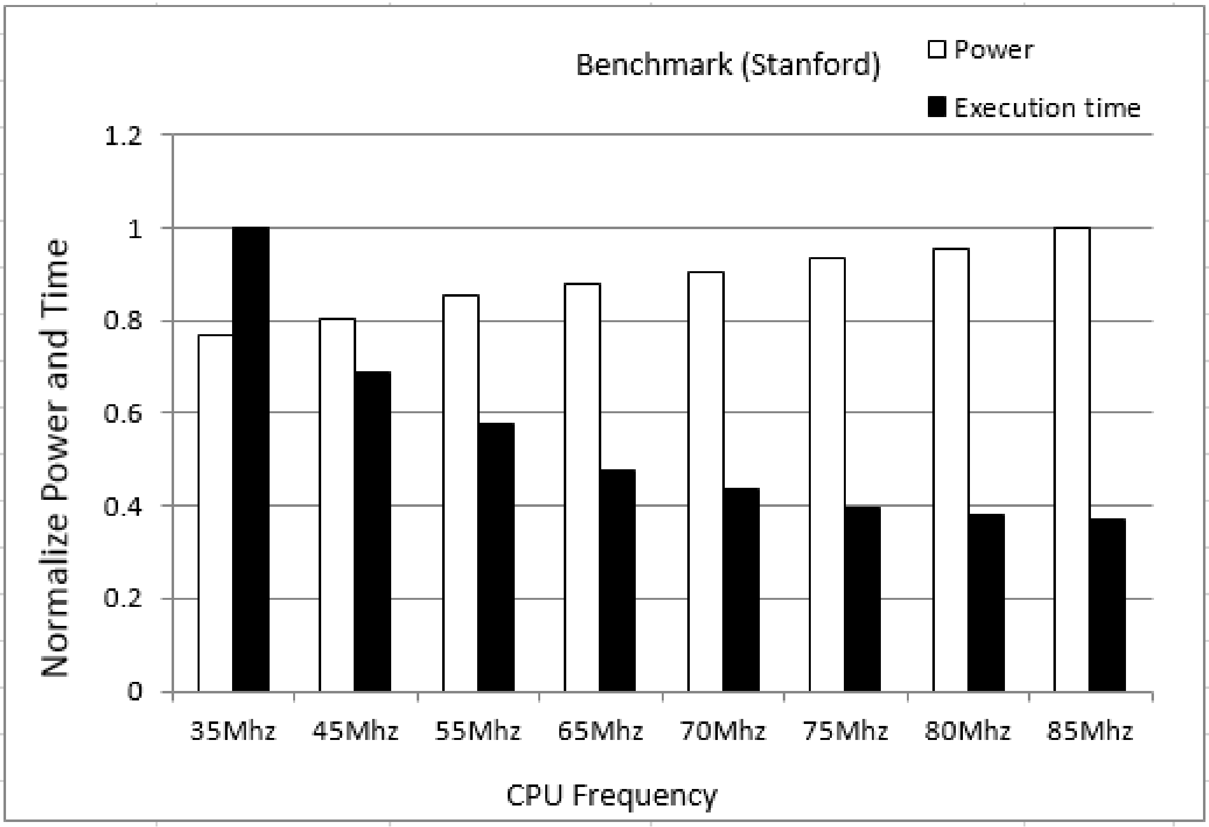


Figure 4.5: Normalized execution time and power consumption for Stanford benchmark application

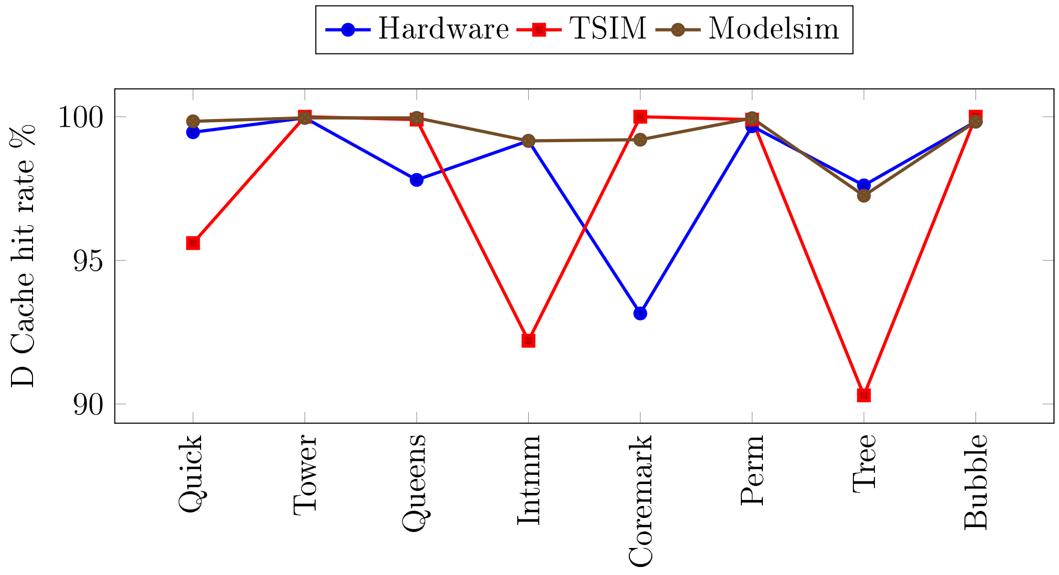


Figure 4.6: Comparison of D-Cache Hit rate with GRMON (H/W)

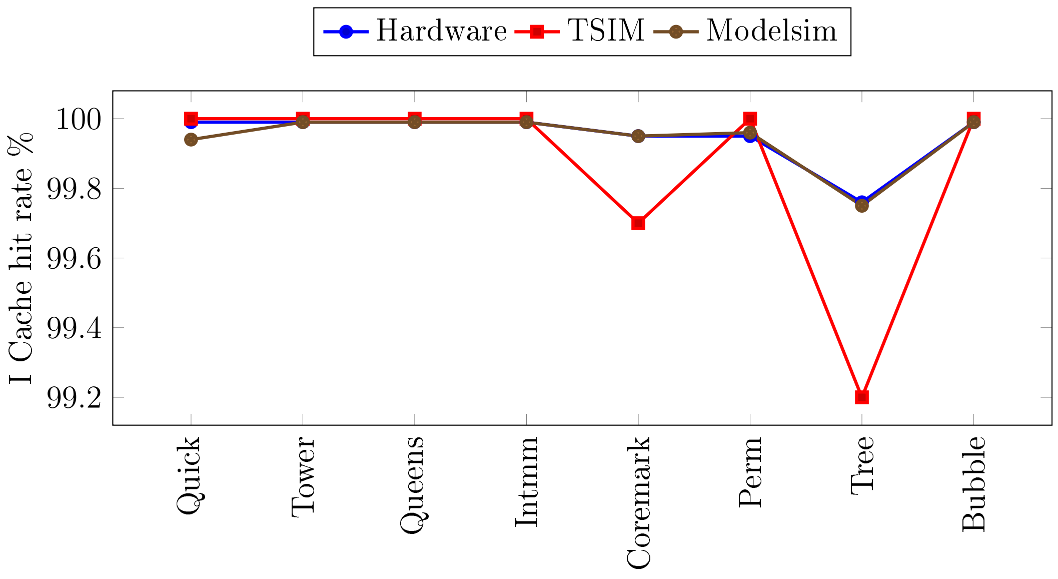


Figure 4.7: Comparison of I-Cache Hit rate with GRMON (H/W)

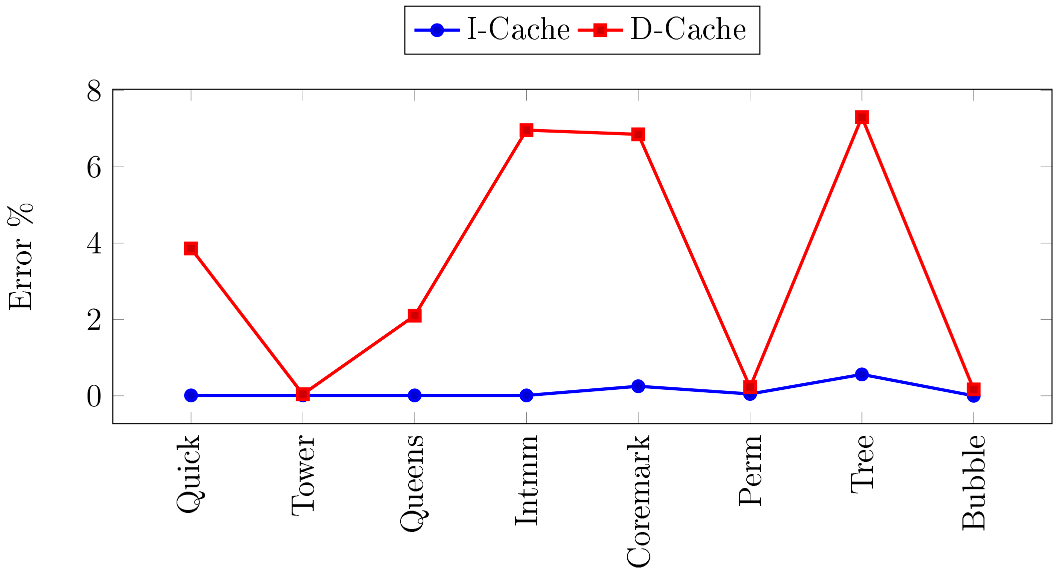


Figure 4.8: Error plot of I,D cache hit rate GRMON (H/W) VS TSIM

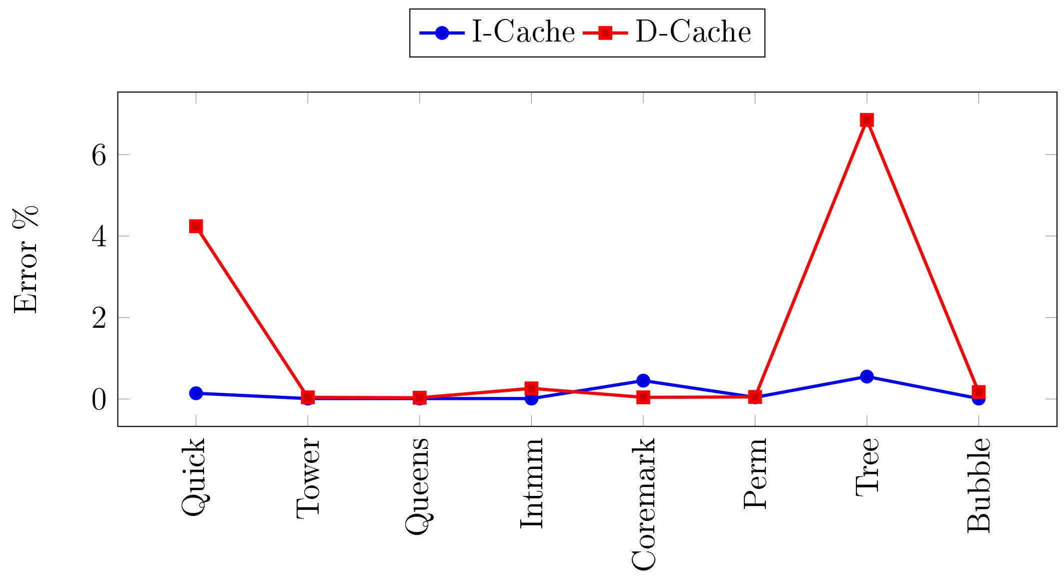


Figure 4.9: Error plot of I,D cache hit rate Modelsim (Simulation) VS TSIM

Chapter 5

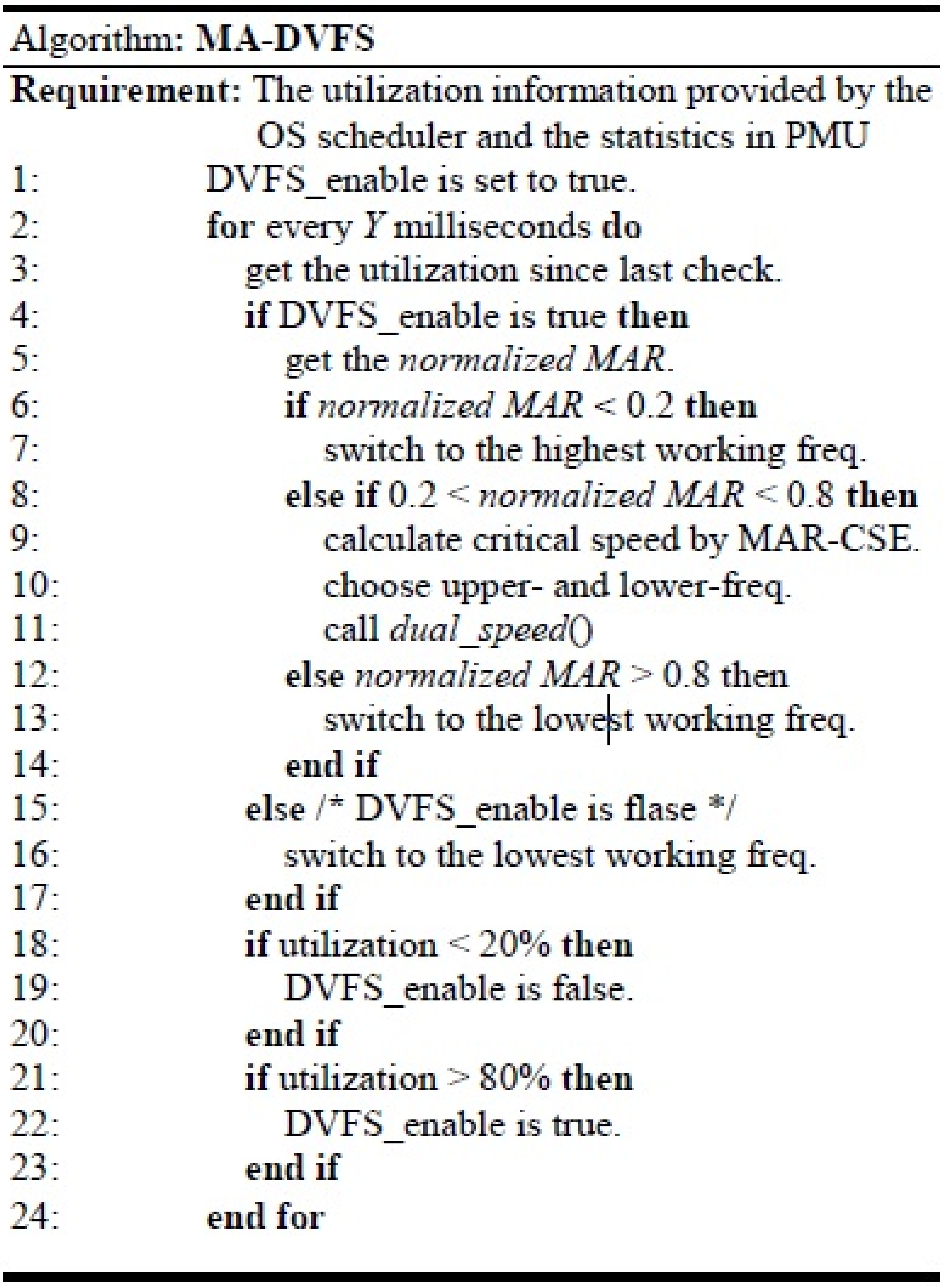
Conclusion

This dissertation has a major contribution in open source hardware design of LEON3 soft core processor. The hardware design of LEON3 written in VHDL hardware description language has been modified to include support for run-time reconfiguration of system frequency and real time processor statistics in terms of cache hit/miss ratio. The contribution of this dissertation allows developers and researchers to apply various intelligent algorithms based on memory access rate to reconfigure system frequency in order to optimize both performance and power consumption. Prior to this research work these algorithms cannot be validated on a LEON3 system as the open source LEON3 design neither supports run-time frequency scaling nor gives real time processor statistics which are pre-requisites to use memory aware algorithms.
This MS dissertation has produced two research articles accepted in international conferences in 2016. Research article titled *Real Time Implementation of DVFS Enhanced LEON3 MPSoC on FPGA* has been accepted in International Conference on Intelligent and Advanced System (ICIAS 2016) and research article titled *Enhancement of LEON3 processor for Real Time and Feedback Applications* has been accepted in International Seminar on Intelligent Technology and Its Applications (ISITIA - 2016).

5.1 Future Work

This MS dissertation has a great potential of future work, the contribution of this research work can be extended to implement memory aware algorithm for dynamic reconfiguration of hardware variable such as number if processor cores, cache size and system operating frequency which can lead to better performance and optimal power consumption. One such algorithm is presented in [22] where system operating frequency is dynamically reconfigured based on memory access rate of the running application. This algorithm has been implemented in Android and Linux operating system with underlying ARM architecture [22] [7]. As this MS dissertation comes with prerequisites of this algorithm, the MAR algorithm can be implemented on a LEON3 system to improve performance and power requirements of LEON3 which is specifically designed for space applications. The summary of MA-DVFS algorithm is presented in Figure 5.1 where MAR is the memory access rate defined by the following expression.

$$MAR = \frac{Instruction\,cache\,misses + Data\,cache\,misses}{Number\,of\,Executed\,Instructions}. \tag{5.1}$$



```
Algorithm: MA-DVFS
Requirement: The utilization information provided by the
             OS scheduler and the statistics in PMU
1:  DVFS_enable is set to true.
2:  for every Y milliseconds do
3:      get the utilization since last check.
4:      if DVFS_enable is true then
5:          get the normalized MAR.
6:          if normalized MAR < 0.2 then
7:              switch to the highest working freq.
8:          else if 0.2 < normalized MAR < 0.8 then
9:              calculate critical speed by MAR-CSE.
10:             choose upper- and lower-freq.
11:             call dual_speed()
12:         else normalized MAR > 0.8 then
13:             switch to the lowest working freq.
14:         end if
15:     else /* DVFS_enable is flase */
16:         switch to the lowest working freq.
17:     end if
18:     if utilization < 20% then
19:         DVFS_enable is false.
20:     end if
21:     if utilization > 80% then
22:         DVFS_enable is true.
23:     end if
24: end for
```

Figure 5.1: Pseudocode of MA-DVFS Algorithm.

Appendix A

VHDL Source Code - Dynamic Frequency Reconfiguration

```
entity clkgen_virtex5 is
  generic (
    clk_mul   : integer := 1;
    clk_div   : integer := 1;
    sdramen   : integer := 0;
    noclkfb   : integer := 0;
    pcien     : integer := 0;
    pcidll    : integer := 0;
    pcisysclk : integer := 0;
    freq      : integer := 25000;        -- clock frequency in KHz
    clk2xen   : integer := 0;
    clksel    : integer := 0);             -- enable clock select
  port (
          rst_adcm: in std_logic;
          cnt     : in std_logic_vector(2 downto 0);
          dcm_clk_off : out std_ulogic;
    clkin    : in  std_ulogic;
    pciclkin : in  std_ulogic;
    clk      : out std_ulogic;                      -- main clock
    clkn     : out std_ulogic;                      -- inverted main clock
    clk2x    : out std_ulogic;                      -- double clock
    sdclk    : out std_ulogic;                      -- SDRAM clock
    pciclk   : out std_ulogic;                      -- PCI clock
    cgi      : in clkgen_in_type;
    cgo      : out clkgen_out_type;
    clk1xu   : out std_ulogic;                      -- unscaled clock
    clk2xu   : out std_ulogic                       -- unscaled 2X clock
  );
end;

architecture struct of clkgen_virtex5 is

  component BUFG port (O : out std_logic; I : in std_logic);
```

```
end component;

component BUFGMUX port ( O : out std_ulogic; I0 : in std_ulogic;
                         I1 : in std_ulogic; S : in std_ulogic);
end component;

component DCM
  generic (
    CLKDV_DIVIDE : real := 2.0;
    CLKFX_DIVIDE : integer := 1;
    CLKFX_MULTIPLY : integer := 4;
    CLKIN_DIVIDE_BY_2 : boolean := false;
    CLKIN_PERIOD : real := 10.0;
    CLKOUT_PHASE_SHIFT : string := "NONE";
    CLK_FEEDBACK : string := "1X";
    DESKEW_ADJUST : string := "SYSTEM_SYNCHRONOUS";
    DFS_FREQUENCY_MODE : string := "LOW";
    DLL_FREQUENCY_MODE : string := "LOW";
    DSS_MODE : string := "NONE";
    DUTY_CYCLE_CORRECTION : boolean := true;
    FACTORY_JF : bit_vector := X"C080";
    PHASE_SHIFT : integer := 0;
    STARTUP_WAIT : boolean := false
  );
  port (
    CLKFB    : in  std_logic;
    CLKIN    : in  std_logic;
    DSSEN    : in  std_logic;
    PSCLK    : in  std_logic;
    PSEN     : in  std_logic;
    PSINCDEC : in  std_logic;
    RST      : in  std_logic;
    CLK0     : out std_logic;
    CLK90    : out std_logic;
    CLK180   : out std_logic;
    CLK270   : out std_logic;
    CLK2X    : out std_logic;
    CLK2X180 : out std_logic;
    CLKDV    : out std_logic;
    CLKFX    : out std_logic;
    CLKFX180 : out std_logic;
    LOCKED   : out std_logic;
    PSDONE   : out std_logic;
    STATUS   : out std_logic_vector (7 downto 0)


                  );
end component;
```

```
  COMPONENT adcm
        PORT(
                CLKIN_IN : IN std_logic;
                DADDR_IN : IN std_logic_vector(6 downto 0);
                DCLK_IN : IN std_logic;
                DEN_IN : IN std_logic;
                DI_IN : IN std_logic_vector(15 downto 0);
                DWE_IN : IN std_logic;
                RST_IN : IN std_logic;
                CLKFX_OUT : OUT std_logic;
                CLK0_OUT : OUT std_logic;
                DRDY_OUT : OUT std_logic;
                LOCKED_OUT : OUT std_logic
                );
        END COMPONENT;

  component BUFGDLL port (O : out std_logic; I : in std_logic);
  end component;

constant VERSION : integer := 1;
--constant CLKIN_PERIOD_ST : string := "20.0";
constant FREQ_MHZ : integer := freq/1000;

--attribute CLKIN_PERIOD : string;
--attribute CLKIN_PERIOD of dll0: label is CLKIN_PERIOD_ST;
signal gnd, clk_i, clk_j, clk_k, clk_l, clk_m, lsdclk : std_logic;
signal clk_x, clk_n, clk_o, clk_p, clk_i2, clk_sd, clk_r: std_logic;
signal dll0rst, dll0lock, dll1lock, dll2xlock : std_logic;
signal dll1rst, dll2xrst : std_logic_vector(0 to 3);
signal clk0B, clkint, pciclkint : std_logic;

----------------DCM_ADV--------------------------------
signal clkfx,clk0,locked,drdy,dcm_clk : std_logic;
signal di :  std_logic_vector(15 downto 0);
signal clkscale_m7d10 : std_logic_vector(15 downto 0)  := 16#0609#;
signal clkscale_m8d10 : std_logic_vector(15 downto 0)  := 16#0709#;
signal clkscale_m7d20 : std_logic_vector(15 downto 0)  := 16#0613#;
signal clkscale_m9d20 : std_logic_vector(15 downto 0)  := 16#0813#;
signal clkscale_m11d20 : std_logic_vector(15 downto 0) := 16#0A13#;
signal clkscale_m13d20 : std_logic_vector(15 downto 0) := 16#0C13#;
signal clkscale_m15d20 : std_logic_vector(15 downto 0) := 16#0E13#;
signal clkscale_m17d20 : std_logic_vector(15 downto 0) := 16#1013#;
-------------------------------------------------------
begin
        dcm_clk_off <= dcm_clk;
        clk_i <= clkfx when (locked = '1') else dcm_clk;
  gnd <= '0';
```

```
clk <= clk_i when (CLK2XEN = 0) else clk_p;
clkn <= clk_m; clk2x <= clk_i2;

c0 : if (PCISYSCLK = 0) or (PCIEN = 0) generate
  clkint <= clkin;
end generate;

c2 : if PCIEN /= 0 generate
  pciclkint <= pciclkin;
  p3 : if PCISYSCLK = 1 generate clkint <= pciclkint;
      end generate;
  p0 : if PCIDLL = 1 generate
    x1 : BUFGDLL port map (I => pciclkint, O => pciclk);
  end generate;
  p1 : if PCIDLL = 0 generate
    x1 : BUFG port map (I => pciclkint, O => pciclk);
  end generate;
end generate;

c3 : if PCIEN = 0 generate
  pciclk <= '0';
end generate;

clk1xu <= clk_k;
clk2xu <= clk_x;
bufg0 : BUFG port map (I => clk0B, O => dcm_clk);
bufg1 : BUFG port map (I => clk_j, O => clk_k);
bufg2 : BUFG port map (I => clk_l, O => clk_m);
buf34gen : if (CLK2XEN /= 0) generate
  cs0 : if (clksel = 0) generate
    bufg3 : BUFG port map (I => clk_n, O => clk_i2);
  end generate;
  cs1 : if (clksel /= 0) generate
    bufg3 : BUFGMUX port map (S => cgi.clksel(0),
        I0 => clk_o, I1 => clk_n, O => clk_i2);
  end generate;
  bufg4 : BUFG port map (I => clk_o, O => clk_p);
end generate;
dll0rst <= not cgi.pllrst;

  dll0 : DCM
    generic map (CLKFX_MULTIPLY => clk_mul,
                                        CLKFX_DIVIDE => clk_div,
                    DFS_FREQUENCY_MODE => "LOW",
                                 DLL_FREQUENCY_MODE => "LOW")
    port map ( CLKIN => clkint, CLKFB => clk_k,
                             DSSEN => gnd, PSCLK => gnd,
                   PSEN => gnd, PSINCDEC => gnd,
                                RST => dll0rst , CLK0 => clk_j ,
                   CLKFX => clk0B , CLK2X => clk_x ,
                                CLKFX180 => clk_l , LOCKED => dll0lock );
--   end generate;

        Inst_adcm: adcm PORT MAP(
                CLKIN_IN => clkint ,
                DADDR_IN => "1010000",
                DCLK_IN => clkint ,
                DEN_IN => '1',
                DI_IN => di ,
                DWE_IN => '1',
                RST_IN => rst_adcm ,
                CLKFX_OUT => clkfx ,
                CLK0_OUT => clk0 ,
                DRDY_OUT => drdy ,
                LOCKED_OUT => locked
        );


  clk2xgen : if (CLK2XEN /= 0) generate
      dll2x : DCM generic map
          (CLKFX_MULTIPLY => 2,
                CLKFX_DIVIDE => 2,
        DFS_FREQUENCY_MODE => "LOW",
                DLL_FREQUENCY_MODE => "LOW")
      port map ( CLKIN => clk_i , CLKFB => clk_p ,
                               DSSEN => gnd , PSCLK => gnd ,
                               PSEN => gnd , PSINCDEC => gnd ,
                               RST => dll2xrst(0), CLK0 => clk_o ,
                CLK2X => clk_n ,  LOCKED => dll2xlock );
--     end generate;
    rstdel2x : process (clk_i , dll0lock)
    begin
      if dll0lock = '0' then dll2xrst <= (others => '1');
      elsif rising_edge(clk_i) then
        dll2xrst <= dll2xrst(1 to 3) & '0';
      end if;
    end process;
  end generate;

  di_drive : process (cnt ,rst_adcm) is
  begin
  if (cnt = "000") then
   di <= clkscale_m7d10;
  elsif (cnt = "001") then
   di <= clkscale_m8d10;
        elsif (cnt = "010") then
      di <= clkscale_m7d20;
            elsif (cnt = "011") then
      di <= clkscale_m9d20;
            elsif (cnt = "100") then
      di <= clkscale_m11d20;
            elsif (cnt = "101") then
      di <= clkscale_m13d20;
            elsif (cnt = "110") then
      di <= clkscale_m15d20;
            else
      di <= clkscale_m17d20;
    end if;
     end process;

    clk_sd1 : if (CLK2XEN = 0) generate
      clk_i2 <= clk_x;
      dll2xlock <= dll0lock;
      clk_sd <= dcm_clk;
    end generate;

    clk_sd2 : if (CLK2XEN = 1) generate clk_sd <= clk_p; end generate;
    clk_sd3 : if (CLK2XEN = 2) generate clk_sd <= clk_i2; end generate;


    sd0 : if (SDRAMEN /= 0) and (NOCLKFB=0) generate
      cgo.clklock <= dll1lock;
        dll1 : DCM generic map
                          (CLKFX_MULTIPLY => 2,
                          CLKFX_DIVIDE => 2,
              DFS_FREQUENCY_MODE => "LOW",
                          DLL_FREQUENCY_MODE => "LOW",
              DESKEW_ADJUST => "SOURCE_SYNCHRONOUS")
          port map ( CLKIN => clk_sd, CLKFB => cgi.pllref,
                                        DSSEN => gnd, PSCLK => gnd,
                   PSEN => gnd, PSINCDEC => gnd, RST => dll1rst(0),
                                   CLK0 => lsdclk, --CLK2X => clk2x,
                   LOCKED => dll1lock);
--      end generate;
      bufgx : BUFG port map (I => lsdclk, O => sdclk);
      rstdel : process (clk_sd, dll2xlock)
      begin
        if dll2xlock = '0' then dll1rst <= (others => '1');
        elsif rising_edge(clk_sd) then
          dll1rst <= dll1rst(1 to 3) & '0';
        end if;
      end process;
    end generate;
```

```
  sd1 : if ((SDRAMEN = 0) or (NOCLKFB = 1)) and (CLK2XEN /= 2)
  generate
    sdclk <= clk_i;
    cgo.clklock <= dll0lock when (CLK2XEN = 0) else dll2xlock;
  end generate;

  sd1_2x : if ((SDRAMEN = 0) or (NOCLKFB = 1)) and (CLK2XEN = 2)
  generate
    sdclk <= clk_i2;
    cgo.clklock <= dll2xlock;
  end generate;
  cgo.pcilock <= '1';

end;
```

Appendix B

VHDL Source Code - Performance Monitoring Unit

B.1 Hardware source modification for I-Cache

```
 library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
library gaisler;
use gaisler.libiu.all;
use gaisler.libcache.all;
use gaisler.mmuconfig.all;
use gaisler.mmuiface.all;

entity mmu_icache is
  generic (
    icen       : integer range 0 to 1   := 0;
    irepl      : integer range 0 to 3   := 0;
    isets      : integer range 1 to 4   := 1;
    ilinesize  : integer range 4 to 8   := 4;
    isetsize   : integer range 1 to 256 := 1;
    isetlock   : integer range 0 to 1   := 0;
    lram       : integer range 0 to 1 := 0;
    lramsize   : integer range 1 to 512 := 1;
    lramstart  : integer range 0 to 255 := 16#8e#;
    mmuen      : integer                := 0
  );
  port (
    rst : in  std_logic;
    clk : in  std_logic;
    hitcounter_p : out std_logic_vector(31 downto 0);
    cache_access_count_p : out std_logic_vector(31 downto 0);
    hpc_rst    : in   std_ulogic;--najam added--
    holdn   : in std_ulogic;
    ici  : in  icache_in_type;
    ico  : out icache_out_type;
    dci  : in  dcache_in_type;
    dco  : in  dcache_out_type;
    mcii  : out memory_ic_in_type;
    mcio  : in  memory_ic_out_type;
    icrami  : out icram_in_type;
    icramo  : in  icram_out_type;
    fpuholdn  : in  std_logic;
    mmudci  : in  mmudc_in_type;
    mmuici  : out mmuic_in_type;
    mmuico  : in mmuic_out_type
);
end;

architecture rtl of mmu_icache is
begin
 hitcount: process(clk)
  begin
    if rising_edge(clk) then
          if(rst = '1') then
      if(ici.inull = '0') then
        if(holdn = '1') then
          if(r.hit = '1') then
            hitcounter <= hitcounter + 1;
          else
            hitcounter <= hitcounter;
          end if;
                        end if;
        end if;
                  else
                  hitcounter <= "00000000000000000000000000000000";
      end if;
    end if;

  if hpc_rst='1' then--modified--
    hitcounter <= "00000000000000000000000000000000";
  end if;
  end process;

  cache_access: process(clk)
  begin
    if rising_edge(clk) then
         if(rst = '1') then
      if (ici.inull = '0') then
        if(holdn = '1') then
          cache_access_count <= cache_access_count + 1;
          else
            cache_access_count <= cache_access_count;
          end if;
                    end if;
                    else
                    cache_access_count <= 16#0000#;
        end if;
    end if;
    if hpc_rst='1' then--modified--
      cache_access_count <= "00000000000000000000000000000000";
    end if;
    end process;
    end;
```

B.2 Hardware source modification for D-Cache

```
  d_hitcount: process(clk)
      begin
        if rising_edge(clk) then
                  if(rst = '1') then
          if(dci.enaddr and holdn and not dci.nullify) = '1' then
            if(r.hit = '1') then
              d_hitcounter <= d_hitcounter + 1;
            else
              d_hitcounter <= d_hitcounter;
            end if;
                          end if;
                          else
                          d_hitcounter <= 16#0000#;
          end if;
      end if;
      if hpc_rst='1' then--modified--
      d_hitcounter <= "00000000000000000000000000000000";
    end if;
      end process;

       dcache_accesscount: process(clk)
      begin
        if rising_edge(clk) then
                  if(rst = '1') then
          if(dci.enaddr and holdn and not dci.nullify) = '1' then
            dcache_accesscounter <= dcache_accesscounter + 1;
          else
            dcache_accesscounter <= dcache_accesscounter;
          end if;
                    else
                    dcache_accesscounter <= 16#0000#;
                    end if;
      end if;
    if hpc_rst='1' then--modified--
    dcache_accesscounter <= "00000000000000000000000000000000";
end if;
   end process;
```

Bibliography

[1] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom *et al.*, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International.* IEEE, 2010, pp. 108–109.

[2] D. E. D. G. V. A. M. S. U. Shruthi N, Prashant Kulshreshtha, "Performance Estimation of a LEON 3FT Processor Based Design for Spacecraft Applications," *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)*, vol. 9, no. 3, pp. 48–54, June 2014.

[3] A. Hofmann, R. Wansch, R. Glein, and B. Kollmannthaler, "An FPGA based on-board processor platform for space application," in *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on.* IEEE, 2012, pp. 17–22.

[4] S. De Florio, E. Gill, S. D'Amico, and A. Grillenberger, "Performance comparison of microprocessors for space-based navigation applications," in *Small Satellites for Earth Observation: 7th International Symposium of the International Academy of Astronautics (IAA), 4-8 May 2009, Berlin, Germany*, 2009.

[5] N. Shruthi and C. Vinay, "LEON 3FT Processor Based Design for Spacecraft Applications-Frequency Based Performance Analysis," *International Journal of Engineering Science Invention*, vol. 3, no. 12, pp. 21–27, 2014.

[6] D. Guzman, M. Prieto, S. Sanchez, J. Almena, O. Rodriguez, and D. Meziat, "Improving the LEON Spacecraft Computer Processor for Real-Time Performance Analysis," *Journal of Spacecraft and Rockets*, vol. 48, no. 4, pp. 671–678, 2011.

[7] Y. Liang, P. Lai, and C. Chiou, "An energy conservation dvfs algorithm for the android operating system," *Journal of Convergence*, vol. 1, no. 1, 2010.

[8] P. Patel, "Embedded systems design using fpga," in *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*, Jan 2006, pp. 1 pp.–.

[9] A. G. Schmidt, N. Steiner, M. French, and R. Sass, "HwPMI: an extensible performance monitoring infrastructure for improving hardware design and productivity on FPGAs," *International Journal of Reconfigurable Computing*, vol. 2012, p. 2, 2012.

[10] I. Zaidi, A. Nabina, C. N. Canagarajah, and J. Nunez-Yanez, "Power/area analysis of a fpga-based open-source processor using partial dynamic reconfiguration," in *Digital System Design Architectures, Methods and Tools, 2008. DSD'08. 11th EUROMICRO Conference on.* IEEE, 2008, pp. 592–598.

[11] B. Lin, A. Mallik, P. Dinda, G. Memik, and R. Dick, “User- and process-driven dynamic voltage and frequency scaling,” in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, April 2009, pp. 11–22.

[12] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. Ishii, T. Y. Nguyen, J. L. Burns *et al.*, “A 32-bit powerpc system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling,” *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 11, pp. 1441–1447, 2002.

[13] S. Jafri, M. Tajammul, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen, “Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in cgras,” in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, July 2013, pp. 104–112.

[14] E. Intel, “Speedstep® technology for the intel® pentium® m processor white paper, march 2004,” Recovered 30/1/2011 from World Wide Web: ftp://download. intel. com/design/network/papers/30117401. pdf, Tech. Rep., 2004.

[15] J. Kong, J. Choi, L. Choi, and S. W. Chung, “Low-cost application-aware dvfs for multi-core architecture,” in *Convergence and Hybrid Information Technology, 2008. ICCIT’08. Third International Conference on*, vol. 2. IEEE, 2008, pp. 106–111.

[16] Y.-W. Yang and K. S.-M. Li, “Temperature-aware dynamic frequency and voltage scaling for reliability and yield enhancement,” in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference.* IEEE Press, 2009, pp. 49–54.

[17] Z.-h. Wu, “Brain-machine interface (bmi) and cyborg intelligence,” *Journal of Zhejiang University Science*, vol. 15, pp. 805–806, 2014.

[18] T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors,” in *Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on.* IEEE, 1998, pp. 197–202.

[19] G. Dhiman and T. S. Rosing, “Dynamic voltage frequency scaling for multi-tasking systems using online learning,” in *Proceedings of the 2007 international symposium on Low power electronics and design.* ACM, 2007, pp. 207–212.

[20] K. Choi, R. Soma, and M. Pedram, “Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 18–28, 2005.

[21] A. Weissel and F. Bellosa, “Process cruise control: event-driven clock scaling for dynamic power management,” in *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems.* ACM, 2002, pp. 238–246.

[22] W.-Y. Liang, S.-C. Chen, Y.-L. Chang, and J.-P. Fang, “Memory-aware dynamic voltage and frequency prediction for portable devices,” in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA ’08. 14th IEEE International Conference on*, Aug 2008, pp. 229–236.

[23] D. Marculescu, "On the use of microarchitecture-driven dynamic voltage scaling," in *Workshop on Complexity-Effective Design*, vol. 42. Citeseer, 2000.

[24] S. Ghiasi, J. Casmira, and D. Grunwald, "Using ipc variation in workloads with externally specified rates to reduce power consumption," in *Workshop on Complexity Effective Design*. Citeseer, 2000.

[25] N. Ho, P. Kaufmann, and M. Platzner, "A hardware/software infrastructure for performance monitoring on LEON3 multicore platforms," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–4.

[26] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A performance counter architecture for computing accurate CPI components," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5, pp. 175–184, 2006.

[27] P.-H. Chen, C.-T. King, Y.-Y. Chang, and S.-Y. Tseng, "Multiprocessor system-on-chip profiling architecture: design and implementation," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*. IEEE, 2009, pp. 519–526.

[28] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-system chip multiprocessor power evaluations using FPGA-based emulation," in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*. IEEE, 2008, pp. 335–340.

[29] W. L. Bircher and L. K. John, "Complete system power estimation using processor performance events," *Computers, IEEE Transactions on*, vol. 61, no. 4, pp. 563–577, 2012.

[30] X. Liu, L. Shen, C. Qian, and Z. Wang, "Dynamic Power Estimation with Hardware Performance Counters Support on Multi-core Platform," in *Advanced Computer Architecture*. Springer, 2014, pp. 177–189.

[31] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[32] A. Asaduzzaman, F. N. Sibai, and A. Abonamah, "A dynamic way cache locking scheme to improve the predictability of power-aware embedded systems," in *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*. IEEE, 2011, pp. 756–759.

[33] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 234–237.

[34] X. Vera, B. Lisper, and J. Xue, "Data cache locking for tight timing calculations," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 1, p. 4, 2007.

[35] E. Mezzetti, N. Holsti, A. Colin, G. Bernat, and T. Vardanega, "Attacking the sources of unpredictability in the instruction cache behavior," in *16th International Conference on Real-Time and Network Systems (RTNS 2008)*, 2008.

[36] J. Arulraj, P.-C. Chang, G. Jin, and S. Lu, “Production-run software failure diagnosis via hardware performance counters,” in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1. ACM, 2013, pp. 101–112.