Functional Cores: Trade a Little Performance to Save Power

Mohsen Riahi Alam and Mostafa E. Salehi
School of Electrical and Computer Engineering
University of Tehran
Tehran, Iran, 14395-515
{riahialam, mersali}@ut.ac.ir

***Abstract*— Recent process scaling in CMOS technology increases transistor counts while there is limited power budget on chip. As power is more expensive than area, trading area to save power is recommended. In this regime, specialized and energy-efficient processing engines can increase the energy efficiency by reducing energy per computation. To achieve this goal, we introduce specialized hardware accelerators namely functional cores. Functional cores are the hardware realization of a function in software program. Considering energy consumption and performance, these cores perform the functions more efficiently compared to their execution on a general purpose processor. Since decreasing power consumption is more important than execution time in several applications, we exploit hardware accelerators for power reduction rather than performance improvement. To this end, we use high level synthesis for automatic generation of functional cores in ASIC domain. Experimental results obtained on a set of benchmark programs reveal that depending on function size, 32% to 75% reduction in power consumption is observed.**

Keywords*— *High level synthesis, voltage and frequency scaling, low power design, hardware accelerator.

I. INTRODUCTION

Nowadays, the VLSI design concerns have shifted from high performance to low power designs. Specialized hardware implementation of a set of computations improves computational efficiency of general purpose processors. Manual application-specific hardware design is effortful because it is time-demanding and requires complex Register Transfer Level (*RTL*) code. On the other hand, generating an RTL by high level synthesis (*HLS*) for implementing the specified behavior and yet satisfying the design constraints is much easier. Using HLS reduces cost and time-to-market significantly, while manual hardware design is too expensive and time-consuming. HLS also makes the design space exploration more efficient and is a rapid method for software/hardware co-design.

Three main stages of HLS process are compilation, design algorithms, and RTL generation. HLS starts with the algorithmic description of an application, RTL component library, and design constraints to automatically generate RTL mixture of data-path and control logic design [1]. In this work, we use an HLS tool for generating functional cores of hardware/software systems to improve the performance and energy efficiency of the system. The objective is different form common hardware accelerators. Accelerators focus on improving performance, however, the proposed functional cores trade performance to save power.

Static voltage and frequency scaling is a quite popular low power technique in embedded system designs. In these systems, different voltages and frequencies are predetermined [3]. Despite using lower voltage and frequency for each functional core, their performance is still better than general purpose processors. Voltage and frequency scaling improves power efficiency, but decreases the performance. However, hardware accelerators improve performance. We use both of these techniques in functional cores. First, we use hardware accelerator to trade the area for energy efficiency and improving performance. Next, by trading a little performance of the hardware accelerator, we reduce the power consumption.

There are several high-level synthesis frameworks available on different domains. After investigating some new open source HLS tools and their properties, we have chosen LegUp which is a high-level synthesis tool developed at the University of Toronto [2]. LegUp is developed to provide the performance and energy benefits of hardware and gets a standard C program and generates FPGA-based software/hardware system-on-chip. In this work, we use LegUp hybrid flow to generate software/hardware system-on-chip in ASIC domain and then run its hardware accelerator under nominal clock. To achieve a low-power design, we use design time voltage and frequency scaling in our system. To evaluate our proposed system, we need to have some benchmark programs. We have chosen five benchmarks consisting of a great set of computationally intensive programs in various categories from CHStone [4].

The paper is organized as follows: In Section II, we briefly introduce some of the previous work on the energy efficient hardware accelerators and voltage and frequency scaling. System architecture and functional core generation is discussed in Section III & IV. Section V presents our low power idea and motivation. The experimental platform, tools, and results are asserted in Section VI. Finally, the paper is concluded in Section VII.

II. RELATED WORK

Authors in [5] create *conservation cores* which is a new class of circuits and is aimed to increase the energy efficiency of mature applications. They present a toolchain to automatically synthesize conservation cores from some

978-1-4799-1972-7/15/$31.00

applications' source code and then determine whether they can significantly reduce energy and energy-delay for a wide range of applications. Their results show that conservation cores can reduce energy consumption by up to 2.1× for whole applications.

BERET [6] is an energy-efficient co-processor that can be used in a wide range of applications. In this approach, at first, recurring instruction sequences are identified in a program's execution. Then, suitable ones are mapped to the BERET hardware. BERET hardware execution reduces instruction fetch, instruction decode, and register file access energy consumption in the general purpose processor. In average, BERET reduces energy consumption by a factor of 3-4X for the selected program's regions. The average energy savings for the entire application run is 35% over a single-issue in-order processor.

CMOS circuit energy efficiency can be improved by scaling the supply voltage. In [7], authors investigate the effect of lowering the supply and threshold voltages on the energy efficiency of CMOS circuits. Dynamic voltage scaling (DVS) dynamically adjust the supply voltage to meet the performance demands. Using DVS capability, the processor system which is proposed in [8] improves the energy efficiency up to a factor of 10x without decreasing the throughput.

Due to the run time overhead of dynamic voltage scaling, static voltage scaling is a good solution in such embedded systems. A compiler energy optimization strategy based on static voltage scaling is presented in [9]. In this strategy, the compiler determines a single supply voltage level for the input program and it does not change during the execution. The authors also use performance-enhancing compiler optimizations to create opportunities for voltage scaling to save energy rather than increasing program performance. By applying optimizations to a given code, a performance slack can be created which can be defined as a difference between the original (non-optimized) execution time and the optimized execution time. The current trend in optimizing compilers is to exploit this slack to reduce the execution time. However, it can be exploited for more energy reduction in the same performance. In our work, instead of compiler optimizations, we use hardware accelerator for providing more slacks for energy reduction.

III. System Overview

Fig. 1 shows our system-on-chip in ASIC domain that is proposed for specialized and energy-efficient processing. We use LegUp hybrid flow (Hardware/Software Partitioning) in which MIPS processor begins execution of C programs until we reach the function which is best implemented as hardware accelerator (named functional core). Then, MIPS processor sends arguments to the functional core and asserts the start signal of functional core. Then it goes to the stall state until a finish signal is asserted by the functional core triggering MIPS processor to resume its execution. In this system, MIPS and functional cores use a shared data cache. In LegUp's regular design, supply voltage and clock frequency are the same for all of the components of system-on-chip. However, in our low-power design, each functional core might have a different supply voltage and a different clock frequency.

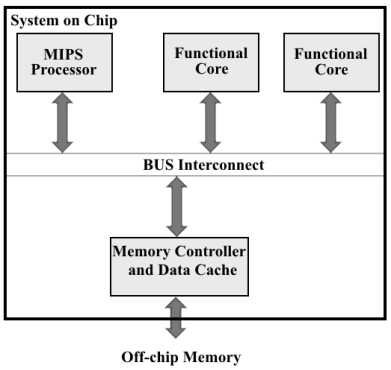

Fig. 1. Our system on chip based on LegUp.

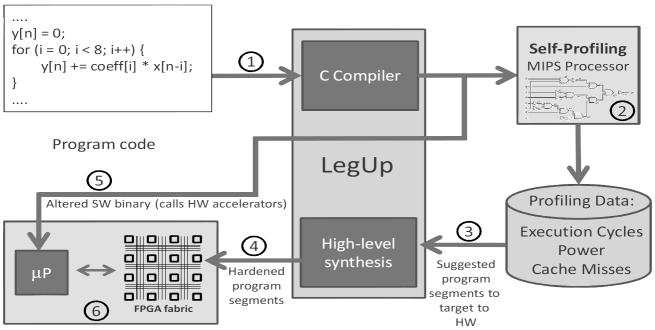

Fig. 2. LegUp hybrid flow[2].

IV. Funcional Core

A. Hardware Accelarators Selection

According to Fig. 2, we apply LegUp's hybrid flow to generate a system by choosing some program segments to be synthesized into FPGA circuits as hardware accelerators while the remaining program segments execute on a soft processor. In LegUp, Tiger MIPS [10] and Altera Avalon interfaces are used as soft processor and processor/accelerator communication media, respectively. LegUp exploits open-source LLVM (low-level virtual machine) [11] for compiler optimizations and high level language parsing as well. LegUp developers create new back-end compiler passes within LLVM for hardware synthesis.

The LegUp framework is capable of profiling a real-time hardware module using an extra circuitry which is coupled with Tiger MIPS processor. The information of each operation is extracted using the profiling feature when running each program on the processor. Comparing the extracted information related to each program segment, the highly suitable program segments are distinguished as the candidates for developing hardware accelerator. Regarding the execution cycles, cache miss stalls, and energy consumption of each function in function-level hardware profiling results, the best candidates of hardware accelerators are chosen among the functions of the five CHStone benchmarks. TABLE I. shows the functions which are chosen for hardware accelerators in the five CHStone benchmarks.

In hybrid flow, LegUp gets the list of C functions of the program which should be implemented as hardware

accelerators. Then, it generates Verilog modules for the main functions and its sub-functions. Finally, LegUp attaches interfaces to them for bus communication.

B. Library Characterization and Synthesis

LegUp is an FPGA-based HLS tool and uses relevant RTL library characterization for FPGA family to assign corresponding hardware operations to each LLVM instruction. In this work, we want to achieve low power Application Specific Integrated Circuit (ASIC) design flow for generating hardware accelerator using LegUp HLS tools. Therefore, an ASIC design RTL library characterization similar to FPGA-based flow is required. For this purpose, various bit-width sequential and combinational logic elements (e.g. adder, subtractor, multiplier, and, or, etc.), which are used to create hardware circuit through HLS, are synthesized to gate-level. This gate level synthesis is done using a digital synthesis tool and 90nm CMOS standard cell library. The results such as dynamic and static power consumption, critical path delay, area, etc. are extracted for each element. LegUp uses this information in its design algorithms.

V. Low Power Idea

There are two main components that constitute the power used by a CMOS integrated circuit: static power and dynamic power. In this paper, we focus on dynamic power consumption and propose a method for decreasing it. Equation 1 shows that dynamic power is proportional to the square of the operating voltage. Therefore, reducing the voltage reduces the power consumption significantly. Furthermore, since frequency is directly proportional to supply voltage, the frequency of the circuit can also be reduced, therefore, a cubic power reduction is achieved [3].

$$P=ACV^2f \quad (1)$$

Design-time voltage and frequency scaling is one of the most common ways to reduce the power consumption. In design flow, firstly circuits are designed to exceed the performance requirements. Then, the supply voltage is reduced to meet the performance constraints of the system [3].

Hardware accelerator is the use of customized hardware to perform functions faster than its software equivalent execution on general purpose processors. Furthermore, execution on hardware accelerators is more energy efficient than general purpose processors. Due to these specifications, *LegUp*'s hybrid execution has higher performance and is more energy efficient compared with software execution. The increased performance in hybrid execution creates a performance time slack which is defined as the difference between the software execution time and the functional core execution time. Since, low power design is our main concern in this work, we trade this performance (time) slack to save more power. In the other words, we decrease the processing speed (static voltage and frequency scaling) of the functional cores based on their acceleration ratios to save more power.

To determine the voltage and frequency scaling amount, we should simulate five *CHStone* benchmarks in our experiments at RT level and find performance slack for each one. First, based on the profiling results, we choose the best function of each benchmark for implementing it as hardware accelerator. Then the hybrid system is generated for it. Next, these benchmarks are simulated in software and hybrid mode at RT level and the number of clock cycles is extracted in two modes. TABLE I. shows the execution clock cycles required for each function that is run on MIPS processor and functional core with their differences as performance slack. These slacks show the quantity of functional cores' acceleration compared to MIPS processor for each selected function. These saving cycles let us decrease functional core's operating voltage and frequency for saving power.

As mentioned before, we used design-time voltage and frequency scaling for the functional cores to decrease dynamic power. According to Fig. 3, we use two new operating points in 90nm process technology [12]. Nominal operating point (N) voltage is 1.2V and its delay is the reference for normalization. Delay of point A and B are 1.5 and 2 times larger than nominal delay, respectively. Also supply voltages of these points are 880mV and 745mV, respectively. Based on the acceleration ratio, we use one of these two points for functional cores execution. In the other words, based on acceleration ratio, each functional core has a period that is x times of the master clock period.

Among the five selected *CHStone* benchmarks which are examined in our experiments, the functional core of *Encode* in *ADPCM* does not improve the performance as much as the other benchmarks. Therefore, we could not scale its operating voltage to point B. However, other benchmarks provide good improvements for performance, so we can scale the operating voltage to 745mV and decrease half of frequency.

TABLE I. Number of Clock Cycles for Selected Functions Executed on MIPS Processor and Funcional Cores.

Selected Function (Benchmarks)	Number of Clock Cycles		
	MIPS	***Functional core***	***slack***
Encode (ADPCM)	64525	39607	24918
Float64_add (dfadd)	255011	958	254053
Float64_mul (dfmul)	112605	287	112318
Gsm_LPC_Analysis (GSM)	31576	6764	24812
Initialize_Buffer (motion)	29383	8500	20883

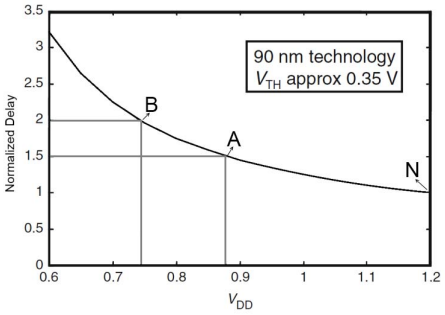

Fig. 3. Nominal operating point and two new operating point[12].

TABLE II. EXECUTION TIME, DYNAMIC ENERGY AND POWER IN THREE DIFFERENT SIMULATION CONFIGURATION FOR FIVE CHSTONE BENCHMARKS.

Selected Function (Benchmarks)	Software mode Clock Frequency 200MHz			Hybrid with regular functional core Clock Frequency 200MHz			Hybrid with scaled functional core			
	Execution Time (ns)	*Dynamic Energy (nJ)*	*Dynamic Power (mW)*	*Execution Time (ns)*	*Dynamic Energy (nJ)*	*Dynamic Power (mW)*	*Scaled Clock Frequency (MHz)*	*Execution Time (ns)*	*Dynamic Energy (nJ)*	*Dynamic Power (mW)*
Encode (ADPCM)	951898	13269	13.94	827305	9803	11.85	133	926323	8706	9.40
Float64_add (dfadd)	1423528	16394	11.52	153260	973	6.35	100	158050	932	5.89
Float64_mul (dfmul)	696358	7563	10.86	134765	735	5.45	100	136200	730	5.36
Gsm_LPC_Analysis (GSM)	312193	4293	13.75	188130	2051	10.90	100	221950	1617	7.28
Initialize_Buffer (motion)	296568	4385	14.79	192150	904	4.71	100	234650	842	3.59

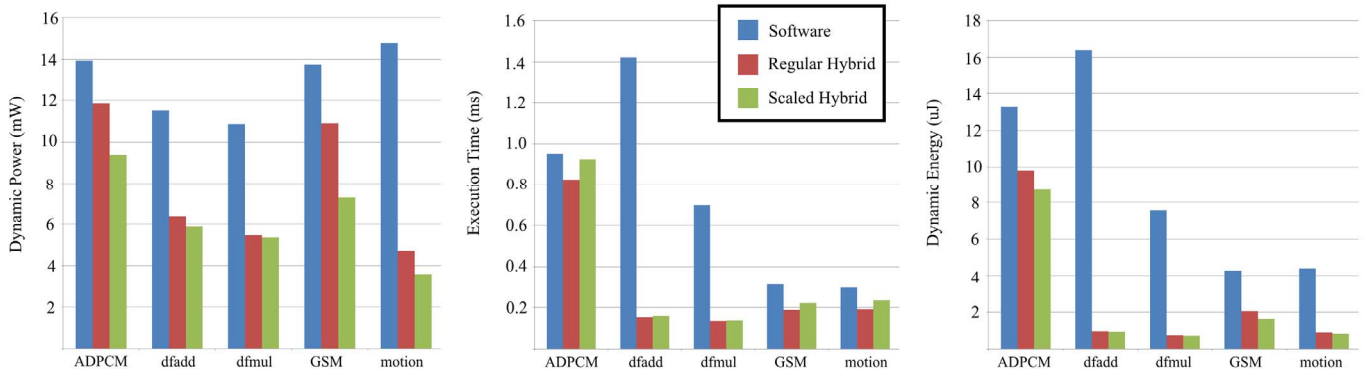

Fig. 4. Execution time, dynamic energy and dynamic power for five CHStone benchmarks in three different simulation configuration.

VI. EXPERIMENTAL RESULTS

A. Experimental Flow

Three different simulation configurations and five *CHStone* benchmarks are used in our experiments. The first simulation configuration is pure software flow, in which the entire benchmark programs run on *Tiger MIPS*. In the second and third simulation configurations, based on *LegUp*'s hybrid flow, the best function for hardware acceleration is converted to the functional cores and the remaining program segments run on *Tiger MIPS*. These benchmarks are simulated at gate level in software and two hybrid modes as well. Clock frequency in the first and second configurations is set to 200MHz for all components. However, in the third configuration clock signal of functional cores have a lower frequency which is determined by acceleration ratios and the slack times presented in TABLE I.

To evaluate our low power techniques and power measurement of each configuration, MIPS processor and functional cores are synthesized with a digital logic synthesis tool using 90nm CMOS standard cell library. For each benchmark the switching activity data (Value Change Dump file) is generated by gate level simulation. These VCDs are analyzed by a commercial power analysis tool and power is estimated for each benchmark program. For processor's instruction cache power measurement, we extract the number of cache accesses in our simulations and use *CACTI* dynamic power model [13] for power estimation.

B. Experimental Results

TABLE II. compares execution time, dynamic power and dynamic energy for three different system configurations. In software mode, as expected, maximum power and energy is consumed with a long execution time compared with two other hybrid modes. In the regular hybrid mode, functional cores improved execution time by 50% in average; however, according to Fig. 4 little improvement about 13% for *ADPCM* benchmark is obtained, which results in a small time slack. This small time slack does not let us to scale voltage and frequency as much as they can be scaled for the other benchmarks.

According to Fig. 4, more power reduction is achieved for the scaled hybrid mode compared to the regular hybrid mode. However, since the execution time is increased, energy consumption is remained quite unchanged.

Fig. 5 shows the percentage of average reductions in execution time, dynamic energy, and power. In regular hybrid mode, exploiting functional cores improves performance, dynamic power and energy consumption. As mentioned before, in such embedded systems, power is more important than performance. To trade performance for power reduction, we scaled voltage and frequency in hybrid mode.

In average, scaled hybrid mode compared with regular hybrid mode, has a longer execution time and its dynamic power is 10% more reduced.

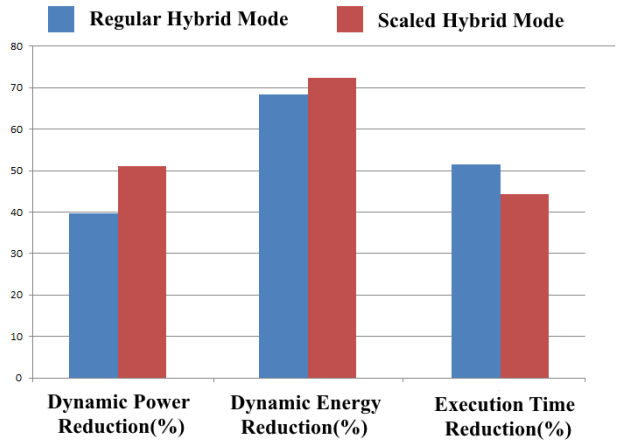

Fig. 5. Average execution time, dynamic power and dynamic energy reductions for five CHStone benchmarks in two hybrid mode simulation.

VII. Conclusion

Power dissipation limits the capabilities of embedded systems. Therefore, power reduction techniques are very important in embedded system designs. In this work, we focus on power/performance trade-offs in application-specific embedded architectures. Five benchmark programs are simulated using three different system configurations including: pure software flow where the entire program is executed on the processor, regular hybrid flow where proper accelerators are incorporated to perform computation intensive functions, and proposed scaled hybrid flow where design-time voltage and frequency scaling is applied to the accelerators. Evaluation metrics (execution time, dynamic power, and energy consumption) are evaluated for each benchmark and configuration. Based on the simulation results, for regular hybrid flow, the evaluation metrics are improved by 51%, 39%, and 68%, respectively. Using our proposed scaled hybrid scheme, the performance improvements can be traded in favor of dynamic power reduction, which is more critical in embedded systems. Compared with software implementation, the evaluation metrics are improved by 44%, 51%, and 72% using scaled hybrid flow, which confirms the effectiveness of the proposed scheme for embedded systems.

References

[1] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Des. Test Comput.*, vol. 26, no. 4, pp. 8–17, 2009.

[2] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, pp. 1-27, 2013.

[3] P. R. Panda, B. V. N. Silpa, A. Shrivastava, and K. Gummidipudi, Power-efficient System Design. Springer US, 2010.

[4] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *IEEE International Symposium on Circuits and Systems, (ISCAS),* pp. 1192–1195, 2008.

[5] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: reducing the energy of mature computations," in *ACM SIGARCH Computer Architecture News,* vol. 38, no. 1, pp. 205–218, 2010.

[6] S. Gupta, S. Feng, A. Ansari, S. Mahlke, and D. August, "Bundled Execution of Recurring Traces for Energy-efficient General Purpose Processing," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture,* pp. 12–23, 2011.

[7] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power CMOS," IEEE J. on Solid-State Circuits, , vol. 32, no. 8, pp. 1210–1216, 1997.

[8] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," IEEE J. on Solid-State Circuits, , vol. 35, no. 11, pp. 1571–1580, 2000.

[9] H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C.-H. Hsu, and U. Kremer, "Energy-conscious compilation based on voltage scaling," in *ACM SIGPLAN Notices,* vol. 37, no. 7, pp. 2–11, 2002.

[10] tigerMIPS University of Cambridge 2010. The Tiger MIPS processor (http://www.cl.cam.ac.uk/teaching/0910/ECAD+Arch/mips.html).University of Cambridge.

[11] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis transformation," *International Symposium on Code Generation and Optimization(CGO)*, pp. 75–86, 2004.

[12] J. Rabaey, Low Power Design Essentials. Springer, 2009.

[13] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI6.0: A Tool to Model Large Caches," *HP Laboratories, Tech. Rep.* HPL-2009-85, 2009.