ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electronics Engineering (29^{th}cycle)

Energy-efficient hardware design based on high-level synthesis

By

Fahad Bin Muslim

******

Supervisor(s):
Prof. Luciano Lavagno, Supervisor

Doctoral Examination Committee:
Prof. Roberto Guerrieri, Universita' di Bologna
Prof. Antonio Abramo, Universita' di Udine
Ing. Davide Quaglia, Universita' di Verona
Prof. Leonardo Reyneri, Politecnico di Torino
Prof. Claudio Passerone, Politecnico di Torino

Politecnico di Torino
2017

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Fahad Bin Muslim
2017

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents

Acknowledgements

First and foremost, I would like to thank Allah-the Almighty for all His blessings bestowed upon me. Without His will, I would not have made it this far.

I would like to express a special gratitude to my supervisor and mentor Prof. Luciano Lavagno for the excellent ideas that he proposed for me to work on. The work was challenging but it was through his commendable guidance which enabled me to get some useful results eventually. He was always there to address my work-related as well as other general queries even with his extremely busy schedule. Besides, I would like to thank all the faculty members and researchers from the department of electronics and telecommunications (DET) at Politecnico di Torino, who supported me in any way during the course of my PhD.

I would also like to thank my several friends that I made during my stay here in Turin for making my stay extremely pleasant and memorable. I would also like to take this opportunity to acknowledge the support of my group mates (past and present) at the High-level synthesis group at Polito. The discussions with them have always been extremely rewarding and I learnt a lot from them.

A special word of appreciation for my family members including my parents, wife, siblings and others for their unconditional support and prayers. They always had extremely encouraging words for me especially, when the chips were down.

Finally, I am extremely thankful to the higher education commission (HEC) Pakistan for funding my doctorate at the Politecnico di Torino. This is a great initiative by the government of Pakistan to create a pool of high quality researchers who can ultimately contribute to the prosperity of the nation. I find myself more equipped after my PhD to contribute to this goal.

Abstract

This dissertation describes research activities broadly concerning the area of High-level synthesis (HLS), but more specifically, regarding the HLS-based design of energy-efficient hardware (HW) accelerators. HW accelerators, mostly implemented on FPGAs, are integral to the heterogeneous architectures employed in modern high performance computing (HPC) systems due to their ability to speed up the execution while dramatically reducing the energy consumption of computationally challenging portions of complex applications. Hence, the first activity was regarding an HLS-based approach to directly execute an OpenCL code on an FPGA instead of its traditional GPU-based counterpart. Modern FPGAs offer considerable computational capabilities while consuming significantly smaller power as compared to high-end GPUs. Several different implementations of the K-Nearest Neighbor algorithm were considered on both FPGA- and GPU-based platforms and their performance was compared. FPGAs were generally more energy-efficient than the GPUs in all the test cases. Eventually, we were also able to get a faster (in terms of execution time) FPGA implementation by using an FPGA-specific OpenCL coding style and utilizing suitable HLS directives.

The second activity was targeted towards the development of a methodology complementing HLS to automatically derive power optimization directives (also known as "power intent") from a system-level design description and use it to drive the design steps after HLS, by producing a directive file written using the common power format (CPF) to achieve power shut-off (PSO) in case of an ASIC design. The proposed LP-HLS methodology reduces the design effort by enabling designers to infer low power information from the system-level description of a design rather than at the RTL. This methodology required a SystemC description of a generic power management module to describe the design context of a HW module also modeled in SystemC, along with the development of a tool to automatically produce the CPF file to accomplish PSO. Several test cases were considered to validate the proposed

methodology and the results demonstrated its ability to correctly extract the low power information and apply it to achieve power optimization in the backend flow.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

ALU Arithmetic and Logic Unit

ASIC Application Specific Integrated Circuit

BRAM Block RAM

CG Clock Gating

CGIC Clock-Gated Integrated Cells

CMOS Complimentary Metal Oxide Semiconductor

CPF Common Power Format

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

DRAM Dynamic RAM

DSE Design Space Exploration

DVFS Dynamic Voltage Frequency Scaling

FIFO First-In First-Out

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

HDL Hardware Description Language

HLS High-Level Synthesis

HPC High Performance Computing

IDCT Inverse Discrete Cosine Transform

KNN K-Nearest Neighbor

MSV Multi-Supply Voltage

OpenCL Open Computing Language

PMB Power Management Block

PSO Power Shut-Off

QoR Quality of Results

RAM Random Access Memory

RCA Ripple-Carry Adder

RTL Register Transfer Level

SoC System On Chip

TCF Toggle Count Format

UPF Unified Power Format

Chapter 1

Introduction

Modern electronic devices, driven by exceeding market requirements, are required to perform a variety of tasks. Considering the example of mobile handsets; they were initially meant to mainly support voice calls and text messaging i.e. short messaging service (SMS). Modern smart phones, in comparison, are required to support a variety of sophisticated features such as video calls, voice over IP (VoIP) and multimedia messaging to name a few. The support for such increasing features in modern electronic devices is provided by advanced system-on-chip (SoC) designs consisting of heterogeneous system architectures.

Such heterogeneous systems essentially consist of a combination of multi-core processors and hardware accelerators for speeding up the execution of compute intensive operations while consuming considerably lower power [1], [2]. Traditionally, graphics processing units (GPUs) have been used as accelerators in combination with central processing units (CPUs) but unfortunately high performance computing (HPC) systems based on GPUs are inefficient in terms of their power consumption [3]. Modern field programmable gate arrays (FPGAs) fortunately have the ability to offer considerable execution speed while consuming only a fraction of the power as compared to several high-end GPUs [4]. These FPGAs, hence are strong competitors to the traditional GPU-based accelerators and are part of heterogeneous systems in modern HPC systems.

It can be concluded from the discussion above that power consumption in modern SoCs is as critical as their computational abilities. Hardware designers are always looking to develop designs which are optimal both in terms of power and timing

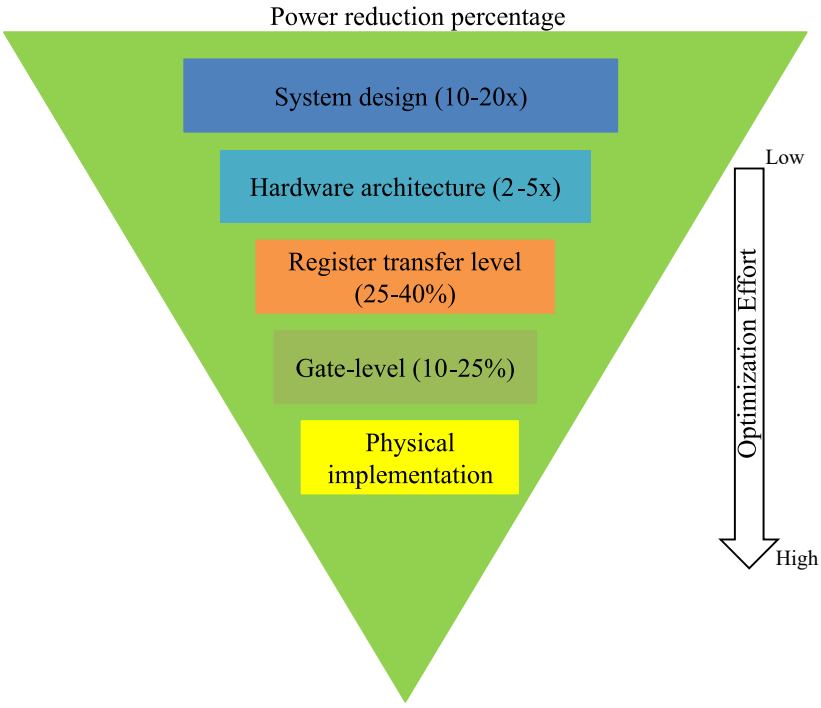

Fig. 1.1 Power saving vs design effort over different levels of a design flow

which unfortunately are conflicting performance parameters. It is easier to achieve pareto-optimality, if the power is considered at a higher level of abstraction i.e. at the system-level of the design flow. This is because major architectural decisions related to cost, performance and power consumption are made at higher abstraction level [5]. The opportunity to save power decreases as we go down the design flow from system-level to the register transfer level (RTL) and below, while the optimization effort increases. This is depicted in Fig. 1.1.

1.1 FPGA based heterogeneous computing system

Modern HPC systems are meant to analyze diverse range of complex phenomena in a variety of fields such as data mining, fault simulation and fluid dynamics to name a few. This requires the modern computers to provide huge computational abilities in a sustainable manner. There has been a general consensus since long that constant performance improvements in modern computers can not be obtained merely by increasing their operating parameters e.g. the processor clock speed. While such

measures may yield performance enhancement, they also result in a higher power consumption than is desirable for modern HPC systems [6].

One of the solutions to this issue is to deploy systems that utilize accelerators e.g. GPUs and FPGAs, in combination with multi-core processors to speed up the execution of compute-intensive operations while still consuming considerably less energy. Additionally, it is desirable to expose parallelism in applications e.g. by using various parallel programming languages, so as to better exploit the parallel architectures offered by these accelerators. Such systems offer sufficient execution speed along with providing better energy efficiency [2].

GPUs are known to provide huge computational power and hence have been used extensively for HPC applications. However, with their continuously increasing computational capabilities, GPUs are also known to be extremely inefficient as far as their energy consumption is concerned [7]. The energy consumption can be significantly reduced while still having a considerable amount of computational power by utilizing FPGA-based accelerators. FPGAs are well known for their reconfigurability as well as their energy efficiency. This fact has been recognized in the industry as evident from the decisions by Microsoft and Baidu to use FPGAs as accelerators rather than GPUs in their respective search engines [8],[9].

A major limitation while utilizing FPGAs in modern heterogeneous systems is the complexity in programming them. Traditional FPGA programming requires sufficient expertise in one of the several hardware description languages (HDL) e.g. Verilog and VHDL. Hardware designers normally use these languages for FPGA programming but several useful algorithms are usually written in non hardware-specific languages e.g. C/C++. It requires a lot of effort on the part of hardware designers to port the algorithms for implementing them on FPGAs [10]. Furthermore, describing hardware at such low-level languages limits the designers to explore only a limited number of architectural options due to their much slower design and verification cycles [11].

These issues can be rectified by an approach called High-level Synthesis (HLS) which enables the designers to implement their algorithms, written in several higher level languages such as C, C++ and SystemC, directly on FPGAs. HLS causes a significant reduction in both the design and verification time and effort as compared to an HDL design. Additionally, it allows designers to have several considerably different hardware implementations, satisfying a variety of design constraints, from

a single high-level code by merely providing different directives to the HLS tool, a process called design space exploration (DSE) [12], [13].

Recognizing the abilities of FPGAs as competitors to the traditional GPU-based accelerators and the programming complexity affiliated with them, various HLS tools have been developed recently by the leading FPGA manufacturers e.g. Xilinx and Altera, for their respective devices. These HLS tools have been developed to directly support the RTL synthesis starting from the description of an algorithm in a parallel programming framework namely OpenCL developed by the Khronos group to enable execution of applications on heterogeneous platforms.

OpenCL is based upon C/C++ and is used to expose parallelism in an application to enable speed up by exploiting the concurrency offered by the hardware accelerators e.g. GPUs. It holds an advantage over the very similar Compute Unified Device Architecture (CUDA) programming framework by NVIDIA. This is due to the fact that unlike CUDA being used to program NVIDIA GPUs only, OpenCL offers higher execution portability thus enabling the execution of the same OpenCL code on a variety of hardware platforms e.g. CPUs and GPUs. With the development of HLS tools using OpenCL code as an input, the same code used to execute the application on a CPU/GPU can now be used for FPGA implementation as well with the same optimization effort as that on CPU/GPU-based platforms [14].

As mentioned before, OpenCL offers execution portability but unfortunately, it does not offer performance portability. This implies that even though the same OpenCL code can be executed on a variety of hardware platforms, the performance would vary from one device to another. OpenCL implementation and optimization support on Xilinx FPGAs is provided by the SDAccel™ tool chain from Xilinx, utilizing tools from Vivado® design suite for RTL and logic synthesis [15], [16].

1.2 High-level synthesis based low power methodology

Fig. 1.1 shows that maximum power can be saved by specifying power intent at the system-level of the hardware design flow. This would also require the least optimization effort as compared to the cases where optimization is considered at lower levels of abstraction. In a conventional design flow, the designers must

optimize manually written RTL to satisfy strict power requirements by applying numerous power optimization techniques to optimize both the leakage power as well as the dynamic power. This process involves taking into account a complex multi-dimensional problem space while considering a variety of low-level information regarding the RTL design within a strict time limitation. This obviously becomes extremely difficult and the solution is to raise the level of abstraction above the RTL for easier power optimization and to utilize RTL synthesis automation tools [17]. The power optimization techniques usually target supply voltage and clock control methodologies e.g. power shut-off (PSO), dynamic voltage frequency scaling (DVFS), clock gating (CG) and multi-supply voltage (MSV). PSO is particularly useful to reduce leakage power in hand-held devices which are usually powered by energy sources (e.g. batteries) that can only provide a limited amount of energy [18].

Previously, the total power dissipation in a complimentary metal oxide semiconductor (CMOS) was dominated by the dynamic power but leakage power has now become an equally significant contributor with the prevailing nanometric technologies. It can be assumed that this situation would continue to worsen with the continuous scaling of technology. Fig. 1.2 clearly shows this trend [19]. Dynamic power consumption in CMOS occurs during the active mode when the signals through the CMOS toggle hence changing their logic states, resulting in the charging and discharging of load capacitors [20]. Leakage power on the other hand consists of *active leakage* and *standby leakage*. The standby leakage power consumption occurs during circuit sleep mode while the active leakage is caused by the leakage current that still flows even in the operation mode. The active leakage becomes a significant contributor to the leakage power in deeply scaled devices [21].

As discussed before, HLS takes a high-level description of a design and automatically generates several RTL descriptions satisfying different area/performance/power constraints thus reducing both the design and verification time and effort. It certainly is desirable to have a fully automated low power flow, enabling the incorporation of both logic as well as power into the design starting from the system-level. This would require a methodology to automatically capture the power intent of a design at the system-level while using HLS to achieve a broad set of target system implementations.

A broad overview of such a methodology is shown in Fig. 1.3. Unlike the traditional methodology involving power intent definition together with the RTL,

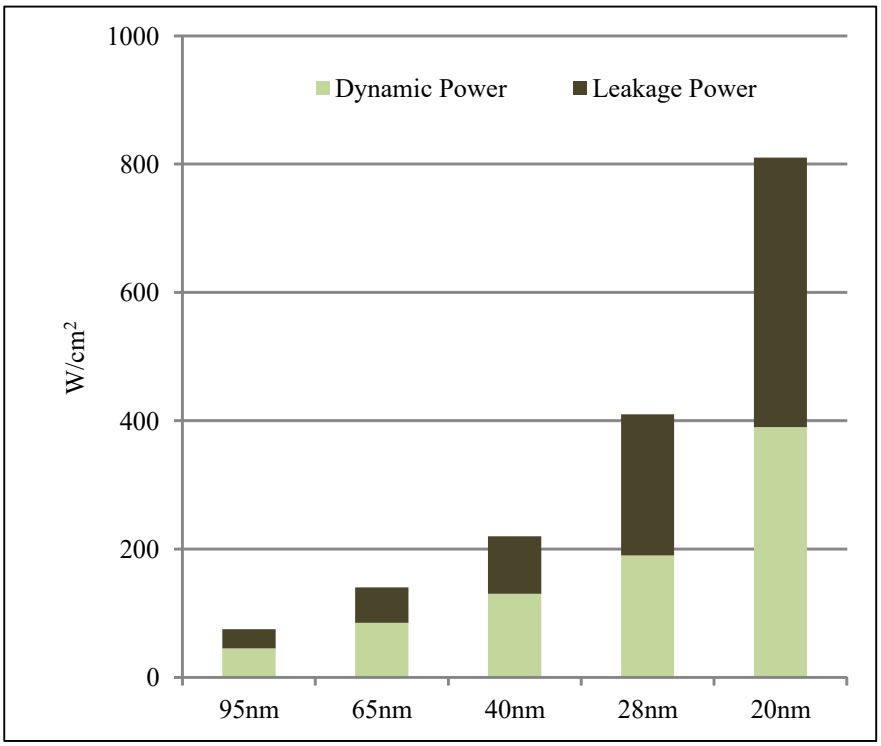

Fig. 1.2 Power versus Technology in CMOS

this methodology involves defining power intent directly using the system-level code and using HLS to automatically generate the RTL. This is followed by applying the already defined power intent in the later stages of the design flow to achieve power optimization. High-level synthesis preserves the naming convention of the design hierarchy including instances, signals and ports. This in turn enables designers to directly define power intent on the system-level code, which is more convenient to understand owing to its higher level of abstraction. Additionally, adding power management logic at the system-level makes functional verification simpler via the design-specific system-level testbench [5].

1.3 Problem Statement

This dissertation deals with some issues regarding high-level synthesis in general and low-power hardware accelerator design in particular. FPGAs can be categorized as reconfigurable digital hardware which have been used in a variety of applications ranging from signal processing to high performance switches. They also represent

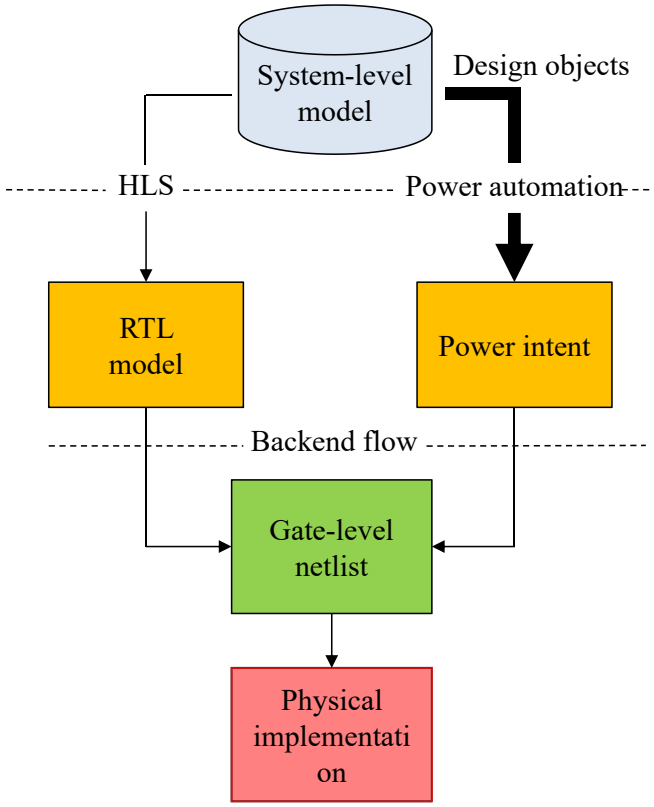

Fig. 1.3 Overview of HLS based low power design methodology

excellent options as accelerators due to their flexibility, energy efficiency and high performance. FPGAs are usually programmed using hardware description languages like verilog or VHDL. To be proficient in these languages, one needs to have considerable knowledge about computer architecture and hardware design [10]. This programming complexity affiliated with FPGAs has been the reason for reluctance shown by designers in using FPGAs as accelerators. HLS, however, gives us a way around this issue by allowing us to program the FPGAs directly through C/C++ and other programming frameworks based on them e.g. SystemC and OpenCL. With this issue being taken care of, FPGAs can be strong competitors to the prevailing trend of using GPUs as hardware accelerators.

With regards to obtaining energy-efficient application specific integrated circuit (ASIC) designs, describing power intent at the system-level is still a stiff challenge for system architects and designers. Power optimization is usually considered at the RTL in a conventional design flow when most of the architectural decisions have already been made. It is thus desired to have a methodology that enables the description of both the behavioral functionality as well as the power intent at a higher level of abstraction while using HLS tools at the front end of the proceeding design flow. There, however, exists no commercial tools or methodology presently for this purpose and the required effort must still be done manually. A brief overview of the methodology which can yield this automation has been shown in Fig. 1.3.

1.4 Contribution

One of the goals of the thesis is to explore a methodology using HLS to implement an application written in OpenCL on an FPGA-based accelerator. The main contributions from this activity are:

- investigation of the issues encountered when implementing and optimizing a code, written in non-hardware specific OpenCL on a Xilinx FPGA device. The methodology is based on HLS and it is intended to apply a variety of HLS optimization attributes and techniques and see how an OpenCL code responds to them to yield the desired performance on an FPGA.
- secondly, it is also intended to emphasize on the difference in compilation of an OpenCL code to be executed on a GPU and an FPGA, stemming mainly

due to their different architectures. While GPUs due to their fixed architecture, support just-in-time compilation, the FPGAs due to their flexible architecture, allow a more thorough exploration of various code optimizations than allowed by just-in-time compilation. The OpenCL standard thus, allows offline compilation of the OpenCL code to be executed on an FPGA.

- finally, it is shown by comparing the performance of multiple algorithm implementations on various platforms that the code executing efficiently on an FPGA is very different from the one leading to the best implementation on a GPU.

The thesis also deals with the development of a methodology based on HLS to automatically generate power optimization directives to achieve PSO (leading to reduced leakage power) in an ASIC design in the form of a common power format (CPF) file by using the relevant information, extracted from the system-level design e.g. the module to be shut-off, as well as other important information e.g. technology specifications. CPF is a standard endorsed by the low power coalition at Silicon Integration Initiative (Si2) and can be used to describe power intent for a design [22].

It should be noted that large FPGAs (with several transistors) also suffer from huge leakage power consumption. This is because of the increase in leakage component of each of the several transistors embedded into modern heavily scaled FPGAs. Furthermore, in large FPGAs, a sizeable portion of the available resources may remain unutilized if the design does not fill up the device completely. This would result in leakage power due to both the utilized and unutilized parts of the FPGA [23].

The authors in [24] have proposed a method to reduce the leakage power significantly in FPGAs by shutting down individual accelerators dynamically during idle periods at run-time through a technique called *dynamic power-gating*. The same authors in [25] also present a technique to achieve fine-grained power gating by shutting down selected portions of hierarchical designs with large accelerators i.e. sub-accelerators, while the rest of the sub-accelerators are still running. The identification of idle states along with their duration however, in FPGAs is very difficult to achieve thus discouraging designers to make the required effort. Power gating in ASIC designs however is much more common, wherein, the opportunities for power gating can be identified manually and exploited using standard file formats

such as unified power format (UPF) or CPF [23]. Hence, it is worthwhile to note that while many of the optimizations used during the course of the first activity are relevant both for ASIC and FPGA designs, the second activity (concerned with using CPF for power gating) pertains exclusively to an ASIC design. The main contributions from this activity are:

- exploring issues encountered while obtaining a fully automated low power ASIC design flow starting from the system-level description to the physical design. This would involve the development of a generic system-level power management module to enable firstly the specification and later on the application of the power intent to achieve a power efficient design.
- development of a tool that can automatically generate the power optimization directives to achieve PSO for a given system design context.

1.5 Organization of the thesis

This thesis presents a collection of the work done in the field of electronic design automation using high-level synthesis with an emphasis on power/energy efficient designs. The first activity deals with an exploration of the prospects of using FPGAs as accelerators in modern heterogeneous systems by utilizing HLS along with their performance comparison with some high-end GPUs. This activity is covered in this document from chapters 2 to 4. The second activity explores the idea of developing a fully automated low power design methodology complementing the HLS to obtain power-efficient ASIC designs. The details can be found in chapters 5, 6 and 7. Chapter 8 concludes the work along with mentioning some future directions in which both the activities can be pursued. A brief description of each chapter is presented here.

Chapter 2: Basic theory regarding heterogeneous system architectures is presented here with reference to both GPU- and FPGA-based acceleration.

Chapter 3: This chapter gives a description of OpenCL programming language which is a parallel programming framework and enables execution portability over a variety of device platforms.

Chapter 4: This chapter compares FPGAs versus GPUs in terms of execution speed, power and energy consumption. This is done by utilizing a widely used classification algorithm i.e. K-Nearest Neighbor (KNN), as a test case.

Chapter 5: This chapter explains the details regarding system-level power management module which enables the provision of signals necessary to achieve power gating thus yielding a power-efficient ASIC design.

Chapter 6: The description of our proposed low power HLS-based methodology (LP-HLS) is presented here along with a description of the tool that we developed to automatically derive the power optimization directives necessary to achieve PSO.

Chapter 7: This chapter presents the design test cases that were used to validate our LP-HLS methodology. The results validating our proposed methodology are also presented.

Chapter 8: The work is finally concluded along with a brief description of the future directions in which the activities can be pursued.

A sample CPF file used for power gating one of our test cases i.e. an Inverse Discrete Cosine Transform (IDCT) design in a JPEG IDCT decoder is provided in Appendix A.

Chapter 2

Heterogeneous System Architecture

A heterogeneous system refers to a system comprising of several different processors and cores. Such multi-core architectures offer high performance along with better power efficiency by not only using additional processor cores but by using specialized hardware called *accelerators* to handle certain computationally challenging portions of the applications.

2.1 Why Heterogeneous architecture?

Besides the complexity i.e. the number of transistors per square inch of a processor, the processor frequency also traditionally follows the Moore's low very closely. This trend of a continuous increase in the frequency has however been hindered by certain physical constraints e.g. the power density [26]. The equation for power density specifically depicting the influence of frequency is given by (2.1):

$$P = C\rho f V_{dd}^2 \tag{2.1}$$

where P is the power density i.e. the power dissipated per unit area, C represents the total capacitance, ρ is the transistor density i.e. the number of transistors per unit area, f represents the processor frequency and V_{dd} is the supply voltage [27].

It should be noted that (2.1) ignores the leakage power which contributes significantly to the over all power consumption in CMOS sub-micron technologies [28].

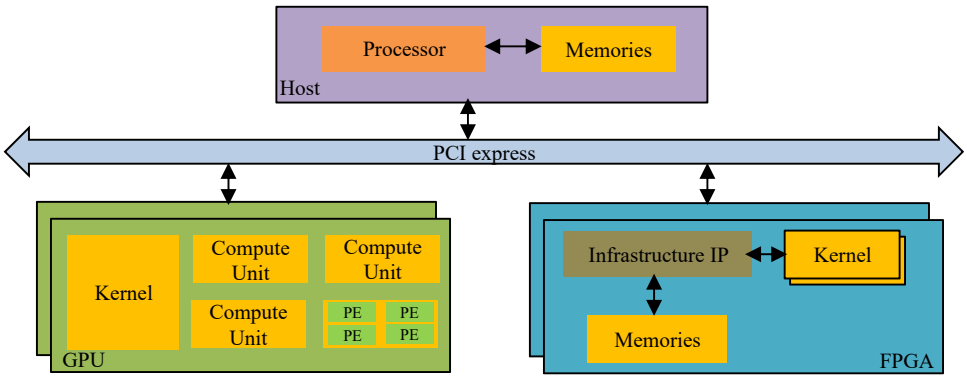

Fig. 2.1 A Typical Heterogeneous System Architecture

As evident from (2.1), the clock frequency of the processor can not be increased unbounded due to the so called *power wall*.

One of the solutions to improve performance while still keeping the power consumption in check, is to use multi-core processors running at lower frequencies and supply voltages than the single core processors. For example, a dual core design running at 85% of the supply voltage and frequency offers 180% better performance while still consuming approximately the same power as a single core design [26]. The issue with the multi-core processors in use presently though, is that most of their resources are spent on logic and cache thus, resulting in majority of the power being consumed by non-computational units.

Heterogeneous architectures give us an alternative solution to this issue. These architectures utilize the multi-core processors in combination with high performance accelerators for a given power or transistor budget. This essentially means that the accelerators use fewer transistors and operate at lower frequencies as compared to the CPUs hence consuming lower power/energy. These accelerators typically do not operate standalone and rely on traditional processors to manage them. This implies that the CPUs are responsible to explicitly manage data transfer and execution while using the accelerator cores [26], [29]. The accelerators used in such heterogeneous systems may be GPUs, FPGAs or a combination of both. In the terminology of heterogeneous system architecture, the multi-core processor is typically called a *host* while the hardware platforms used to accelerate certain portions of the applications are called *devices*. A typical heterogeneous system is shown in Fig. 2.1. The various important components of such a heterogeneous system are described here briefly.

2.2 Multi-core processors

As mentioned earlier, increasing clock frequency to achieve better processor performance is not a viable option due to the corresponding significant increase in energy, heat dissipation e.t.c as represented by (2.1). One way of countering this is to lower the clock frequency while packing more processor cores in a chip i.e. the *multi-core processors*. These multiple cores will share some resources e.g. memory, network etc but are still capable of handling independent operations. Such processors enable the developers to exploit both data-level and task-level parallelism. To fully exploit the capabilities of such multi-core processors, the programmers may need to write multi-threaded codes, utilize libraries offering shared memory parallelism e.g. OpenMP or use a message passing library e.g. MPI [10].

2.3 Graphics Processing Units

Another option to speed up the execution of parallelizable portions of the algorithms is to use GPU-based accelerators as co-processors. A GPU is in principle a device consisting of an extremely parallel microprocessor and a private memory with a very high access bandwidth. They were originally developed to cater to the increasing demand for hardware accelerated 3D graphics. The interest in the use of GPUs for HPC applications started developing with the introduction of CUDA parallel programming framework by NVIDIA in 2007. GPUs are meant to execute in parallel, the same set of instructions on different data in single instruction multiple data (SIMD) fashion. Unlike multi-core CPUs, the GPUs are designed in such a way so as to devote more transistors to data operations instead of data caching and control [26].

A GPU consists of several parallel processing elements called streaming multiprocessors to execute the kernel functionality in a parallel manner. Each streaming multiprocessor further consists of multiple cores, with each core made up of several components such as arithmetic and logical units (ALUs), thread-schedulers, load/store units, scratchpad memories, caches etc. The cache size in GPUs is much smaller as compared to the CPUs as they are designed for stream or throughput computing involving smaller data reuse as compared to the CPUs. A GPU always acts as a device in combination with a CPU being used as a host for loading data

intensive tasks on the GPU and offloading the results along with managing the data transfers involved in the process. A GPU consists of its own device memory of a few gigabytes (GBs) and is connected to the host through a PCI-Express (PCIe) bus as shown in Fig. 2.1 [7].

2.4 Field Programmable Gate Arrays

While GPUs offer great computational abilities that can be exploited while using them as accelerators, they unfortunately have a very poor power efficiency [7], [3]. FPGAs provide an alternate option as accelerators offering considerable computational abilities while still consuming considerably small amount of power/energy as compared to the several modern high end GPUs. This is mainly because of the control systems in FPGAs being hardwired hence, eliminating the need to fetch, decode and execute instructions. Furthermore, the on-chip SRAM in case of FPGAs are better customizable to the specific applications, thereby cutting on the multiplexing energy costs. While FPGAs were initially used for discrete logic, there has been a drastic expansion in their fields of usage ranging from signal processing to high performance embedded computing and more recently in high performance computing [26]

FPGAs offer a highly parallel architecture which can be used to achieve a considerable amount of acceleration. A typical FPGA consists of logic blocks, memory blocks and DSP slices each surrounded by programmable interconnects as shown in Fig. 2.2. The FPGAs offer high performance along with high versatility and power efficiency owing to their conceptually simpler design [26]. The idea of using FPGAs as accelerators normally suffers due to the complexity involved in programming them. This issue however, can be resolved by using e.g. SDAccel™ tool chain from Xilinx which enables us to program Xilinx FPGAs directly using the OpenCL parallel programming language. The tool chain includes both the Vivado high-level synthesis tool as well as the logic and physical design tools from the Vivado® design suite [15], [16].

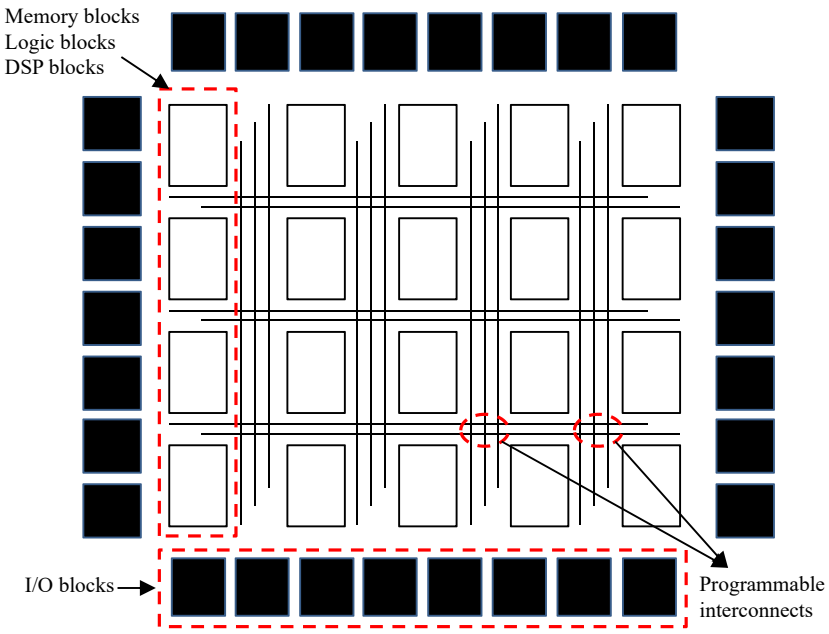

Fig. 2.2 FPGA Architecture

Chapter 3

Open Computing Language

OpenCL is a parallel programming framework for programming multi-core and heterogeneous compute platforms [30], [31]. Thus, it lies at the intersection of the programming languages corresponding to the individual computing platforms constituting a heterogeneous system as shown in Fig. 3.1. OpenCL offers execution portability thus enabling code execution on various supporting devices through minimal modifications to the host code. The programming language is based on C99 and supports both data-parallel and task-parallel programming models [26].

3.1 Platform Model

The OpenCL platform model mainly consists of a multi-core CPU called a *host*. Host is responsible for setting up the environment to enable an OpenCL program to execute on one or more devices. In terms of OpenCL, a *device* represents any supported hardware platform that can be used to accelerate the compute intensive portions of an application referred to as the *kernels*. An OpenCL device consists of compute units (CU) each further divided into processing elements (PE) as shown in Fig. 3.2. Several concurrent executions of the kernel body (called *work-items*) takes place on multiple processing elements. The work-items are further grouped into *work-groups* which are being executed by multiple compute units.

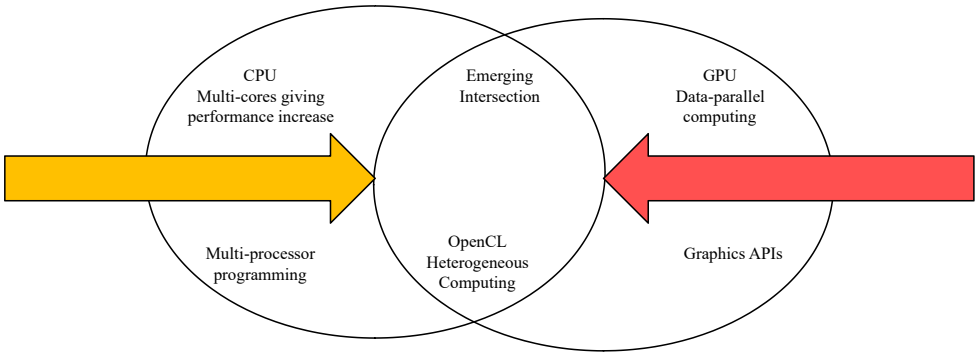

Fig. 3.1 OpenCL for heterogeneous programming

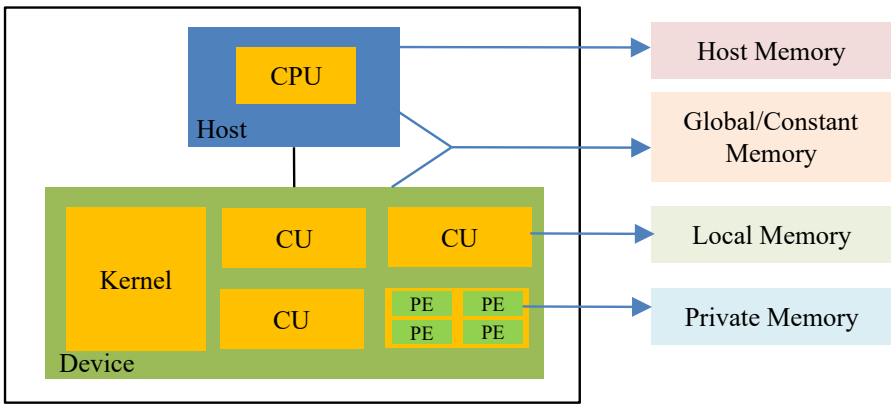

Fig. 3.2 Platform and Memory Model of OpenCL

3.2 Memory Model

The memory is broadly divided into host (i.e. CPU) memory and device (i.e. GPU or FPGA) memory. The device memory is further divided into private memory (specific to each work-item). This memory is the smallest i.e. O(10) words per work-item but is the fastest to access at the same time. Local memory is shared by all the work-items in a work-group and is around O(1-10) kbytes per work-group. This is also slower than the private memory. Global/constant memory is shared by all the work-groups. Global memory is around O(1-10) Gbytes while constant memory is around O(10-100) Kbytes. Access to the global memory is the slowest among all the device memories. Finally, the host memory resides on the CPU and can be few Gbytes in size. The OpenCL memory model is also shown in Fig. 3.2. It should be noted that memory management in OpenCL is done explicitly i.e. by moving data from host memory to global memory to local memory and then back.

3.3 Synchronization in OpenCL

The work-items in an OpenCL kernel are executed in an out-of order manner to ensure high performance by relieving the programmer i.e. by extracting parallelism from the code automatically and speeding it up on a given hardware platform [32]. Since the execution order of work-items across different devices can not be ascertained, OpenCL standard introduces the concept of *barriers* to ensure memory consistency. A barrier represents a check point with in a work-group such that all the work-items belonging to that work-group must reach this point before any of them can proceed with the rest of the computations [15]. Synchronization in the execution of work-items belonging to different work-groups is not possible in OpenCL.

Chapter 4

Battle of Accelerators: FPGA or GPU?

This chapter describes the adopted HLS-based methodology to implement a popular classification algorithm i.e. the K-Nearest Neighbor algorithm, on Xilinx FPGAs. Multiple implementations of the algorithm are considered and their performance on FPGA and GPUs is compared as well.

4.1 KNN Algorithm

K-Nearest Neighbor (KNN) algorithm is an important algorithm for classification finding applications in a diverse range of fields such as computer vision, pattern recognition and machine learning etc. KNN can be used to detect the k nearest neighbors of a specific query point among severel reference data points. Usually, the training datasets are very large, thus causing the computation cost of the algorithm to be very large [33]. Fortunately, the algorithm consists of a high level of parallelism and hence, we can accelerate it considerably by utilizing the parallel architectures of GPUs or FPGAs. The algorithm consists of the following steps:

1. For given n number of points in the reference data set R and a specified query point q, find the n distances between the query point and each point in the reference data set. Squared Euclidean distance is used here i.e. for two bi-

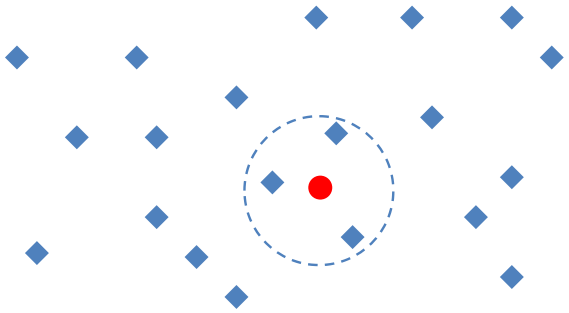

Fig. 4.1 Illustration of the KNN algorithm with k=3 and n=20

dimensional points (x_1, y_1) and (x_2, y_2), the Squared Euclidean distance is given by (4.1).

$$d = (x_1 - x_2)^2 + (y_1 - y_2)^2 \tag{4.1}$$

2. Sort the n distances calculated in step 1 while maintaining the corresponding indices of the points in the reference data set R.

3. Return the k points in the reference data set R relative to the k smallest distances obtained from step 2.

For a set R of n reference (training) data points in a d-dimensional space and a query point q, the k-nearest neighbor algorithm returns the k points in R that are closest to the query point q. This is illustrated for $k = 3$ and $n = 20$ in Fig.4.1. The red sphere represents the query point while the blue diamonds represent the points of the reference data set.

4.2 Related Work

The use of FPGAs as an alternative option to the traditional GPUs for acceleration has already been highlighted before. Some relevant work in this domain is presented here. In addition, this section of the thesis also mentions some work done previously to accelerate the KNN algorithm.

A thorough performance comparison between a CPU, a GPU and an FPGA implementation of a complex computer vision algorithm targeting linear structure detection has been presented in [34]. The authors in that work demonstrated that

the FPGA used i.e. the Xilinx Spartan LX150 FPGA outperformed both the AMD Radeon HD6870 GPU and the Intel Core i7 processor in terms of both the power consumption and the execution speed. OpenCL was used in that activity to port the code from CPU to GPU while VHDL was used for the implementation on their FPGA counterpart. The manual effort for this translation was also studied. This effort was certainly found to be higher for the VHDL-based FPGA implementation than for the OpenCL-based GPU implementation. We however, used an HLS-based approach for FPGA implementation starting directly from the OpenCL-based GPU implementation. This certainly causes a considerable reduction in the overall design effort along with giving us the ability to generate multiple hardware implementations from a single high-level OpenCL code merely by providing different directives to the HLS tool.

The authors in [10] have performed a very detailed performance comparison of multiple hardware accelerators for several implementations of the quantum Monte Carlo application. Numerous programming languages i.e. CUDA, OpenCL, C++, Brook+ and VHDL, have been used for various hardware platforms such as Intel multi-core CPUs, several GPUs from Radeon and NVIDIA and a Xilinx Virtex 4 LX160 FPGA. The analysis carried out in that paper for a large number of computations resulted in the combination of CUDA and NVIDIA GPUs providing the best performance while the FPGA performed the worst. The authors identified the main reason for this, as using older FPGA against extremely powerful GPUs and CPUs. The authors in that work also described their experience regarding the complexity of programming FPGAs through VHDL which took them about an year to perform.

An FPGA-based heterogeneous platform for KNN implementation was presented in [35]. Compilation of the OpenCL code onto the FPGA was carried out by using Altera's OpenCL compiler. A variety of hardware platforms were considered e.g. an Intel Core i7-3770 processor, an AMD Radeon HD7950 GPU and a Stratix IV 4SGX530 FPGA from Altera. The authors in that paper managed to obtain an FPGA implementation that outperformed both the GPU and CPU in terms of power/energy consumption per computation. The GPU implementation however was found to be faster than the FPGA implementation, most probably due to the GPU's higher global memory access bandwidth.

Various parallelization techniques corresponding to the nearest neighbor algorithm were surveyed in [36]. The author strongly emphasized both the requirement as

well as the opportunity to parallelize such algorithms. Acceleration of a brute force nearest neighbor algorithm through GPUs (while utilizing CUDA and CUBLAS library) was proposed in [33],[37]. Obviously, a huge increase in execution speed as compared to a highly optimized C++ library was obtained as a result.

It can thus be concluded that FPGA can be a favorable option in comparison to GPU for acceleration especially, when energy-per-computation is the main deciding factor. The designers in Baidu are thus weighing their options to use FPGAs as accelerators for their deep learning models for image search [8]. Microsoft also recently announced their decision of using FPGAs as accelerators in combination with Intel processors in their Bing search engine [38]. Considering the demand for FPGA-based acceleration in combination with their programming complexity, main FPGA manufacturers i.e. Altera and Xilinx, have also recently developed HLS tools enabling designers to implement the OpenCL codes directly on their respective FPGAs [15], [39]. This is hence, a hot topic for the design community at present and hence this motivated us to carry out an extensive research work in this regard.

4.3 Methodology

This section describes our adopted methodology to implement the algorithm on an FPGA starting from its OpenCL code. The power analysis flow is presented as well.

4.3.1 FPGA Implementation

Fig. 4.2 depicts the various steps performed by the SDAccel tool from Xilinx for direct FPGA implementation of an input OpenCL code. The flow begins with the functional verification of the OpenCL code through a software (SW) based simulation called CPU emulation in the context of an SDAccel-based flow. CPU emulation represents the fastest step of this design flow enabling a quick verification of the design functionality. An x86-based CPU is used in this step to execute both the host and kernel codes [15].

This step normally consists of adding a testbench to the host code, which performs the same functions in software as done in hardware and then compares the results from both. The testbench setup generates stimuli to drive the data and control ports of

the design under test (DUT). Furthermore, it monitors the output thereby validating the functionality of the DUT. Once the functionality is verified, the performance of each individual kernel i.e. the performance of an individual compute unit (in case of OpenCL, it is a work-group) and its resource usage are estimated. This gives an early estimate of the eventual performance gains by considering the targeted hardware platform along with the generated compute units for executing the application.

This is followed by RTL simulation using the same testbench that was used for CPU emulation. This step in the SDAccel-based flow is called hardware (HW) emulation and is used for functional verification of the compute units which are created for all the kernels and for their overall performance analysis. While CPU emulation ensures the functional accuracy of the application, its functionality on the hardware is verified by HW emulation. SDAccel needs to generate the logic implementation for each compute unit before HW emulation. This step hence, takes longer to complete in comparison with the CPU emulation. Vivado HLS is run under the hood in this step for custom logic generation corresponding to the application hence maximizing performance and minimizing resource utilization at the same time. Vivado™ Integrated Design Environment (IDE) is utilized afterwards in the *build application* step for connecting the generated custom units to the infrastructure IPs provided by the target hardware, such as the interface for the processor which is used to pass arguments to the kernel to start its execution and wait for its completion and the DDR DRAM interface [15]. Finally the generated system is packaged to be deployed on the supported FPGA-based boards.

4.3.2 Power Analysis

The power consumption in case of the FPGA implementation is estimated by utilizing the power analysis capabilities of Vivado®. Vivado supports power estimation through all the steps encompassing the FPGA design beginning with the logic synthesis up to the "place and route" stage. We perform power estimation in this work after the design is routed. This is because, the power analysis at this stage is the most accurate as it is based on the exact logic and routing resources being read from the already implemented design database [40]. The complete power analysis flow based on the Vivado power analysis features is shown in Fig. 4.3.

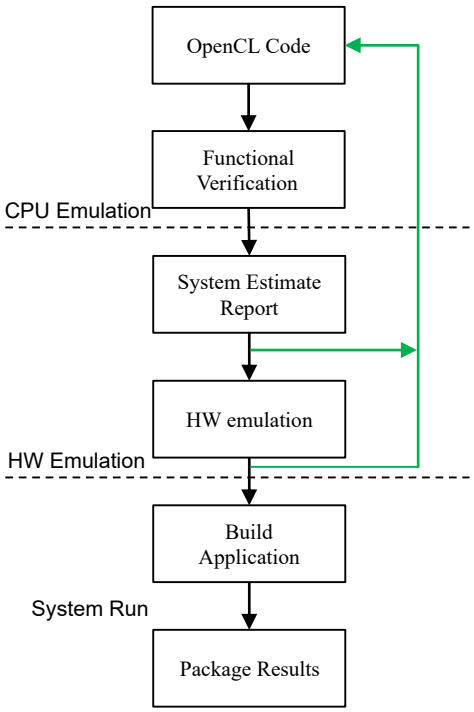

Fig. 4.2 SDAccel Based FPGA Design Methodology Flow

The power analysis can either be performed by using the default switching probabilities assigned to the design primary inputs by Vivado. The tool propagates these values to all the internal signals as well. This method is not as accurate as the vector-based power estimation. The vector-based estimation consists of RTL simulation and extraction of switching information of the design by performing activity profiling using test vectors provided to the design input ports as stimuli. The vector-based estimation is more accurate but the process takes considerably longer time to complete. In comparison, power estimation, based on default switching probabilities, provides a good trade-off between the accuracy in power estimation and the compute efficiency [40].

We used the vector-based approach to estimate power more accurately in the case of FPGA implementation. Switching Activity Interchange format (SAIF) file was used to capture the switching activities for the design which were then utilized to obtain accurate power reports. The GPU power on the other hand was estimated by using the NVIDIA system interface utility (nvidia-smi) that exploits the NVIDIA Management Library (NVML) and can be used to profile and manage all the NVIDIA GPUs installed on a specific platform [41].

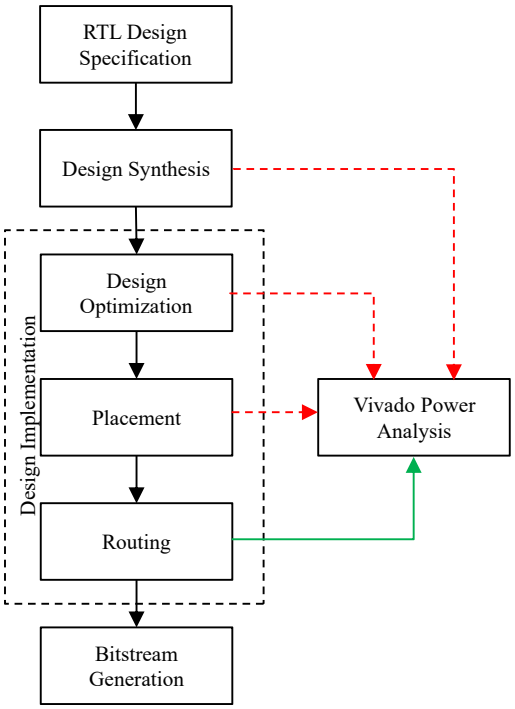

Fig. 4.3 Power Estimation and Analysis Flow

4.4 Test case implementations

The baseline code in this activity is based on the parallel implementations from [42], [43]. Two significantly different versions of the KNN algorithm are considered and their GPU- and FPGA-based implementations compared in terms of execution speed, energy and power performances. The implementations differ mainly based on whether the neighbor estimation is performed on the processor or on the accelerator. In case where the CPU estimates the neighbors, the CPU execution time is also considered while calculating the execution time of the algorithm. The two implementations are presented here.

4.4.1 Implementation 1

This implementation uses a parallel execution of the distance calculation task of the KNN algorithm on the device (GPU/FPGA), while the nearest neighbor estimation is performed on the host. The distance calculation task is readily parallelizable, as distinct independent points are read from the reference data set for calculating

several distances. The implementation utilizes global memory accesses only and hence its performance mainly depends on the global memory access bandwidth of the various accelerators. It is illustrated in Implementation 1.

Implementation 1: Distance calculation on device and neighbors on host

Input: A query point q and R, a set of reference points;
Output: Indices of the k reference points with the smallest distance from q;
Begin
On device:
function DISTANCE CALCULATION
for *each reference point* $r \in R$ **do**
    compute the floating-point distances between q and all points $r \in R$;
end
end function
On host:
function NEIGHBOR ESTIMATION
for $i = 0$ *to* $k-1$ **do**
    print the index in R of the $i-th$ smallest element of the sorted distance vector;
end
end function
End

4.4.2 Implementation 2

This implementation performs both the distance calculation as well as the neighbor estimation task on the device in two separate kernels namely "DISTANCE CALCULATION" and "NEIGHBOR ESTIMATION" respectively. It utilizes an automatic optimization offered by SDAccel called "On-chip global memories". This option automatically maps the global memory buffers used merely for inter-kernel communication, to the on-chip block RAMs. This optimization is depicted in Fig. 4.4. It should be noted that the global memory buffers are normally mapped to external slower DRAMs. This is shown in Fig. 4.4 as well. The pseudocode for this implementation is given in Implementation 2.

This implementation makes sense with reference to acceleration of the KNN algorithm only for the dimensionality of each point of R being high, thus making

the distance computation task more dominant as compared to finding the k smallest distances.

Implementation 2: KNN on device using multiple kernels

Input: A query point q and R, a set of reference points;
Output: k smallest floating-point distances with their indices in a single work-group;

```
1  Begin
2  On device:
3  declare a floating-point global distance array "dist" for inter-kernel
     communication;
4  function KERNEL1: DISTANCE CALCULATION
5  for each reference point r ∈ R do
6      compute all the floating-point distances between q and all points r ∈ R and
         save in "dist";
7  end
8  end function
9  function KERNEL2: NEIGHBOR ESTIMATION
10 for i = 0 to k − 1 do
11     print the index in R of the i − th smallest element of the distance vector;
12 end
13 end function
14 End
```

4.5 Experimental Setup

The experimental setup consists of three target devices shown in Table. 4.1. The first device is an NVIDIA GeForce GTX960 GPU with 1024 cores and a maximum operating frequency of 1178MHz. The device has about 2GB GDDR5 of global memory, with 112GB/s of memory bandwidth. It is accessible from the host through a PCIe 3.0 interface with 16 lanes. The second device is an NVIDIA Quadro K4200 GPU with 1344 CUDA cores and a maximum clock frequency of 784MHz. The device has about 4GB of GDDR5 global memory, with 172.8GB/s of memory bandwidth. It is accessible from the host through a PCIe Gen2 interface with 16 lanes. The third device is an Alpha data ADM-PCIE-7V3 FPGA board with a Virtex-7 690t. The global memory consists of two DDR3 memories with 21.3GB/s of bandwidth. The host can access it through a PCIe Gen3 interface with 8 lanes.

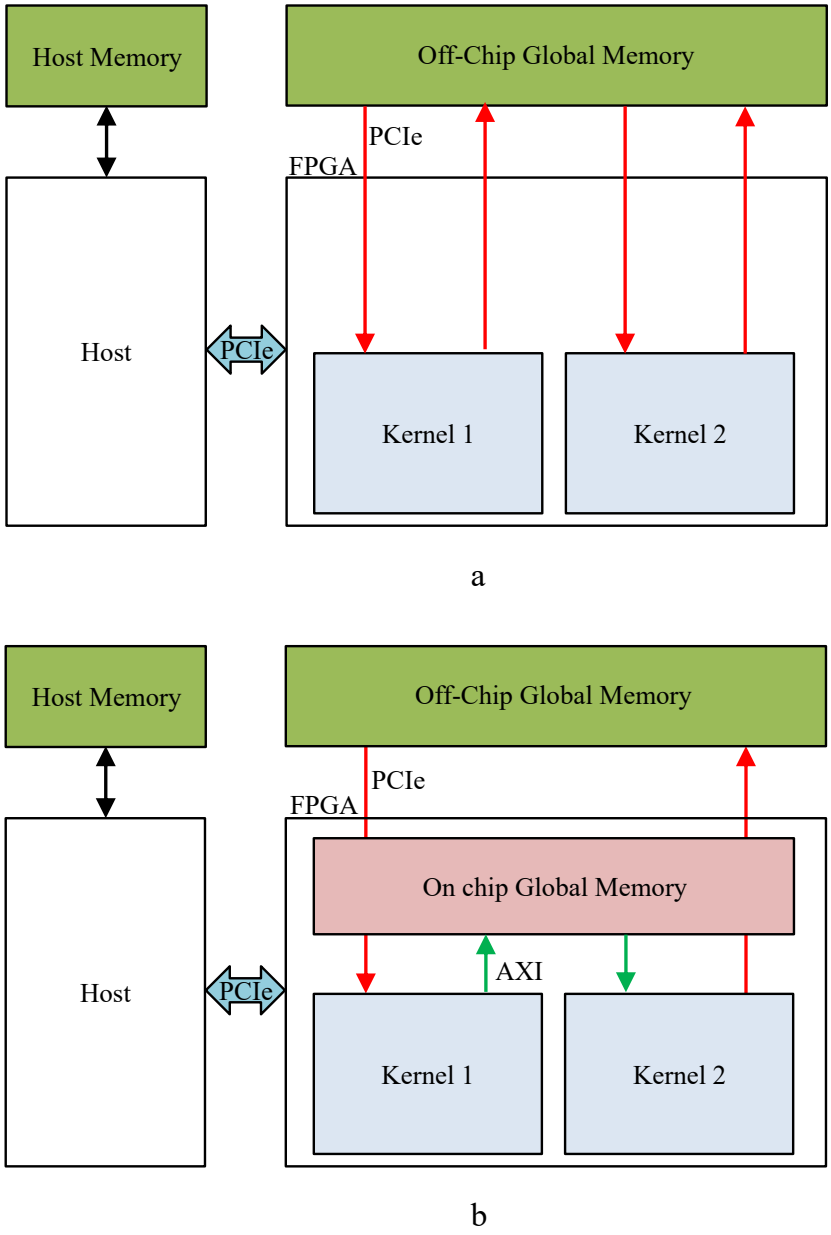

Fig. 4.4 (a) Traditional global memory buffer vs (b) On-chip global memory buffer

Table 4.1 Target Platforms Comparison

Device	Global Memory Size	Bandwidth (GB/s)	Bus Interface	min t_{clk}	Datasheet Power (W)	Idle Power (W)
GTX960	2GB GDDR5	112.0	PCIe 3.0 x16	0.85ns	120	8
K4200	4GB GDDR5	172.8	PCIe 2.0 x16	1.27ns	108	13
FPGA	Two 8GB SODIMMs	21.3	PCIe 3.0 x8	*	-	-

* See the rest of the tables for the reported clock and the actual power values in each test case.

4.6 Results

The experiments with the KNN algorithm use the data set from [44]. It contains data from "Unisys corporation" consisting of locations (latitudes and longitudes) of a number of hurricanes and is used by the KNN algorithm to find the locations corresponding to k nearest hurricanes to a given query point. The value of k is typically very small as compared to the number of points n in the reference data set. k has been set to 5 in all our experiments. The number of points in the reference data set is about 0.3 million.

In case of the FPGA implementation, the concurrency offered by the FPGA is utilized by using several HLS-based optimizations offered by SDAccel. The (*reqd_work_group_size*) attribute described by the OpenCL standard has been used in both implementations to specify the number of work-items in a single work-group. This in turn specifies the iteration count of the work-item loop which enables the HLS tool to optimize performance while the custom logic for the kernel is being generated. 2-element vector data types were used in both cases (rather than the C structs) to read the 2-dimensional data points, thereby causing the memory access throughput to be improved. Another optimization which was used in both the implementations is the use of burst transfers between the the off-chip global memory and the on-chip local memory. Large bursts improve efficiency as the memory access overhead is shared across large amounts of data being transferred [14].

Moreover, loop pipelining was used as well to improve throughput. The SDAccel-based flow can pipeline both the work-item loops as well as any explicit loops in the

kernel. Pipelining overcomes the limitations of loop unrolling for loops accessing global memory as in our case, by better matching the limited number of global memory ports available. The limited number of global memory ports may result in data access conflicts thereby limiting the performance gains obtained by loop unrolling by serializing potentially parallel loop iterations [45].

The GPU versus FPGA results in terms of execution time, energy and power consumption for Implementation 1 are presented in Table 4.2. The resource utilization in case of FPGA is also given in the table. This implementation utilizes the accelerators merely for distance calculation between the query point and all the points of reference data set. The nearest neighbor estimation on the other hand is performed by the host i.e. processor, hence the CPU time is added as well to the table as a part of the total KNN execution time. The reported clock frequency by Vivado HLS in this case is 240MHz.

As clear from Table 4.2, both GPUs in Implementation 1 are faster than the FPGA due to their comparatively higher DRAM access bandwidth. The FPGA however, consumes significantly smaller energy/power in comparison to both the GPUs. As mentioned before, power analysis in case of FPGA is done by using power analysis features of Vivado. The reported power in case of the GPUs is based on the results we obtained by utilizing the NVIDIA system management interface utility.

The performance comparison for Implementation 2 is shown in Table 4.3 This implementation is also operating at 240MHz clock frequency. This implementation uses the "on-chip global memories" optimization option offered by SDAccel to map the global memory buffers used for communication between multiple kernels to the block RAMs. A global memory buffer called "dist" as shown in Implementation 2 has been used for inter-kernel communication. This is an automatic optimization provided by SDAccel for the cases where it detects a global memory buffer which is not required to be visible to the host.

The FPGA implementation in this case is significantly faster than both the GPUs. The two kernels are executed sequentially on the GPUs and the slower DRAM is utilized. Reasons contributing to the high latency of DDR lies in the complexity. The DDR interface uses a controller to manage the refresh cycles, address multiplexing and interface timing [46]. In addition to latency, these frequent refresh cycles cause a higher power overhead as well [47]. These kernels on FPGA however, utilize the block RAMs (i.e on-chip global memories) and are executed in a pipelined manner.

Table 4.2 Performance analysis of implementation 1

Parameters/Devices	FPGA	GTX960	K4200
Device time	1.24ms ($t_{clk} = 4.17ns$)	0.04ms	0.05ms
CPU sort time	3.0ms	3.0ms	3.0ms
Total time	4.24ms	3.04ms	3.05ms
Power (Device)	0.346W	30W	40W
Energy (Device)	0.43mJ	1.2mJ	2mJ
Utilization	BRAMs = 0	NA	
	DSPs = 12 (0.33%)		
	FFs = 3109 (0.36%)		
	LUTs = 2006 (0.46%)		

Table 4.3 Performance analysis of implementation 2

Parameters/Devices	FPGA	GTX960	K4200
Total time	1.23ms ($t_{clk} = 4.17ns$)	0.93s	3.11s
Power	2.56W	90W	60W
Energy	0.003J	84J	187J
Utilization	BRAMs = 512 (34.83%)	NA	
	DSPs = 12 (0.33%)		
	FFs = 23892 (2.78%)		
	LUTs = 11838 (2.76%)		

Considering that the on-chip global memory is implemented on the FPGA itself, it has low latency and high throughput. Moreover, the NEIGHBOR ESTIMATION kernel also performs faster on FPGA than on the GPU. This is because, GPUs do not handle conditionals very efficiently, while they can still be pipelined on an FPGA. These conditionals on the GPUs create the so-called "thread divergence" problem. This issue arises due to the fact that on such Single Instruction Multiple Data processors, the work-items, for which the condition is adjudicated as false, must stall while the rest of the work-items are executed and vice-versa. The FPGA also out-performs both GPUs in terms of power and energy consumption. The FPGA power consumption in this case however, is around seven times higher than the FPGA implementation of Implementation 1. This is because of the excessive block RAM accesses that were not present in Implementation 1.

The FPGA vs GPU performance comparison for few other important algorithms, i.e. Montecarlo methods for financial models and bitonic sorting algorithms, has

Table 4.4 Summary of FPGA vs GPU performance results for various test cases

Test cases	Criticality	Best case FPGA		Best case GPU		Time ratio	Energy ratio
		Time	Energy	Time	Energy		
KNN Impl 1	Mem access	4.24ms	0.43mJ	3.04ms	1.2mJ	1.4	0.36
KNN Impl 2		**1.23ms**	**0.003J**	0.93s	84J	0.0013	3E-05
BS Eur	FP arithmetic	**0.0788ns**	**1.67nJ**	0.164ns	14.76nJ	0.48	0.11
BS Asian		**0.0815ns**	**1.96nJ**	0.168ns	15.12nJ	0.45	0.13
Heston Eur		**0.157ns**	**3.33nJ**	0.604ns	48.32nJ	0.26	0.069
Heston barrier		0.158ns	4.917nJ	0.813ns	65.04nJ	0.19	0.076
Bit Sort no HLS Opt	Mem access	152ms	760mJ	16ms	480mJ	9.5	1.58
Bit Sort with HLS Opt		17ms	**272mJ**	**16ms**	480mJ	1.06	0.57

* BS Eur (Black Scholes Model European Option), BS Asian (Black Scholes Model Asian Option), Heston Eur (Heston Model European Option), Heston barrier (Heston Model European Barrier Option), FP (Floating-point), Bit Sort (Bitonic-sort).

been added for reference in tabulated form in Table 4.4. The optimal execution time and energy consumption in each case are also indicated in bold font in Table 4.4. A graphical representation of the performance comparison is presented as well in Fig 4.5, where all the bars below unity show the cases where FPGA wins in terms of execution time and energy-per-computation while the rest of the bars indicate the test cases where the GPU outperforms the FPGA.

Although these results were obtained by other members of the research team, they are significant because they show the effectiveness of our adopted HLS-based FPGA design methodology for a diverse range of applications dominated by different aspects, e.g. by memory accesses or by floating point computations.

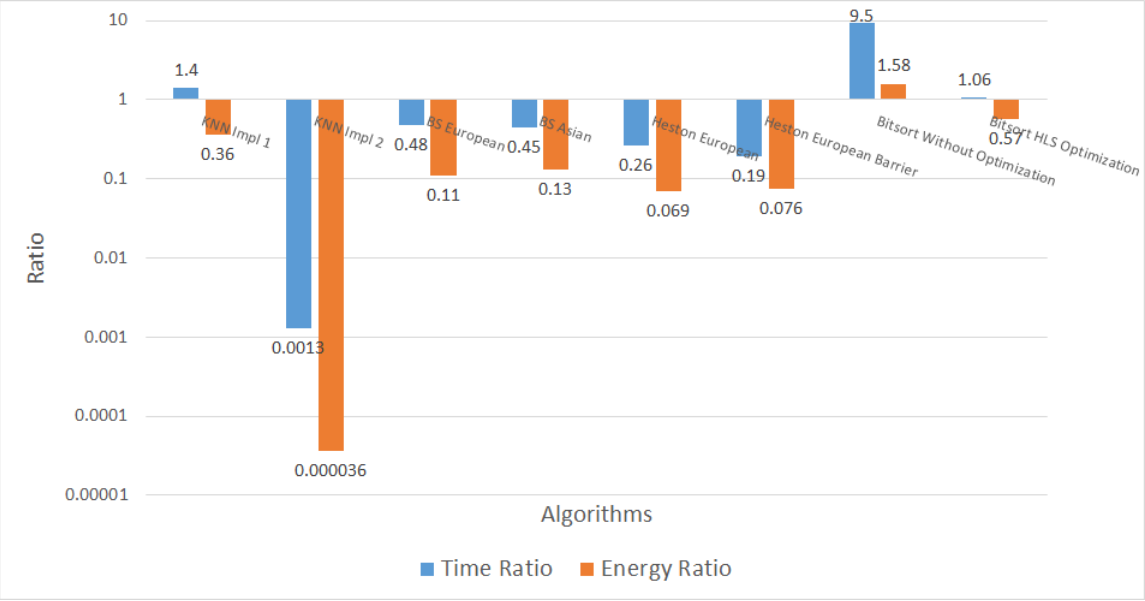

Fig. 4.5 FPGA vs GPU execution time and energy-per-computation ratios for several test case implementations

Chapter 5

Power Management Module

As discussed earlier, power gating can be employed to save static power both for ASIC- and FPGA-based designs by turning off parts of the designs when they are idle. Dynamic power gating of FPGA logic blocks is still a fresh concept and needs further research before it is supported by commercial FPGAs. The power gating for other embedded blocks e.g. BRAMs, phase locked loops (PLLs) and unused I/Os however, is supported by FPGAs from both Xilinx and Altera [48]. The authors in [49] have used a Xilinx Spartan-3 FPGA as a baseline hardware platform and modified it to achieve power gating. Similarly, the authors in [23] have implemented dynamic power gating targeted towards Cyclone-II FPGA on an Altera DE2 board. Power gating in commercial FPGAs however, is not as common as in ASIC designs due to various reasons such as the area overhead corresponding to the power gating logic, lack of knowledge of the application during FPGA architecture design and the resulting performance penalty [23]. Moreover, routing of signals (including the power gating signals) using available routing resources on an FPGA fabric is tricky as some of the available routing resources may get into an idle state during the power OFF process. This chapter hence, presents a system-level power management module that is necessary to provide the power gating logic in order to achieve PSO in an ASIC design.

It is possible to power gate an ASIC implementation starting from its OpenCL representation and using the design flow as presented in Section 4.3.1. The Vivado HLS adds block-level start (*ap_start*) and done (*ap_done*) interfaces to the generated RTL. The *start* interface indicates when a block can start processing the data and

the *done* interface indicates the completion of all the operations by that block [16]. These interfaces can be used to drive the power management block to enforce PSO in ASIC-based accelerators.

The final target here however, is to implement a fully automated low power design methodology starting from the system-level description of the design. This would require the description of a power management block at a level of abstraction higher than the RTL i.e. at the system-level. In order to accomplish this, we use SystemC which is a C++ class library that allows us to create a cycle-accurate model of our software model, hardware architecture and interfaces corresponding to SoC and system-level designs [50]. Hence, SystemC allows us to model hardware using software programming languages. SystemC allows system-level hardware design by enabling support for various features that are pertinent to HW e.g. concurrency support, the notion of time and the support for hardware data types etc. Thus, this activity regarding system-level low-power design is based on SystemC programming framework rather than the OpenCL.

5.1 Overview

A continuous surge in the demand of electronic devices offering multiple sophisticated features necessitates the development of SoCs that can offer high performance along with reasonable power efficiency. For example, the modern smart phones are required to have extensive features such as high data bandwidth (3G, WCDMA, and EDGE), high quality picture, audio and video support (MP3, AAC, JPEG, MPEG, and H.264), Wi-Fi, GPS function, multiple band and network support. Additionally, longer talk time and standby time i.e. longer battery life, are other important selling features in the modern smart phone market [51]. The smart phones are already energy limited as the power is mainly provided by an on-unit battery that can provide only a limited amount of energy. This energy has to be distributed among the various components of the phone and thus each component will have access to only a small portion of the overall supplied energy. Power budgeting hence, is of prime importance in SoCs used in such portable devices.

Additionally, the largest power saving opportunities (with the smallest optimization efforts) are offered by the highest-level of design abstraction i.e. the system-level of design flow as depicted in Fig 1.1. Moreover HLS, due to its benefits e.g. faster

simulation run-time, greater re-use of the design and superior quality of results (QoR), is desired to be used at the front end of the design flow while describing the power intent for the design at the system-level [5]. A brief illustration of such a design flow is depicted is Fig 1.3.

The fact that the naming convention of a design hierarchy including ports, instances and signals is preserved while the design undergoes HLS, is exploited by defining the power intent directly based on the SystemC code. The advantage of doing so, is the relative ease of understanding the code thanks to its higher level of abstraction. Moreover, adding power control logic at system-level makes functional verification extremely simpler through the design-specific SystemC testbench.

It should be kept in mind while specifying the power at the system-level in an HLS-based methodology that the HLS tools would generally delete any signal that does not serve any functionality. The power control signal even if added at the system-level won't serve any functionality, until it is used later-on in the flow (during logic synthesis) to commit the power related commands. This means that the HLS tool would simply delete this signal unless it is directed to preserve it by setting suitable attributes in the HLS tool.

5.2 CMOS Power Optimization

As stated earlier, it is important to optimize both the static and dynamic power in modern SoCs. We use power gating to save static power while clock gating is used to optimize the dynamic power. This section briefly describes the two techniques.

5.2.1 Dynamic Power Optimization

Dynamic power in CMOS mainly occurs due to the toggling of logic states and it depends on the switching activity α, the clock frequency f_c, the supply voltage V_{DD} and the load capacitance C [21], [20]. The dynamic power can mathematically be represented by (5.1).

$$P_{dynamic} = \alpha C f_c V_{DD}^2 \tag{5.1}$$

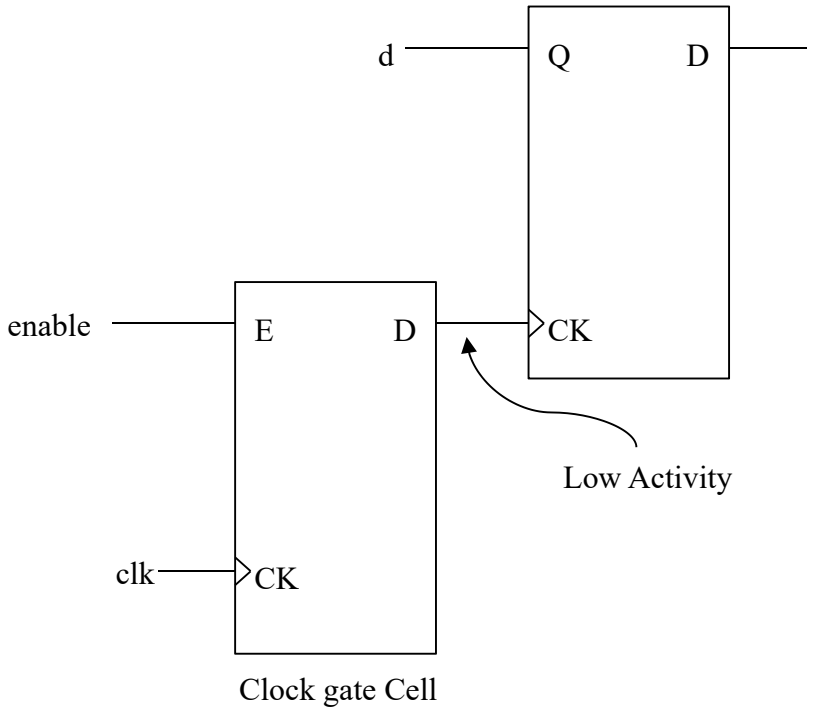

Fig. 5.1 Flip flop with clock gating

Dynamic power can be reduced by reducing any of the above mentioned factors. The design-related parameters however, are α and f_c while the rest of the parameters are technology dependent and can not be controlled by the designer. Clock gating is considered to be one of the most effective options to optimize dynamic power and is thus supported by many commercial synthesis and optimization tools. In simple words, this technique provides a mechanism to shut off the clock to the blocks of the circuit when they are not performing any useful computations [52]. The technique can save considerable amount of power depending upon how often a new value is fed into the circuit. For example, it would be ineffective if a new value is fed every clock cycle but would save 99% of the clock power if a new value is fed once in 100 cycles; by gating 99% of the clock cycles during the time of inactivity. Clock gating is meant to reduce power but it additionally assists the data paths in meeting the timing and results in saving silicon area as it eradicates the requirement of a multiplexer along the data path [53]. A typical clock gating circuit preventing the clocks from reaching the flip-flop while the enable is false is shown in Fig. 5.1.

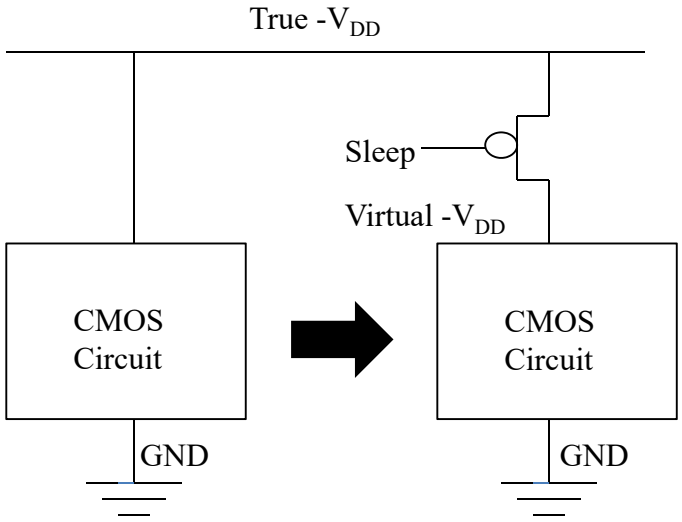

Fig. 5.2 Header switch implementation for power gating a unit

5.2.2 Static Power Optimization

Leakage power can be expressed in terms of the supply voltage V_{DD}, the reverse saturation current I_S, the diode voltage V and the thermal voltage V_T (equal to KT/q). Leakage power is mathematically represented by (5.2).

$$P_{leakage} = I_L V_{DD} = I_S(e^{V/V_T} - 1)V_{DD} \tag{5.2}$$

The design dependent parameters for leakage power optimization are V_{DD} and I_L (hi V_T, low V_T cells). Power gating is used to reduce the leakage power of modules in a design by using a header cell to disconnect V_{DD}, or a footer cell to disconnect the V_{SS} i.e. ground. Both have the same effect of switching off the power to a block that is inactive for considerable amount of time using a control signal [53]. In case of header cells, the supply net connecting the CMOS circuit to the switch is called the virtual V_{DD}. Header cells have been used in this work. A typical header cell is show in the Fig. 5.2. The virtual V_{DD} can be seen as well.

Power gating may be of fine-grain type in which several switches are used to gate each individual cell or it may be of coarse-gating type in which a single switch is used to control a complete block. The former technique offers higher optimization potential at the cost of larger area overhead while the later is better in terms of area overhead but the optimization potential also goes down [53].

5.3 Common Power Format

The power intent for an ASIC design may be described using CPF standard from the low power coalition at Si2 [22]. Unified power format (UPF) can also be used to accomplish this. UPF is an IEEE standard developed by Accellera and it is essentially equivalent to CPF [54]. CPF has been used in this work to describe power intent without the loss of generality. CPF has the ability to provide an integrated power intent that can be used at every stage of the design flow and hence automation of the flow becomes easier. Without CPF, the power intent has to be written at each stage of the design flow e.g. RTL, synthesis, formal verification etc. Even after so much of manual work, the consistency in the flows is hard to guarantee and the flows cannot be automated. CPF is a tool command language (TCL) based language operating on specification objects and design objects. The design objects can be a module, instance, net, a pin or a port appearing in the RTL. The power intent is represented in terms of *power logic*, *power domains* and *power modes* [53]. Low power intent represented by CPF is supported by several advanced low power SoC design tools. It shall be noted that the RTL files do not get modified with the power intent, rather the power intent is essentially separate from the design intent i.e. RTL files, and hence captured separately using the CPF.

5.4 Power-Aware System Model

The system model consists of a SystemC implementation of the design under test (DUT). The activity profile of the DUT is analysed through its simulation via a dedicated SystemC testbench while observing its time-line trace. The activity profile is required mainly to locate the idle periods in a design in order to shut it down during significant duration of inactivity to save leakage power. Dynamic power optimization is obtained by utilizing either the coarse-grained clock gating or fine-grained clock gating. Fine-grained clock gating is implemented directly by the HLS tool and is less effective in general. Coarse-grained clock gating on the other hand has been used in this work, which is implemented by explicitly adding a CG block (written in SystemC) to the system-level description of the design.

5.4.1 Low Power Design flow

SystemC is used to model the entire design which is then provided as an input to the HLS tool to automatically generate the RTL. The main steps involved in HLS are the target platform specification and making micro-architectural decisions to satisfy various area, time and power constraints. This is followed by specifying the scheduling constraints to finally obtain the RTL description of the design. The switching information is obtained by simulating the generated RTL through the same SystemC testbench, used for simulating the original system-level design, by making use of the SystemC wrapper which is generated automatically by the HLS tool.

The switching activity includes the toggle rates and the static probabilities for all the pins/nets of the module. Toggle rate specifies how often the pin or net switches between logic-1 and logic-0 states during the specified duration. The static probability on the other hand represents the probability of a net or a pin being high i.e. in logic-1 state. Toggle Count Format (TCF) is a cadence standard used to represent the switching information of a design. It can be obtained either through encounter RTL compiler by using the $write\_tcf$ command or by simulating the design and dumping the switching information by using the *dumptcf* command with cadence NCSim [55], [56]. The TCF file in this work has been created by using the later approach.

Power-aware logic synthesis is then done beginning with reading the target libraries which in a CPF-based methodology are read from within the CPF file. Clock gating is enabled by using the appropriate attribute to allow the utilization of coarse-grained clock gating logic. This is followed by reading and elaborating the RTL design. Technology mapping of the cells to be used as clock-gated integrated cells (CGIC) is accomplished thereafter. This is followed by reading the power intent that is specified in a separate CPF file along with setting the timing constraints and synthesizing the design. Annotation of the switching activities is then performed followed by application of the power intent. Power structure verification is performed in order to validate the correct insertion of low power cells in accordance to the rules specified in the CPF file. Incremental optimization is done and eventually the gate-level netlist is obtained which is then validated for logic equivalence against the input RTL [5]. The complete design flow is depicted in Fig. 5.3.

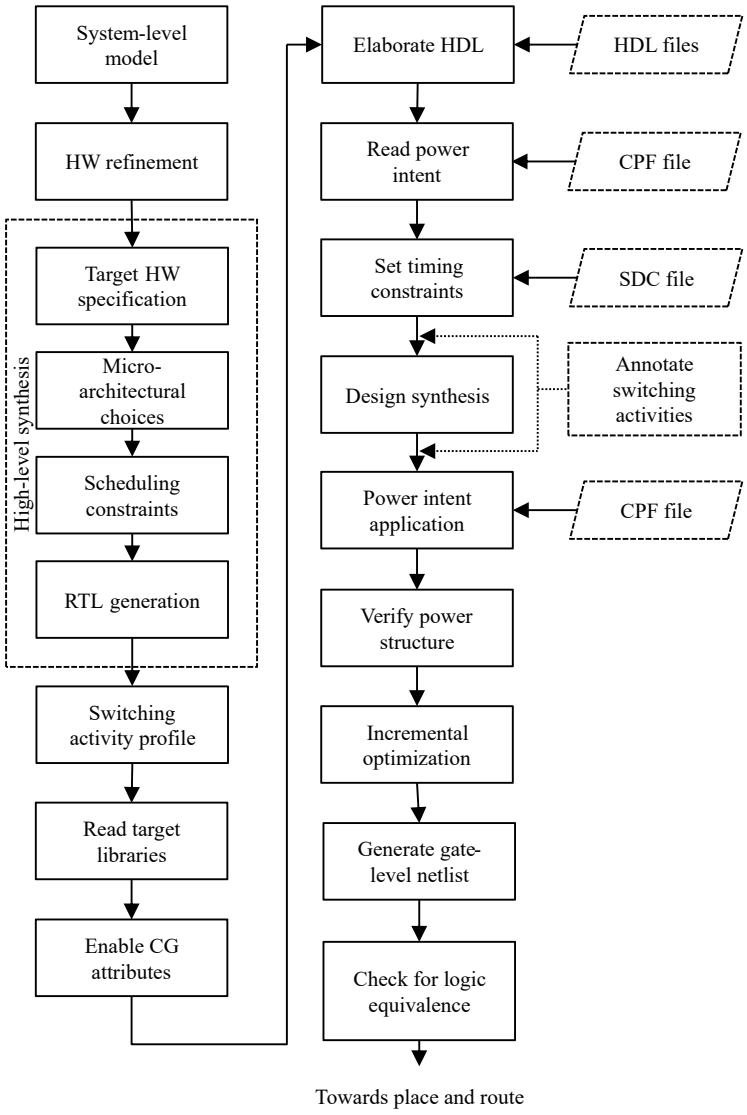

Fig. 5.3 Complete low power design flow

5.4.2 Power Management Block

In an HLS-based low power design flow, the system model, to be used as an input to the HLS tool, needs to be adapted by instantiating in it, a power management block (PMB). This block is responsible to turn the power ON and OFF to the modules in the switchable domain. This module is specified in SystemC and would obviously reside in the always-on power domain. The complete power-aware system model depicting the power management block is shown in Fig. 5.4.

A critical issue to consider while applying PSO is to prevent the propagation of the floating states from the power gated domains to the active domains. It may also be necessary to retain the states of some of the flip-flops before shutting down a portion of the design. The former is done by using isolation cells (ISO) while state retention cells (RET) are used for state retention of flip-flops. The isolation cells are usually placed at the output of the switchable domain as shown in Fig. 5.4 which also shows the state retention cells. Furthermore, header power switches are inserted during physical implementation which enables the cutting-off of supply voltage to the switchable domain.

Moreover, the PMB is required to generate the power control signals in the correct sequence to power gate the desired module. During the power-down process, the isolation must happen before state retention followed by the power shut-off while the reverse sequence shall be followed during power-up process. Typical power-up and power-down sequences are shown in Fig. 5.5. The *enable* signal in Fig. 5.5 indicates the signal that would trigger the power ON/OFF sequences. This signal is identified manually from the design during activity profiling and it indicates the inactivity cycles in a design hence allowing us to activate the power OFF sequence during those cycles. The isolation enable *iso_enable*, state retention enable *ret_enable* and power shut-off enable *pso_enable* signals, as produced by the PMB can also be seen in Fig. 5.5. A purely illustrative SystemC pseudocode of the PMB is given in Algorithm 1.

To make sure that no data is lost during the power-up sequence, a separate first-in first-out (FIFO) SystemC module is also added to the overall system-level design. This is meant to operate as a buffer to save any computational data that might be coming in during the power-on sequence while the power optimized module is still asleep. Moreover, clock gating is achieved by using the SystemC clock gating

Algorithm 1: PMB Algorithm

Input: PSO enable signal *enable*;
Output: Power Control signals *pso_enable*, *iso_enable*, *ret_enable* and clock gating enable signal *CG*;

```
1  Initialization;
2  wait(); //wait for one clock cycle
3  PMB Logic
4  while true do
5      if (enable == 1) then
6          iso_enable = ON;
7          wait();
8          ret_enable = ON;
9          wait();
10         pso_enable = ON;
11         CG = ON;
12     else
13         CG = OFF;
14         pso_enable = OFF;
15         wait();
16         ret_enable = OFF;
17         wait();
18         iso_enable = OFF;
19     end
20 end
```

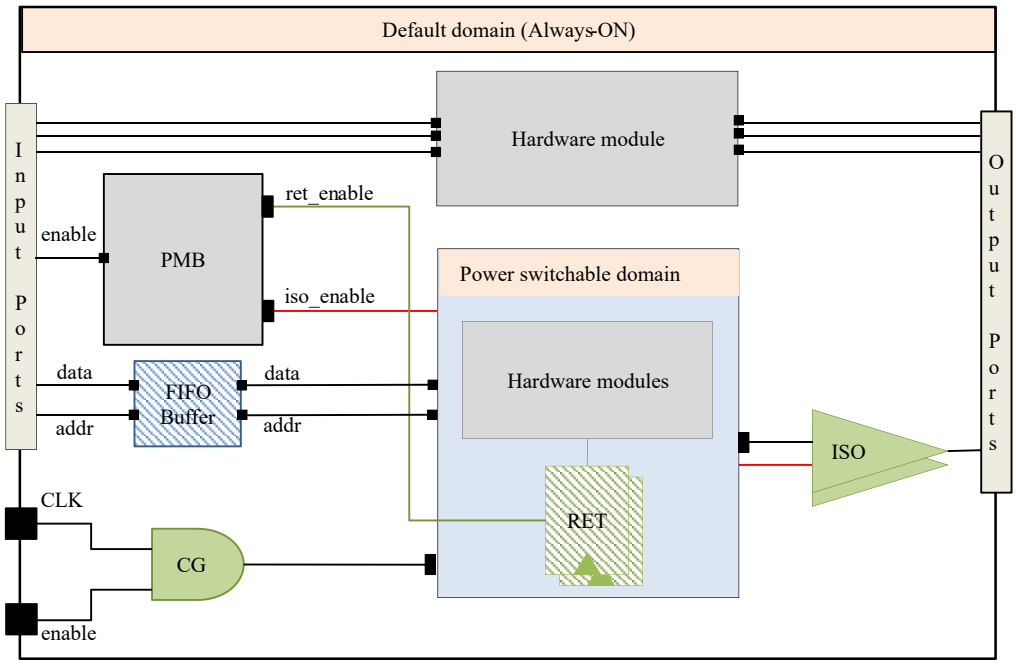

Fig. 5.4 System with power optimization features

module provided by [57]. The CG module used to introduce the clock gating logic in the gate-level netlist can be seen in the Fig. 5.4.

5.5 Integrated Clock gating and Power gating

It is possible to integrate CG and PG to get maximum power reduction by using the same signal to enforce both the techniques [21]. In our case, we have used the same signal to drive both the power gating as well as the clock gating logic, hence, gating the clocks for the instances when the design is powered down [5]. The idea is depicted in Fig. 5.6.

5.6 System-level Power Management Module Validation

An inverse discrete cosine transform (IDCT) being used in JPEG decoder, has been used as a design test case to validate the system-level description of the PMB. This

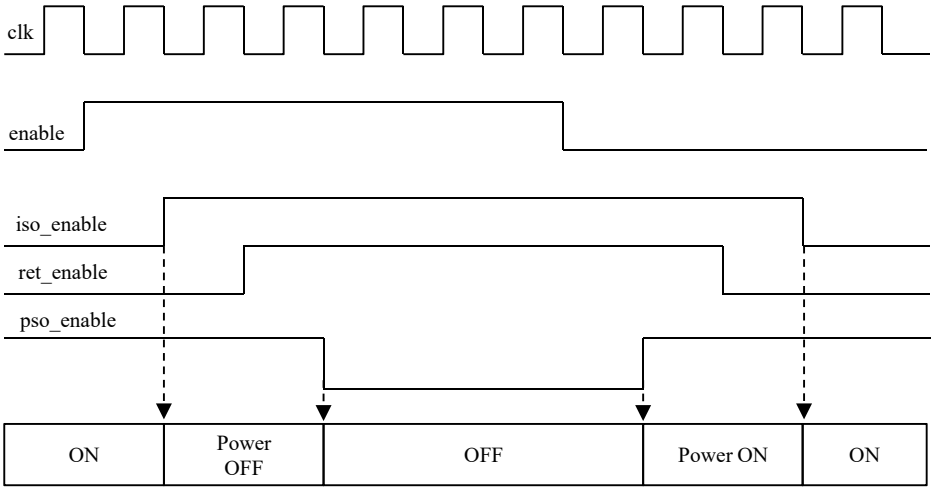

Fig. 5.5 Power Up/Down Sequence

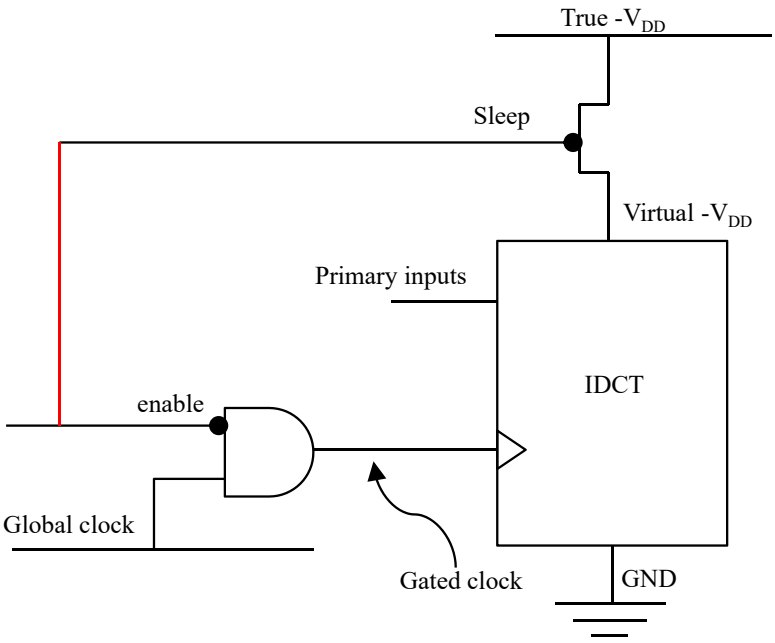

Fig. 5.6 Illustration of CG and PG integration

test case is briefly explained here. This section also presents the experimental setup used during this activity along with the experimental results validating our system-level PMB design methodology.

5.6.1 Inverse Discrete Cosine Transform

IDCT is a widely used algorithm for data compression and is used in several video and image processing standards such as MPEG, JPEG and CCITT H.261 etc [58]. It also finds an application in the hardware accelerators being used in heterogeneous SoCs for smart phones [5]. A typical power-aware JPEG decoder utilizing IDCT is shown in Fig. 5.7. The various modules used to optimize power can be seen as well. Besides IDCT, a JPEG decoder performs several operations like variable length decoding (VLD), zigzag scanning (ZZ), de-quantization (DQ), color conversion and image re-ordering; which are shown in Fig. 5.7.

In our case, we have taken a synthesizable implementation of JPEG IDCT decoder consisting of concurrent processes communicating between themselves at the level of transactions. A 2D-IDCT is implemented by first executing a 1D-IDCT over each column of the data matrix followed by another 1D-IDCT over each row of the matrix. IDCT is the main contributor to the net complexity of a JPEG decoder [59] and hence it is a strong contender to be considered for power gating.

5.6.2 Experimental Setup

A SystemC description of the JPEG decoder IDCT block is taken as a DUT. The modules necessary to obtain power optimization are added and the functionality is verified. This is followed by performing HLS to obtain the equivalent RTL design. RTL simulation is then used to verify functionality by using the SystemC wrapper created by the HLS tool. The power intent is specified and is later committed during logic synthesis to get power aware gate-level netlist. Various tools are used during the logic synthesis flow for logic equivalence checking as well as power structure verification. The complete design flow is implemented using tools from Cadence. These mainly include Cadence C-to-Silicon compiler for HLS and RTL compiler for implementing the backend flow. Furthermore, Conformal® logic equivalence checking tool [60] and Conformal® low power tool [61], both again from Cadence,

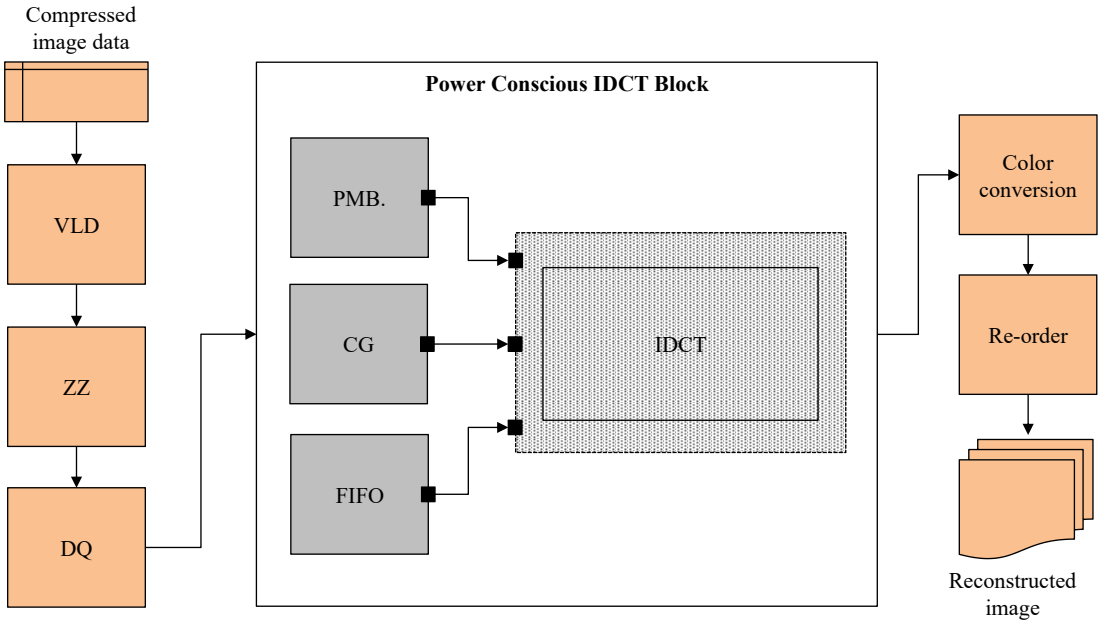

Fig. 5.7 JPEG decoder using IDCT module

are used for RTL/gate-level netlist equivalence checking and low-power design verification respectively. The design has been implemented using the Nangate 45nm OpenCell library [62] with support for low power designs.

5.6.3 Results

Three contrasting implementations of the IDCT design have been considered to validate our proposed system-level design methodology. They include the un-optimized design, design with CG alone and design with both CG and PG. Moreover, various toggle rates on the power control signals have been considered to evaluate the impact of switching on the overall power consumption. The power and area reports post-synthesis have been obtained for all the test cases and compared. The power and area reports for all the test cases are given in Table 5.1 while the toggle rates corresponding to the JPEG usage are given in Table 5.2.

It is clear from Table 5.1 that CG results in reduction of dynamic power by around 10X due to reduction in toggling of the design clock signal resulting from CG. The PG on the other hand reduces leakage power by more than 50%. This is achieved by shutting down the IDCT module in cases where no useful computation is taking place. Table 5.1 also shows the impact on power due to more toggling of

Table 5.1 Power wrt area performance for IDCT test cases

Test Cases	$\mathbf{P_{static}}$ (μW)	$\mathbf{P_{dynamic}}$ (μW)	**Area** (μm^2)
No Pwr Opt	572	12570	43919
CG only	503	1085	40299
CG and PSO	240	655	44124
CG and PSO (4X toggle)	240	680	44124
CG and PSO (8X toggle)	240	726	44124
CG and PSO (32X toggle)	240	849	44124

Table 5.2 Toggle rates for JPEG usage

Power Control pins	**Toggle Rate (tranistions/sec)**
PSO enable pin	33333
ISO enable pin	10^6
RET enable pin	10^6

the power control signals. The baseline toggle rates as presented in Table 5.2 have been obtained by simulating the RTL in JPEG decoder usage scenario and dumping the signal switching values in the form of a TCF file. They are then incremented by 4X, 8X and 32X in order to see the impact of more switching of the IDCT module on the net power consumption. As evident from Table 5.1, more toggling results in more dynamic power consumption as the IDCT module is switched more frequently. The static power on the other hands remains the same due to no change in the static probability of the power control signals. This analysis is important mainly to estimate the amount of switching beyond which any power saving resulting from optimization efforts would become fruitless due to the additional dynamic power consumption resulting from frequent switching of the HW module in the switchable domain. The power consumption for the various test cases is shown graphically in Fig. 5.8.

Table 5.1 also shows that CG results in some area saving as it eliminates the requirement of extra multiplexers on the data path thereby replacing them by clock gating logic [53]. The power gating on the other hand causes an increase in the overall chip area due to the additional low power cells added to the design. This is

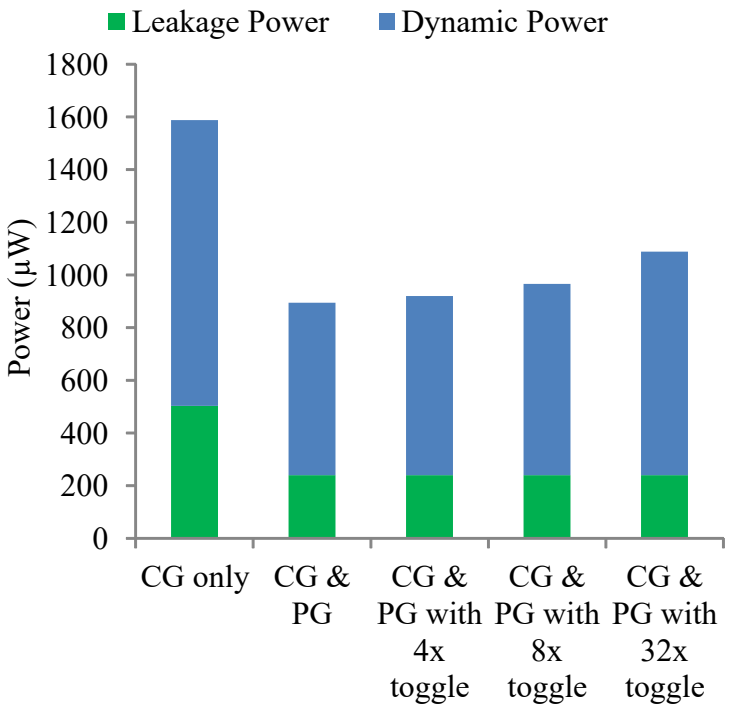

Fig. 5.8 Power Consumption for IDCT test cases

evident from Table 5.1 as well. The logarithmic power versus area diagram for the different implementations is shown in Fig. 5.9.

Table 5.1, Fig. 5.8 and Fig. 5.9 show that our proposed methodology (concerning system-level power management module description) manages to perform as expected. Leakage power saving is obtained by utilizing power gating while dynamic power optimization is obtained by using clock gating. Furthermore, power gating gives a slight area overhead as a trade-off due to the additional low power cells, while clock gating helps in saving area by removing additional multiplexers and replacing them by clock gating logic. The work presented in this chapter eventually contributed to our final goal i.e. to obtain a fully automated methodology for power intent specification using CPF right from the system-level description of the design. This methodology is presented in the next chapter.

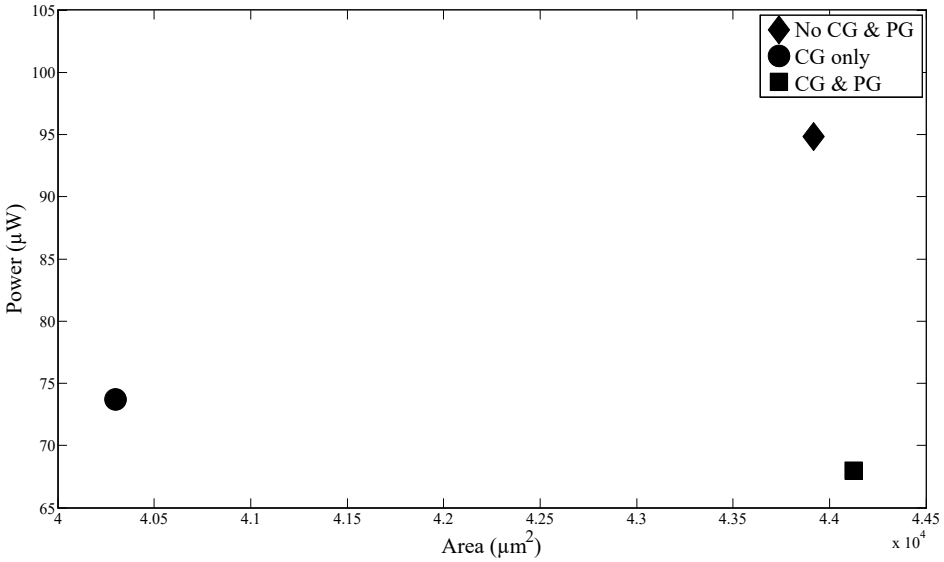

Fig. 5.9 Area versus Power diagram for IDCT test cases

Chapter 6

High-Level Synthesis Based Automated Low Power Methodology

The methodology presented in chapter 5 dealt with describing the power management module at the system-level rather than at the RTL as done in a traditional low power digital design flow. The CPF file used to capture the power intent of a design so as to power gate it, was still written manually. This chapter would complement the work presented in chapter 5 by describing a methodology as well as a tool that we developed to automatically generate the CPF file for a specific system-level design.

Specifying power intent at the system-level significantly reduces the design effort as system-level model is significantly easier to understand as compared to the RTL (which is normally used for defining power intent). Moreover, it would result in a considerable reduction in the simulation and analysis time as compared to that involved while specifying the power intent at lower levels of the design flow [63]. Finally as mentioned before, it would also make the functional verification considerably easier.

6.1 Related Work

A summary of relevant work previously done regarding power reduction strategies in SoCs is presented here. It shall be noted that most of the work done previously targets power optimization using CPF or UPF applied at the RTL description of the

design. The idea regarding power intent description at the system-level has been explored rather sparsely. One such example is the work presented in [64], where the authors presented several techniques like sequential clock gating, power gating, dynamic voltage scaling etc to save power at the RTL level. CPF and UPF were identified as standards to be considered for unified power intent specification for all stages of the design flow. The idea of specifying power intent at the system-level was emphasized upon, but not considered in their work.

The authors in [65] presented a detailed survey of energy measurement and estimation along with description of some common techniques to optimize power above the RTL level. The authors used a JPEG decoder design to validate their work. They emphasized on using system-level power optimization solutions early in the design flow especially for data-flow dominated blocks and their affiliated memory blocks.

A detailed research study regarding HLS-based power optimization strategies proposed over the last decade was presented in [17]. The authors strongly recommended raising the level of abstraction beyond RTL while using HLS tools in order to obtain faster power optimization in modern digital designs. The authors were of the view that this is the only option for modern day designers to meet the strict power requirements accompanying modern hardware design. Several reasons have been identified in this work to motivate this point. A multitude of low-level information ranging from functional, structural, temporal to spatial details would need to be considered together if the optimization was to be done considering manually written RTL. This would obviously make the optimization process very difficult especially considering the strict timing constraints accompanying modern digital designs. Additionally, the authors argued that power scaling necessitates the evaluation and optimization of system architecture as soon as possible in the design flow i.e. before the RTL.

A detailed study of multiple energy efficient system-level design methods has also been presented in [66]. The main hardware components consuming the most energy as identified in this work are the computation, communication and storage units and it is emphasized to consider system-level energy-efficient design techniques in all of the three main hardware components.

Hence, it can be safely said that there exists a general consensus in the design community regarding the importance of system-level power optimization techniques.

Acknowledging this fact, many commercial tools have been developed in the recent past e.g. Vista Architect from Mentor Graphics as well as Chip Vision's Orinoco and PowerOpt^{TM}. A couple of working groups were constituted by IEEE in 2014 to standardize system-level power modeling for SoC devices [67]. The proposition of a fully automated design methodology consisting of power specification at the system-level and its application in combination with HLS thus is a crucial but rather rarely explored domain and is thus a theme of this research activity.

6.2 Methodology Description

This section describes our proposed LP-HLS flow in detail. It consists of an HLS-based flow similar to the one presented in Fig. 1.3. A short overview of the methodology is presented here. The power intent in our proposed flow is derived by extracting the design related information from the system-level design description. This information is combined with the PSO specification rules as well as the technology related information to generate the CPF file. The CPF rules describing low-power design are then applied on the design during the later stages of the backend design flow.

Besides the HLS flow, the LP-HLS methodology also consists of a power intent generator tool (producing the CPF automatically) and a final fully integrated backend design flow, both of which are explained here.

6.2.1 Power Intent Generation Tool

The developed CPF automation tool uses #pragma directives (in the SystemC design file) in combination with specifications file written by the designer, to extract the power intent for a specific design. The basic structure of our automated CPF generation tool is shown in Fig. 6.1 while the various steps involved in CPF generation are depicted in the form of a flow chart in Fig. 6.2.

The technology related information e.g. the technology libraries (usually best, worst and nominal cases), is provided by the (*Tech. spec. file*) as depicted in Fig. 6.1. This file contains the relevant information expressed in a CPF compliant syntax. The information regarding the power nets (created during physical implementation) and

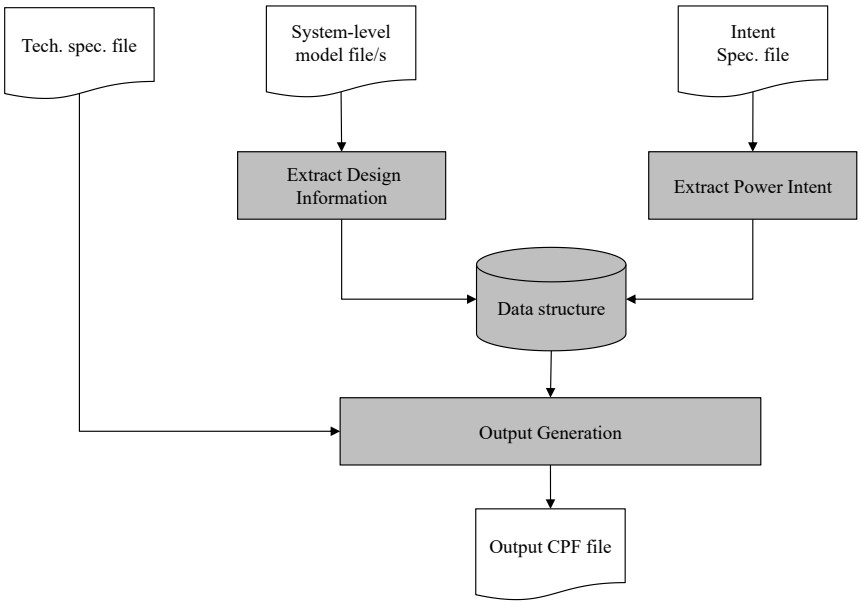

Fig. 6.1 Illustration of the CPF generation tool

their operating conditions e.g. their voltage levels, is also provided by this file. This information is utilized later by the different power modes specified.

The rules required to implement PSO are provided using #pragma directives in the (*Intent Spec. file*) as shown in Fig. 6.1. These rules are dependent on the design specific parameters and include e.g. description of power rules such as power switch, isolation and state retention rules.

Additionally, the CPF generation tool needs as an input, the files describing the system-level model. It would extract the design related information from the model files e.g. the modules belonging to the switchable and always ON domains, the various signals needed to achieve isolation, state retention and PSO etc. This is accomplished by again using #pragma directives before the name of the respective instances. The tool looks for those unique pragmas thereby, creating token strings for the succeeding instances and storing the information for later processing. The power domain names may be specified by the designer or alternately, they may be generated by the parser. Once the custom data structure gathers all the important information from the input files, the power rules compliant with CPF standard are generated corresponding to each pragma directive, while utilizing the information specified in the configuration files.

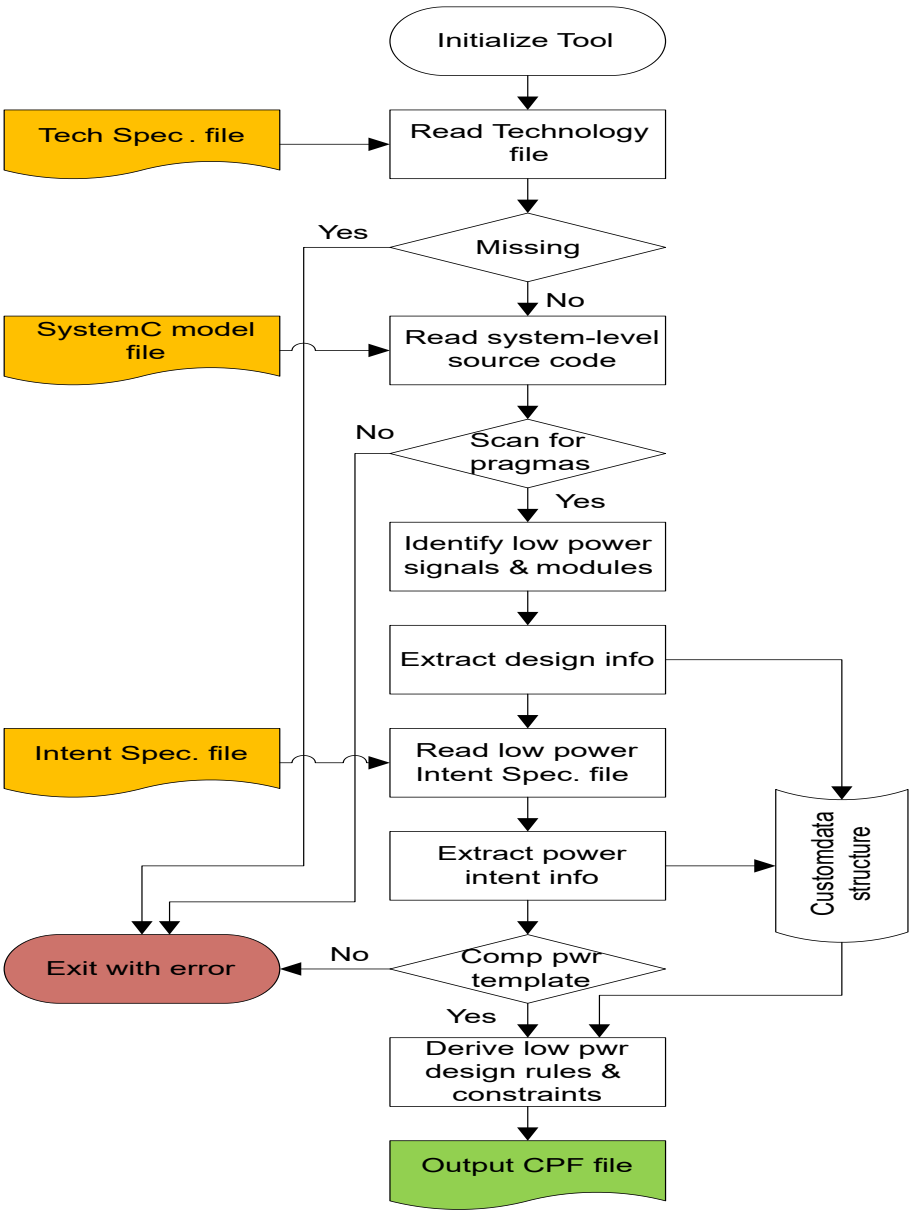

Fig. 6.2 CPF generation Flow

6.2.2 Complete Design Flow

The complete low power design flow beginning with a system-level design and ending with a power-optimized gate-level implementation, while utilizing our proposed LP-HLS methodology combined with the standard RTL-to-gates flow is depicted in Fig. 6.3.

Most of the operational flow is similar to the standard HLS-based low-power design flow as depicted in Fig. 5.3, however, few blocks representing a simplified view of the steps involved in the automatic power intent generation have been added here. The authors in [68] emphasize on the use of a "design improvement loop" at each level of abstraction to assist in finding an optimal solution when the final target is a low-power implementation. Such a loop typically uses a power analyzer (shown shaded in Fig. 6.3) to rank different design, synthesis and optimization alternatives thereby choosing the one that is potentially more effective from the point of view of power consumption. The flip side of this approach however, is that it requires the availability of power estimators in addition to synthesis and optimization tools providing reliable results at different abstraction levels.

The necessary steps involved in the design flow can be briefly summarized here again for completeness. The complete system-level design (with PMB instantiated) goes through the various standard HLS steps e.g. target technology specification, making micro-architecture choices and specifying scheduling constraints, thereby giving the corresponding RTL implementation. This HLS process is accompanied in parallel by our tool that generates the CPF file using the information extracted from the system-level description and the configuration files, following the methodology specified before. This is followed by power-aware logic synthesis which involves several steps e.g. reading and elaborating the RTL design, enabling the application of coarse-grained CG logic, reading the power intent, setting timing constraints followed by design synthesis. Moreover, switching activities are annotated for accurate power analysis and finally power structure verification is performed to verify the correctness of power intent applied along with the logic equivalence check for the gate-level netlist against the input RTL.

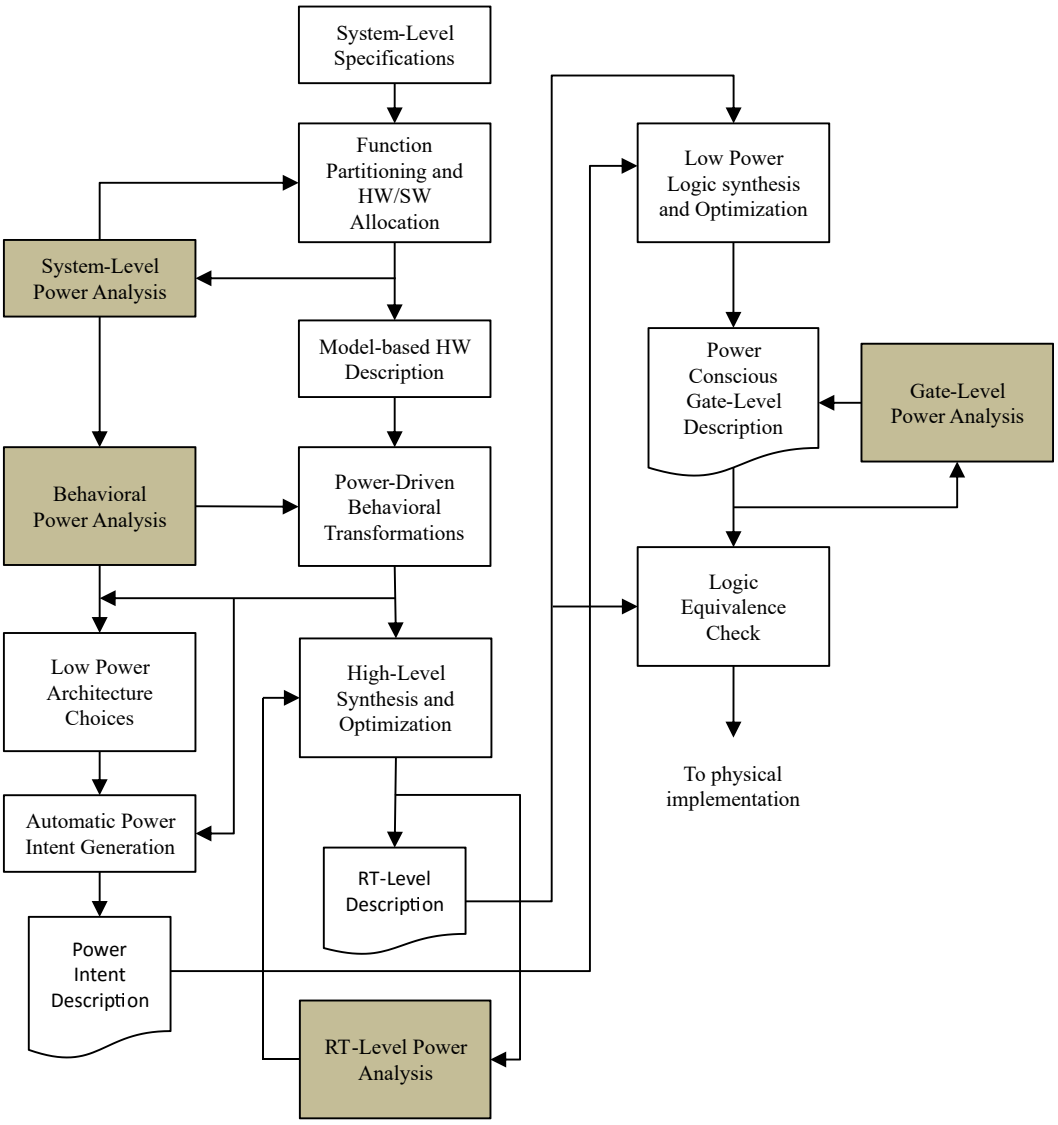

Fig. 6.3 Complete Flow of Low Power High-level Synthesis methodology

Chapter 7

Design Test Cases for Methodology Validation

This chapter presents the test cases which we considered to validate our LP-HLS methodology. The testbench structure used for functional verification of all the design test cases is discussed as well. The test cases range from low to medium level in terms of complexity. The very first test case is a very simple 32 bit hierarchical ripple-carry adder (RCA), wherein the block processing the 16 most significant bits is selected for power optimization thus enabling a power-efficient processing of 16 bit values. This is followed by a comparatively more complex ALU processor design, where we select the multiplication and division blocks for power optimization when they are not in use. Our third case is even more complex JPEG IDCT decoder design where we apply PG and CG to the IDCT block. This has been presented earlier in Section 5.6.1.

7.1 Structure of the testbench for design functional verification

A typical testbench structure used for the functional verification of the design test cases is shown in Fig. 7.1. The testbench structure consists of a stimulus generator used to provide control and data input signals to the DUT. The results are then monitored for validity, as presented in Fig. 7.1. The general structure of DUT

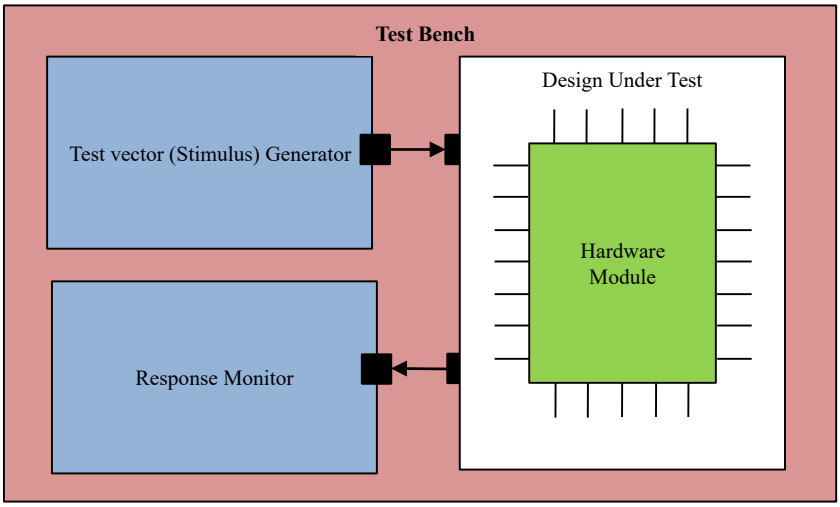

Fig. 7.1 General testbench structure for design validation

consists of a system model consisting of a description of all the modules required for achieving power optimization i.e. the design test case along with other supplementary modules, similar to the one depicted in Fig. 5.4.

We consider pseudo-random number generators (PRNGs) to provide both the input data streams as well as random control logic for power gating the RCA and ALU test cases. Obviously, we could have used probabilistic models to better reflect the behavior of these modules under real life conditions. This is however, a sufficiently explored topic and was beyond the scope of our intended research [69], [70]. It must be noted that we used synthetic switching activity profile in the case of RCA and ALU while the switching activity in case of IDCT design was extracted by simulating it under the real-life JPEG IDCT decoder usage scenario.

7.2 Design Test Cases

The three design test cases used for validating our LP-HLS methodology are presented here.

7.2.1 Ripple Carry Adder

A hierarchical model of the RCA is described in SystemC in order to power gate it. It is composed of two identical 16-bit RCAs indicated by $MSB\_RCA_{0\text{-}15}$ and $LSB\_RCA_{16\text{-}31}$ representing the MSB and LSB processing respectively as illustrated in Fig. 7.2. We chose to put MSB_RCA in the power switchable domain, hence it consists of some additional ports to achieve PSO as can be seen in Fig. 7.2. A $P_{shut\text{-}off}$ signal is used as a trigger for the PSO operation and the same is provided to the output multiplexers as well (as clear from the figure) in order to select between the valid output signals. As mentioned before, a synthetic switching activity profile is considered to estimate the power in the case of RCA. The RCA is considered to be in the ON state starting from 30% of the time and various sweeps are given to the ON state time up until 90% of the overall operational time.

7.2.2 Arithmetic and Logical Unit

The second test case is an ALU processor as depicted in Fig. 7.3 performing a variety of arithmetic and logical operations e.g. addition, subtraction, bitwise AND/OR etc. We chose to apply PSO on the MULTIPLY and DIVIDE blocks as they consume the most hardware resources and hence are the most power consuming blocks [71]. They are hence, kept in the switchable domain shown shaded in Fig. 7.3. A control signal SEL is provided to the encoder as a stimulus to choose between unique opcodes assigned to each individual operation. The same SEL signal is used to turn the power switchable domains ON and OFF as well. The ON period for the power switchable domains in the case of ALU varies between approximately 10% of the time and 90% of the time.

7.2.3 JPEG IDCT decoder

The third test case is a JPEG IDCT decoder design as was presented in Section 5.6.1. We power gate the IDCT module which is the most complex and power consuming part of the overall JPEG decoder and hence it is a natural choice to be considered for PSO optimization.

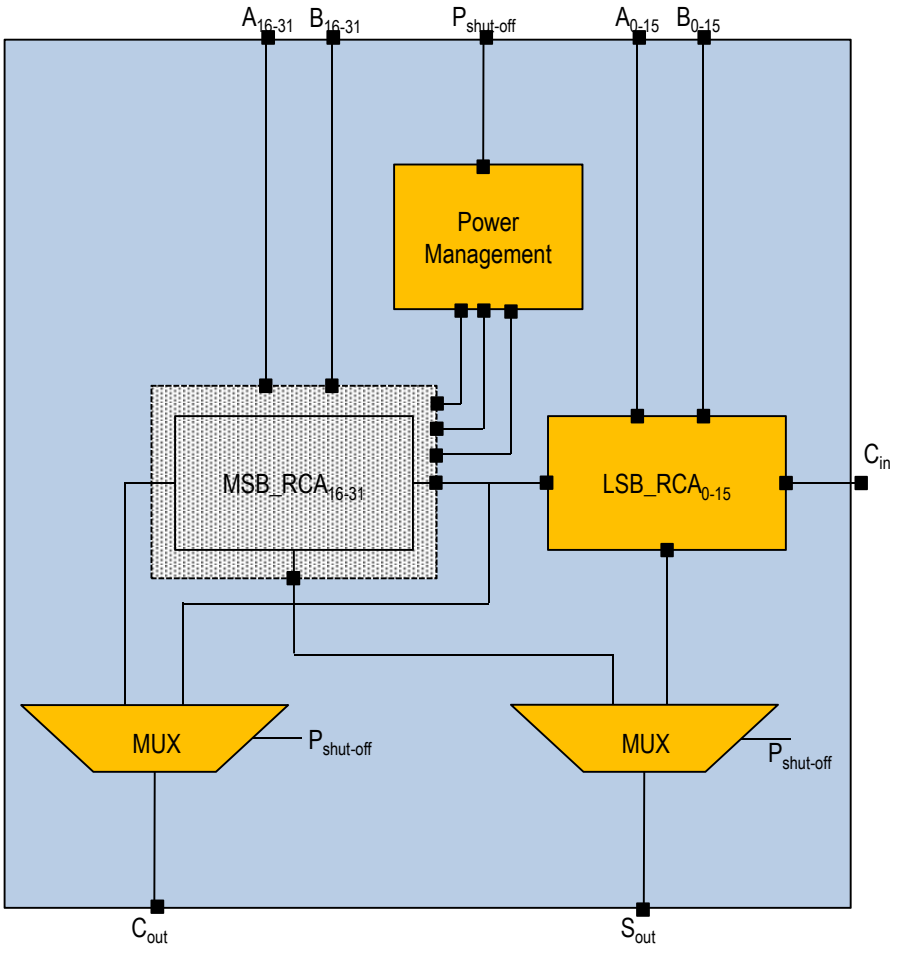

Fig. 7.2 Power Aware 32-bit Ripple Carry Adder

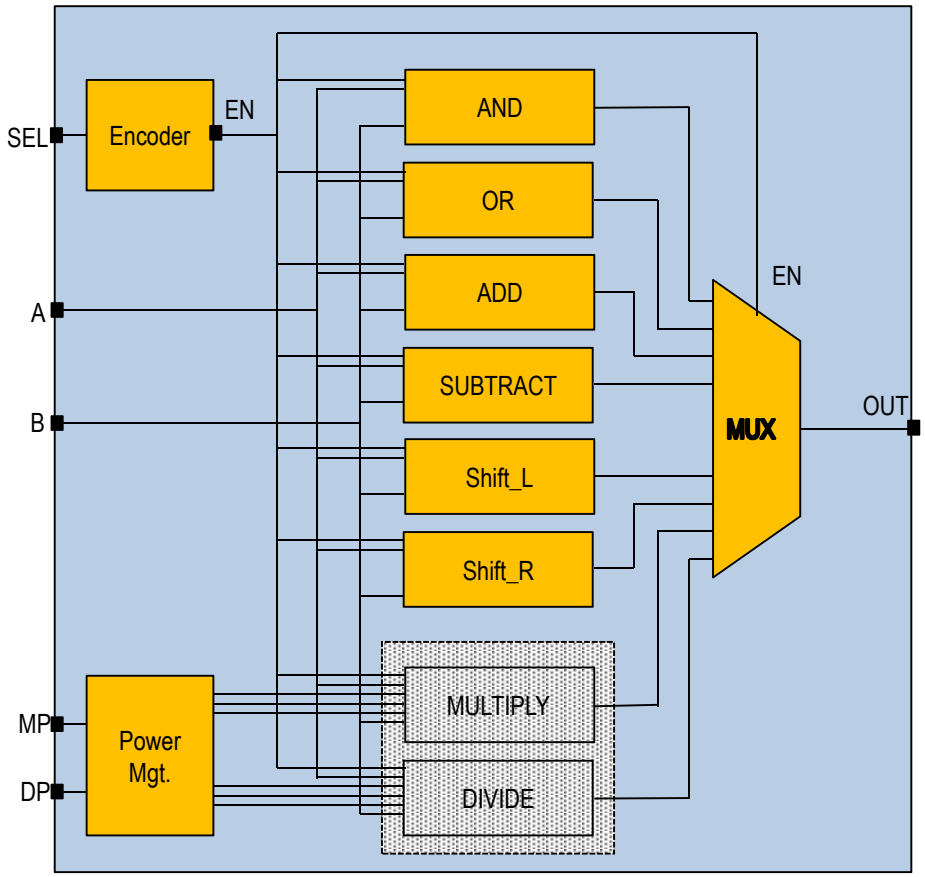

Fig. 7.3 Power Aware Arithmetic and Logical Unit Processor

Besides being the most complex case among our selected test cases, the IDCT module consumes a considerable amount of the RAM resources, which makes it even more desirable to validate our power optimization methodology. This is because the memories in modern SoCs may be as much or sometimes even more hungry for power when compared with the data path [17]. Their dynamic power consumption may amount to around one-third of the overall SoC power, while the remaining two-thirds coming from the data path and the clock-trees [72]. The system model for a power-aware JPEG IDCT decoder is depicted in Fig. 5.7. Since, the switching activity in this case comes from the real-time simulation of a JPEG-IDCT decoder, this enables us to validate our methodology in a more realistic scenario.

7.3 Results

The general experimental setup utilizes a SystemC description of the design under test i.e. the RCA, ALU processor and the IDCT design. Tool chain from Cadence, as mentioned in Section 5.6.2, is used for high-level synthesis, backend implementation as well as for logic equivalence check and power structure verification. Various implementations of the algorithms have been considered for testing our methodology i.e. unoptimized designs, optimized designs (with both PG and CG) and optimized designs with various sweeps considered for the switching activities on the power control signals. The main difference with the flow described before in Section 5.4.1 is that our proposed LP-HLS methodology has been used here to automatically generate the CPF file for power intent description along with generating RTL implementation for each design test case. In comparison, the CPF file previously was written manually while only the PMB was described in SystemC and synthesized as an instantiation along with our design under test in a top-level module.

It should be noted that without utilizing our proposed methodology, the power intent would need to be described manually at a lower level of abstraction e.g. RTL (generated automatically in an HLS based flow). It normally consists of thousands of lines of code, which is very difficult to understand and needs a considerable design effort (for power intent description) as we experienced during the activity presented in [5] and described before in Chapter 5. A huge amount of manual effort was put into that activity, first to understand and then derive design related information from an automatically generated RTL, in order to achieve a low power implementation.

This was followed by manually writing a CPF file to achieve the intended low power implementation. In contrast, our LP-HLS methodology and the accompanying tool can automate this process by extracting the relevant information from the design and generating the CPF file automatically. This would greatly reduce the effort both in terms of design time as well as design understanding. The power analysis is based on the power models incorporated in the standard cell libraries and it depends on the expected state of the signal at the boundary of the standard cells and their rate of toggling [53].

In order to validate our methodology, we took a range of values for the static probabilities as well as the toggle rates of the power control pins corresponding to the PMB. The static probability corresponds to the overall workload of an operation while the toggle rate, as mentioned before, determines the number of signal transitions in a unit time. The static probability affects the static power consumption while the toggle rate influences the dynamic power consumption of the design.

As mentioned before, the switching activities corresponding to IDCT are extracted by performing the RTL simulation in real-time JPEG decoder usage scenario while sweeps are applied only on the toggle rates to perform power analysis. In the rest of the cases, synthetic input data is used and static probability i.e. utilization for the switchable domain is specified as a factor of the utilization of the rest of the design. The toggle rates for the power control pins on the other hand are specified based on the utilization factor.

The power analysis for the RCA is represented in tabular form in Table. 7.1 and depicted graphically in Fig. 7.4. The various test cases are determined based on the operation workload (usage) of the MSB_RCA module. Our first test case is an example with no power optimization and a 50% usage of the MSB_RCA. The rest of the cases include both static and dynamic power optimizations through PG and CG respectively, while using MSB_RCA usage of 90%, 70%, 50% and 30% respectively. It is clear from the table that the power optimizations result in halving of the static power and also result in dynamic power saving by around 2-3X even with 90% MSB_RCA usage as compared to 50% usage in the unoptimized case. The MSB_RCA takes around 31% of the overall chip area of the RCA test case hence making it an acceptable choice to be considered for power optimization. Furthermore, the static and dynamic power consumption of the optimized design reduces as the workload corresponding to MSB_RCA is reduced due a reduction in the overall

static probability and toggling of the power control pins and hence a corresponding reduction in the usage and toggling of modules in the switchable domain.

The second test case is an ALU processor in which we consider power gating (and clock gating) the division (DIV) and multiplication (MULT) blocks. These modules combined consist of a 48% of the overall chip area and hence form natural choices to be considered for design power optimization. They are assigned to two separate power switchable domains and assigned different usage percentages. The usage percentages in case of power optimized designs range from 1% (for DIV) to 10% (for MULT) representing integer workload and sweeps are given from 10%-30% (for DIV) and from 40%-60% (for MULT) corresponding to DSP operations. An unoptimized ALU test case with 10% utilization for DIV module and 40% for MULT is also considered. It is clear from Table. 7.2 as well as from Fig. 7.5 that we get significant amount of saving in both static and dynamic power as we perform power gating and clock gating. The savings are more profound for lower utilization values as expected. The modules assigned to switchable domain represent almost half of the overall chip area in this case, hence resulting in greater overall power saving as compared to that achieved in the RCA case.

Our third and the most complex test case is an IDCT design of a JPEG IDCT decoder. The IDCT module here represents around 96% of the overall chip area and hence the power savings are even higher in this case than the previous two test cases. Various implementations are considered which involve an unoptimized implementation, followed by optimized implementations with IDCT workload corresponding to JPEG usage and finally by increasing the IDCT usage toggle rates by factors of 4X, 8X and 32X respectively. The results are presented in Table. 7.3 and also illustrated in Fig. 7.6. We get a saving by almost 50% in terms of static power due to power gating. We also obtain a 19X saving in dynamic power saving mainly due to clock gating used in optimized implementations. As the amount of toggling increases, the dynamic power increases due to more frequent switching of the IDCT module while the static power remains the same as the static probability for the various test cases remain the same.

This test case is important for validating our proposed methodology also because it uses a considerable amount of memories, accessing which, normally consumes considerable amount of power. We thus, performed an analysis by considering the complete design and the corresponding memory modules involved in terms of total

Table 7.1 Power versus Area for RCA

Activity	$\mathbf{P_{static}}$ (μW)	$\mathbf{P_{dynamic}}$ (μW)	**Area** (μm^2)
No opt @ 50%	70	255	$Total \Rightarrow 3362$ $MSB\_RCA \Rightarrow 1051$ (31%)
90%	35	104	
70%	31	97	
50%	28	93	
30%	25	90	

Table 7.2 Power versus Area for ALU processor

Activity DIV-MULT	$\mathbf{P_{static}}$ (μW)	$\mathbf{P_{dynamic}}$ (μW)	**Area** (μm^2)
No opt @ 10%-40%	190	1072	$Total \Rightarrow 27361$ $DIV \Rightarrow 8219$ (30%) $MULT \Rightarrow 4790$ (18%)
30%-60%	140	341	
20%-50%	133	283	
10%-40%	129	237	
1%-10%	86	160	

power consumption and chip area. This was necessary to demonstrate that the IDCT design uses a considerable amount of RAM resources and hence is a good candidate to demonstrate that our proposed methodology can be applicable to optimize designs with memory accesses consuming significant amount of power. The results are given in Table. 7.4. In case of JPEG usage, the static power due to memory accesses is about 45% of the total power consumed by the module. The RAM dynamic power on the other hand is around 36% of the total power while the area corresponding to RAM is almost half of the overall chip area. These parameters show that the design consists of a considerable amount of memory accesses and can help in verifying our power optimization methodology for memory access intensive designs in general. It shall be noted that the factor of RAM dynamic power versus total dynamic power after optimizations is increased from 11% in unoptimized case to around 36% in the optimized case. This may be because, we employ a very global form of CG in our analysis which is not that effective as far as RAMs are concerned as their write enable pin is already configured carefully to do an almost-perfect clock gating.

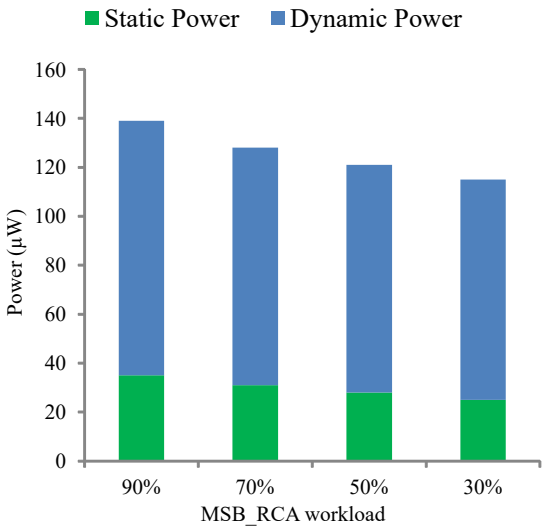

Fig. 7.4 RCA Power curve wrt MSB_RCA workload

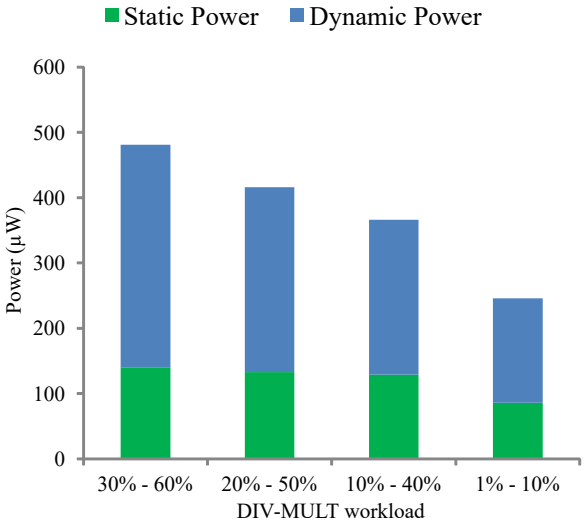

Fig. 7.5 ALU Power curve wrt DIV-MULT workload

Table 7.3 Power versus Area for IDCT

Activity	$\mathbf{P_{static}}$ (μW)	$\mathbf{P_{dynamic}}$ (μW)	**Area** (μm^2)
No opt	572	12570	$Total \Rightarrow 44124$ $IDCT \Rightarrow 42271$ (96%)
32X toggle on enable	240	849	
8X toggle on enable	240	726	
4X toggle on enable	240	680	
JPEG case	240	655	

Table 7.4 Complete IDCT design versus RAM wrt area and power consumption

Activity	$\mathbf{P_{static}}$		$\mathbf{P_{dynamic}}$		**Area** (μm^2)	
	Total	**RAM**	**Total**	**RAM**	**Total**	**RAM**
No opt	572	280	12570	1396	43919	21156
32X toggle on enable	240	108	849	258	44124	21490
8X toggle on enable	240	108	726	250	44124	21490
4X toggle on enable	240	108	680	240	44124	21490
JPEG usage	240	108	655	237	44124	21490

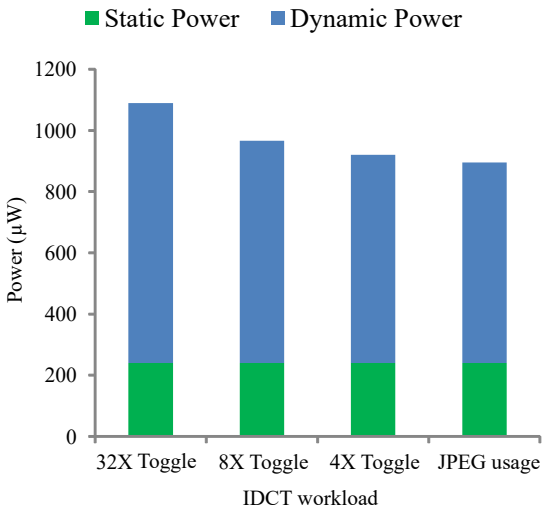

Fig. 7.6 Power curve wrt IDCT workload

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis is a collection of activities carried out during my doctoral studies regarding electronic design automation and methodology in general and power/energy-efficient hardware design methodologies in particular. Specifically, it consists of a combination of two loosely bound research activities that were performed during this duration.

The first activity was regarding the prospects of using FPGAs (rather than GPUs) as hardware accelerators in future HPC systems. FPGAs can offer considerable operational capabilities while consuming only a fraction of the overall power consumed by several high-end GPUs. This is mainly due to the fact that the control structure in the case of FPGA is hardwired thereby eliminating the need to fetch, decode and execute instructions. Furthermore, it is possible to customize the on-chip global memory for applications merely by using some HLS directives, hence resulting in a drastic reduction in multiplexing energy costs. The main hindrance however, in their utilization as hardware accelerators is the complexity in programming them.

To counter this, we explored the idea of using HLS to implement a widely used classification algorithm namely the KNN algorithm on an FPGA directly from its OpenCL code, by utilizing the SDAccel tool from Xilinx. Our analysis showed that, though FPGAs generally offer better power/energy efficiency as compared to GPUs, yet by performing a thorough analysis of the algorithm characteristics, we can have an FPGA implementation that is superior to the GPU even in terms of

execution time. Thus, we were able to have an FPGA implementation that was pareto-optimal in comparison to GPU implementations in terms of energy, power as well as performance.

It exploited the notion that for an FPGA implementation to outperform GPU, we must rely less on the off-chip DRAM and must use the on-chip BRAM more. To that effect, we exploited an optimization offered by SDAccel called "On-chip global memory" optimization to map buffers used for inter-kernel communications to on-chip BRAMs as well as data streaming options. Both of them are very difficult to realize in a GPU e.g. through OpenCL pipes, but are more conveniently available on an FPGA. Moreover, the pareto-optimal implementation also had a kernel i.e. the NEIGHBOR ESTIMATION kernel, consisting of large number of branching operations. Algorithms with such large number of conditionals can adversely effect the performance on a GPU-based platform due to the "thread divergence" problem but they can still be pipelined on an FPGA-based platform.

Thus based on these findings, we can conclude that generally algorithms would work efficiently on FPGAs in comparison to GPUs if they can exploit the comparatively flexible memory architecture of FPGAs in a better way. Local memory in a GPU is managed by a full interconnect network with arbitration. As such, it performs well only when access patterns by work items match its fixed bank structure well. So the algorithmic code needs to be changed in order to adapt to the memory, and it becomes GPU-specific. On the other hand, memory in case of an FPGA can be partitioned based on directives, without the need to change the algorithmic code. It can also assist in data streaming i.e. by utilizing the on-chip global memory buffers. Additionally, algorithms with several conditionals can cause thread divergence on GPUs but they can still be pipelined and hence executed very efficiently on an FPGA.

The second part of the research was pertaining to an HLS-based low power methodology for an ASIC design flow. In particular, the target was to develop a methodology to enable automatic extraction of relevant information for a given design context in order to automatically write a CPF file to perform PSO at the instances of design inactivity. This was accomplished by developing an LP-HLS methodology. This methodology is essentially based on the description of a generic power management module in SystemC to provide the signals necessary to achieve PSO. Additionally, it also consists of a tool that can generate the necessary low power directives (compliant to CPF syntax) to achieve PSO in a given design context.

Multiple hardware accelerators ranging from simple designs to moderately complex ones, were developed in SystemC as test cases to validate the proposed methodology. The design test cases considered were a 32-bit hierarchical RCA, an ALU processor and an IDCT design normally used for data compression. Clock gating was utilized as well to save dynamic power by gating clocks at the instances when the design was being power gated. The main aim of this activity was to achieve a significant reduction in design effort by enabling designers to extract power intent automatically for modular designs, while still utilizing HLS to obtain a wide range of target system implementations. IDCT design presented a test case utilizing significant memory modules which was important as it enabled us to validate our LP-HLS methodology for designs containing considerable number of memory accesses (and the corresponding higher power consumption). Power analysis after logic synthesis (using RTL compiler from Cadence) was performed for a wide range of design usage scenarios and the results validated the ability of our proposed methodology to accurately derive the power intent for the example test cases.

8.2 Future Work

As a future extension of the former activity, we intend to use the findings from that activity to develop an HLS-based methodology for accelerating OpenCL kernels while minimizing energy consumption through FPGA implementation. This would involve enhancing the level of automation in presently available HLS tools beginning with a non hardware-specific OpenCL model. The findings from this intended future activity will contribute significantly to the transition to exascale computing (in modern HPC systems) by reducing the overall energy costs while keeping a check on the algorithm design and optimization costs.

As far as the later activity is concerned, the developed methodology can be extended to include the support for the UPF standard for low power architectures. The work can also be extended to automatically generate CPF files for other power optimization options e.g. multiple supply voltage and dynamic voltage frequency scaling.

References

[1] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014.

[2] Iakovos Mavroidis, Ioannis Papaefstathiou, Luciano Lavagno, Dimitrios S Nikolopoulos, Dirk Koch, John Goodacre, Ioannis Sourdis, Vassilis Papaefstathiou, Marcello Coppola, and Manuel Palomino. Ecoscale: Reconfigurable computing and runtime system for future exascale systems. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 696–701. IEEE, 2016.

[3] Christian De Schryver, Ivan Shcherbakov, Frank Kienle, Norbert Wehn, Henning Marxen, Anton Kostiuk, and Ralf Korn. An energy efficient fpga accelerator for monte carlo option pricing with the heston model. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 468–474. IEEE, 2011.

[4] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2, 2015.

[5] Fahad Bin Muslim, Affaq Qamar, and Luciano Lavagno. Low power methodology for an asic design flow based on high-level synthesis. In *Software, Telecommunications and Computer Networks (SoftCOM), 2015 23rd International Conference on*, pages 11–15. IEEE, 2015.

[6] Blaise Barney et al. Introduction to parallel computing. *Lawrence Livermore National Laboratory*, 6(13):10, 2010.

[7] Sparsh Mittal and Jeffrey S Vetter. A survey of methods for analyzing and improving gpu energy efficiency. *ACM Computing Surveys (CSUR)*, 47(2):19, 2015.

[8] Jian Ouyang, Shiding Lin, Wei Qi, Yong Wang, Bo Yu, and Song Jiang. Sda: Software-defined accelerator for largescale dnn systems. In *Hot Chips*, volume 26, 2014.

[9] microsoft extends fpga reach from bing to deep learning. http://www.nextplatform.com/2015/08/27/microsoft-extends-fpga-reach-from-bing-to-deep-learning/, 2015. [Online; accessed 12-July-2016].

[10] Rick Weber, Akila Gothandaraman, Robert J Hinde, and Gregory D Peterson. Comparing hardware accelerators in scientific applications: A case study. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):58–68, 2011.

[11] Luka Daoud, Dawid Zydek, and Henry Selvaraj. A survey of high level synthesis languages, tools, and compilers for reconfigurable high performance computing. In *Advances in Systems Science*, pages 483–492. Springer, 2014.

[12] Jason Cong. From design to design automation. In *Proceedings of the 2014 on International symposium on physical design*, pages 121–126. ACM, 2014.

[13] Selvaraj Ravi and Michael Joseph. High-level test synthesis: A survey from synthesis process flow perspective. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 19(4):38, 2014.

[14] Xilinx. *SDAccel Development Environment Methodology Guide Performance Optimization*. Xilinx.

[15] Xilinx. *SDAccel Development Environment User Guide*. Xilinx.

[16] Xilinx. *Vivado Design Suite User Guide High-Level Synthesis*. Xilinx.

[17] Zhiru Zhang, Deming Chen, Steve Dai, and Keith Campbell. High-level synthesis for low-power design. *IPSJ Transactions on System LSI Design Methodology*, 8(0):12–25, 2015.

[18] Masanori Kurimoto, Takeshi Yamamoto, Satoshi Nakano, Atsuto Hanami, and Hiroyuki Kondo. Verification work reduction methodology in low-power chip implementation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(1):12, 2013.

[19] Can "less than moore" fdsoi provide better roi for mobile ic? https://www.semiwiki.com/forum/content/2103-can-%C2%93less-than-moore%C2%94-fdsoi-provides-better-roi-mobile-ic.html, 2013. [Online;accessed 30-August-2016].

[20] Preeti Ranjan Panda, BVN Silpa, Aviral Shrivastava, and Krishnaiah Gummidipudi. *Power-efficient system design*. Springer Science & Business Media, 2010.

[21] Li Li, Ken Choi, and Haiqing Nan. Effective algorithm for integrating clock gating and power gating to reduce dynamic and active leakage power simultaneously. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, pages 1–6. IEEE, 2011.

[22] Silicon Integration Initiative et al. Common power format specification 2.0. silicon integration initiative. *Inc., Feb*, 2012.

[23] Rehan Ahmed. *Towards high-level leakage power reduction techniques for FPGAs*. PhD thesis, University of British Columbia, 2015.

[24] Rehan Ahmed, Assem AM Bsoul, Steven JE Wilton, Peter Hallschmid, and Richard Klukas. High-level synthesis-based design methodology for dynamic power-gated fpgas. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2014.

[25] Rehan Ahmed, Steven JE Wilton, Peter Hallschmid, and Richard Klukas. Hierarchical dynamic power-gating in fpgas. In *International Symposium on Applied Reconfigurable Computing*, pages 27–38. Springer, 2015.

[26] Andre R Brodtkorb, Christopher Dyken, Trond R Hagen, Jon M Hjelmervik, and Olaf O Storaasli. State-of-the-art in heterogeneous computing. *Scientific Programming*, 18(1):1–33, 2010.

[27] Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, W Carson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[28] Paulo Francisco Butzen and Renato Perez Ribas. Leakage current in sub-micrometer cmos gates. *Universidade Federal do Rio Grande do Sul*, pages 1–28, 2006.

[29] Benedict Gaster, Lee Howes, David R Kaeli, Perhaad Mistry, and Dana Schaa. *Heterogeneous Computing with OpenCL: Revised OpenCL 1*. Newnes, 2012.

[30] Lee Howes and Aaftab Munshi. The opencl specification, 2015.

[31] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(1-3):66–73, 2010.

[32] David R Kaeli, Perhaad Mistry, Dana Schaa, and Dong Ping Zhang. *Heterogeneous Computing with OpenCL 2.0*. Morgan Kaufmann, 2015.

[33] Vincent Garcia, Eric Debreuve, Frank Nielsen, and Michel Barlaud. K-nearest neighbor search: Fast gpu-based implementations and application to high-dimensional feature matching. In *2010 IEEE International Conference on Image Processing*, pages 3757–3760. IEEE, 2010.

[34] Lars Struyf, Stijn De Beugher, Dong Hoon Van Uytsel, Frans Kanters, and Toon Goedemé. The battle of the giants: a case study of gpu vs fpga optimisation for real-time image processing. In *Proceedings PECCS 2014*, volume 1, pages 112–119. VISIGRAPP, 2014.

[35] Yuliang Pu, Jun Peng, Letian Huang, and John Chen. An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 167–170. IEEE, 2015.

[36] BERKAY Aydin. Parallel algorithms on nearest neighbor search. *Survey paper, Georgia State University*, 2014.

[37] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–6. IEEE, 2008.

[38] microsoft knows exactly where intel's future is. http://www.wired.com/2015/06/microsoft-knows-exactly-intels-future/, 2015. [Online; accessed 12-July-2016].

[39] Deshanand Singh. Implementing fpga design with the opencl standard. *Altera whitepaper*, 2011.

[40] Xilinx. *Vivado Design Suite User Guide Power Analysis and Optimization*. Xilinx.

[41] Luciano Sánchez, Jose Ranilla, and Alberto Cocaña-Fernández. Eecluster: An energy-efficient tool for managing hpc clusters. *Annals of Multicore and GPU Programming*, 2(1):15–24, 2015.

[42] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. IEEE, 2009.

[43] rodinia/opencl/nn. https://github.com/kkushagra/rodinia/tree/master/opencl/nn, 2012. [Online; accessed 12-July-2016].

[44] 2012 hurricane/tropical data for atlantic. http://weather.unisys.com/hurricane/atlantic/2012/index.php, 2012. [Online;accessed 12-July-2016].

[45] Alessandro Cilardo and Luca Gallo. Interplay of loop unrolling and multidimensional memory partitioning in hls. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 163–168. IEEE, 2015.

[46] Memory system design. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/edh_ed51008.pdf, 2010. [Online; accessed 11-August-2016].

[47] Amir Rahmati, Matthew Hicks, Daniel Holcomb, and Kevin Fu. Refreshing thoughts on dram: Power saving vs. data integrity. In *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[48] Assem AM Bsoul, Steven JE Wilton, Kuen Hung Tsoi, and Wayne Luk. An fpga architecture and cad flow supporting dynamically controlled power gating. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(1):178–191, 2016.

[49] Tim Tuan, Sean Kao, Arif Rahman, Satyaki Das, and Steve Trimberger. A 90nm low-power fpga for battery-powered applications. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 3–11. ACM, 2006.

[50] Open SystemC Initiative et al. Systemc version 2.0 user's guide, 2001.

[51] Tobing Soebroto, William Yung, and Andrew Chang. Reducing the dynamic power and leakage power of a high performance soc. *Cadence Design Systems, Inc*, 2006.

[52] Enrico Macii, Leticia Bolzani, Andrea Calimera, Alberto Macii, and Massimo Poncino. Integrating clock gating and power gating for combined dynamic and leakage power optimization in digital cmos circuits. In *Digital System Design Architectures, Methods and Tools, 2008. DSD'08. 11th EUROMICRO Conference on*, pages 298–303. IEEE, 2008.

[53] Rakesh Chadha and J Bhasker. Architectural techniques for low power. In *An ASIC Low Power Primer*, pages 93–111. Springer, 2013.

[54] Shrenik Mehta. Industry standards from accellera. In *21st International Conference on VLSI Design (VLSID 2008)*, pages 728–728. IEEE, 2008.

[55] Cadence. *Low Power in Encounter RTL Compiler*. Cadence.

[56] Cadence. *Toggle Count Format Reference*. Cadence.

[57] Cadence. *Cadence C-to-Silicon Compiler User Guide*. Cadence.

[58] Khalid Sayood. *Introduction to data compression*. Newnes, 2012.

[59] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[60] Cadence. *Encounter Conformal Equivalence Checking User Guide*. Cadence.

[61] Cadence. *Encounter Conformal Low Power User Guide*. Cadence.

[62] Nangate freepdk45 open cell library. http://projects.si2.org/openeda.si2.org/projects/nangatelib, 2011. [Online;accessed 6-September-2016].

[63] Yasaman Samei and Rainer Dömer. Automated estimation of power consumption for rapid system level design. In *2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2014.

[64] Anmol Mathur and Qi Wang. Power reduction techniques and flows at rtl and system level. In *2009 22nd International Conference on VLSI Design*, pages 28–29. IEEE, 2009.

[65] Frank Schirrmeister. Design for low-power at the electronic system level.

[66] Luca Benini and Giovanni de Micheli. System-level power optimization: techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(2):115–192, 2000.

[67] Ieee forms two new working groups to standardize software and system-level energy management and power modeling for system-on-chip devices. http://standards.ieee.org/news/2014/ieee_p2415_p2416_wgs.html, 2014. [Online;accessed 6-September-2016].

[68] Enrico Macii, Massoud Pedram, and Fabio Somenzi. High-level power modeling, estimation, and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1061–1079, 1998.

[69] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(1):5, 2013.

[70] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: a framework for architectural-level power analysis and optimizations*, volume 28. ACM, 2000.

[71] Tung Thanh Hoang, Vineeth Saseendran, Donatas Siaudinis, and Per Larsson-Edefors. Power gating multiplier of embedded processor datapath. In *Ph. D. Research in Microelectronics and Electronics (PRIME), 2011 7th Conference on*, pages 41–44. IEEE, 2011.

[72] Dominik Macko and Katarína Jelemenská. Managing digital-system power at the system level. In *AFRICON, 2013*, pages 1–5. IEEE, 2013.

Appendix A

CPF file for IDCT

```
####################
# Technology part of the CPF #
####################
set_hierarchy_separator /
set_power_unit uW

# Specify libraries #
define_library_set -name lib_wc -libraries
"/tech_libs/nangate/NangateOpenCellLibrary_PDKv1_3_v2010_12/Low_
Power/Front_End/Liberty/ECSM/LowPowerOpenCellLibrary_slow_ecsm.lib
/tech_libs/nangate/NangateOpenCellLibrary_PDKv1_3_v2010_12/
/Front_End/Liberty/ECSM/NangateOpenCellLibrary_slow_ecsm.lib"

# Specify special cells #
## Isolation Cells ##
define_isolation_cell -cells "ISO_FENCE0N_X*" -enable EN
-valid_location to
## Power Switch Cells ##
define_power_switch_cell -cells "HEADER_X*" -power VDD
-power_switchable VVDD -type header -stage_1_enable SLEEP
## Retention Cells ##
define_state_retention_cell -cells "DFFR_X*" -restore_function
RN
```

```
## Always ON Cells ##
define_always_on_cell -cells "AON_BUF_X*"

####################
# Design part of the CPF #
####################
set_design topmodule_rtl

# Declare power/ground nets #
create_power_nets -nets TVDD -voltage 0.95
create_power_nets -nets VDD -internal
create_ground_nets -nets VSS -voltage 0

# Specify power domains #
create_power_domain -name PD_default -default
create_power_domain -name PD_xbus_hw_idct_rtl -instances
{XLXI_3} -shutoff_condition {!XLXI_2/pse}

# Nominal operating conditions #
create_nominal_condition -name off -voltage 0
create_nominal_condition -name on -voltage 0.95

# Modes of operation #
create_power_mode -name PM1 -domain_conditions {PD_default@on
PD_xbus_hw_idct_rtl@on} -default
create_power_mode -name PM2 -domain_conditions {PD_default@on
PD_xbus_hw_idct_rtl@off}

# Design rules #
## Isolation rule ##
create_isolation_rule -name iso1 -from PD_xbus_hw_idct_rtl
-to
PD_default -isolation_condition {XLXI_2/iso_en} -isolation_output
low
-isolation_target from
```

```
## Power switch rule ##
create_power_switch_rule -name psr1 -domain PD_xbus_hw_idct_rtl
-external_power_net TVDD
## State retention rule ##
create_state_retention_rule -name st1 -domain PD_xbus_hw_idct_rtl
-restore_edge {!XLXI_2/ret_en}

####################
# Update libraries #
####################

# Associate library sets with nominal conditions #
update_nominal_condition -name on -library_set lib_wc


# Update the isolation rules #
update_isolation_rules -names iso1 -cells {ISO_FENCE0N_X1
ISO_FENCE0N_X2 ISO_FENCE0N_X4}
# Update powerswitch rules #
update_power_switch_rule -name psr1 -prefix CPF_PS_ -cells
{HEADER_X1 HEADER_X2 HEADER_X4}

# Specify timing constraints #
update_power_mode -name PM1 -sdc_files constraints/mmmc/idct.sdc

# Describing power nets #
create_global_connection -domain PD_default -net TVDD -pins
VDD
create_global_connection -domain PD_default -net VSS -pins
VSS
create_global_connection -domain PD_xbus_hw_idct_rtl -net
VDD -pins VDD
create_global_connection -domain PD_xbus_hw_idct_rtl -net
VSS -pins VSS
```

```
# update Power Domain #
update_power_domain -name PD_default -internal_power_net
TVDD
update_power_domain -name PD_xbus_hw_idct_rtl -internal_power_net
VDD

end_design

# END #
```