Energy Reduction with Run-Time Partial Reconfiguration

Shaoshan Liu, Richard Neil Pittman, Alessandro Forin
Microsoft Research

September 2009

Technical Report
MSR-TR-2009- 2017

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Energy Reduction with Run-Time Partial Reconfiguration

Shaoshan Liu, Richard Neil Pittman, Alessandro Forin
Microsoft Research

ABSTRACT

In this paper we investigate whether partial reconfiguration can be used to reduce FPGA energy consumption. The core idea is that within a hardware design there are a number of independent circuits, and some can be idle for long periods of time. Idle circuits still consume power though, especially through clock oscillation and static leakage. Using partial reconfiguration we can replace these circuits during their idle time with others that consume much less power. Since the reconfiguration process itself introduces energy overhead, it is unclear whether this approach actually leads to an overall energy saving or to a loss. This study identifies the precise conditions under which partial reconfiguration reduces the total energy consumption, and proposes solutions that minimize the configuration energy overhead. Partial reconfiguration is compared against clock gating to evaluate its effectiveness. We apply these techniques to an existing embedded microprocessor design, and show how FPGAs can be used to accelerate application performance while also reducing overall energy consumption.

1. INTRODUCTION

Portable embedded systems, including PDA, mobile phones, and digital cameras have become increasingly popular in recent years. The most important resource in a portable system is its battery, which is a finite energy resource with varying and limited recharge opportunities. However, many embedded devices also demand high performance. For instance, multimedia applications exhibit dynamic program behavior, impose strict real-time constraints, and demand intensive computation capabilities. What makes the situation even worse is that many of the embedded devices have a limited chip area budget, making it infeasible to pack multiple hardware accelerators onto one chip. As a consequence, embedded systems are required to consume little energy and deliver high performance on a dynamic set of applications.

A major advantage of reconfigurable computing systems is their ability to modify the underlying hardware either to deliver high performance or to reduce energy consumption. Thus, reconfigurable computing systems may meet the requirements imposed by many embedded systems. Although many past studies have proposed optimized hardware accelerators [12, 13], little work has focused on leveraging the runtime partial reconfiguration feature to reduce energy consumption. In this paper, we study the impact of using partial reconfiguration to unload accelerators when they are not being used in order to reduce static and dynamic power consumption. One concern with this approach is that the configuration process itself introduces energy overhead, possibly negating any gains from unloading parts of the system. To look at this problem in more detail, we raise the following three questions:

1. Accelerator can reduce program execution time. Does the acceleration hardware also lead to energy reduction as a result of the execution time reduction? If so, under what conditions?
2. Is it worthwhile to use partial reconfiguration to reduce energy consumption in reconfigurable systems? If so, under what conditions?
3. Can partial reconfiguration energy management techniques outperform other techniques such as clock gating? If so, under what conditions?

We intend to answer these questions in the rest of this paper, which is organized as follows: in section 2 we review the related work in FPGA power management and partial reconfiguration; in section 3 we translate our three basic questions into analytical models and identify the conditions for energy reduction; recognizing that we can minimize the configuration overhead by maximizing the configuration speed, in section 4 we present our design of a fully streaming DMA engine to minimize configuration overhead; in section 5 we present the experimental results and we conclude in section 6.

2. BACKGROUND

In this section, we discuss the related work in reconfigurable system power management, partial reconfiguration, as well as introduce the Virtex-4 FPGA and the eMIPS platform on which we performed our experiments

2.1 Power Management

There are two main sources of power consumption in a reconfigurable computing system: static power and dynamic power. Static power consumption occurs as a result of leakage current in the transistors, whereas dynamic power is incurred when the transistors switch. A recent study released by Xilinx [1] indicates that below 0.25 microns static power has grown exponentially with each new process. This study confirms that static power is becoming the largest component of total power consumption in an FPGA. In [2], Tuan *et al.* studied the leakage power of 90 nm FPGA using detailed device-level simulations. Specifically, their study found out that static power consumption directly depends on the values of the configuration bits. Furthermore on this topic, in [3], Anderson *et al.* studied active leakage power optimization techniques for FPGAs. They indicated that the polarity of the inputs and outputs of the circuits in FPGA have a strong impact on the leakage power consumption. Specifically, they indicated that in a modern commercial CMOS process, the leakage power dissipated by elementary FPGA hardware structures, such as buffers and multiplexers, is significantly smaller when the outputs and inputs of these structures are logic 1 versus logic 0.

The other branch of power management techniques deals with dynamic power. In [4], Wang *et al.* studied the impact of clock power on overall chip power consumption. They found that clock distribution can contribute up to 22% of overall power consumption and implemented clock gating on the Virtex-5. Clock gating techniques selectively turn on/off specific branches of the on-chip clock distribution network in order to reduce clock distribution power. They reported clock gating led to an average of 13.5% power reduction on various benchmark circuits. Another technique to reduce clock power is clock scaling. In this technique, when an application does not require high performance it can reduce its clock frequency. In [5], Paulsson *et al.* implemented dynamic clock scaling in a Xilinx Spartan 3 by changing the settings of the Digital Clock Manager (DCM) during runtime such that the clock frequency of the system could scale down to save power when high performance was not necessary. Their results demonstrated that clock scaling led to 8% power saving. More interestingly, their results implied the importance of runtime reconfiguration for the express purpose of power saving.

Supply gating techniques divide the FPGA chip into small regions and switch on/off the power supply to each region using a sleep transistor in order to conserve leakage energy. In [6], Gayasen *et al.* proposed reducing leakage energy in FPGAs using a region-constrained placement. In their scheme, unneeded components can be turned off at runtime. Similarly, Bharadwaj *et al.* [7] proposed a new architecture for standby power management in clustered island-style FPGAs. In this technique, a sleep transistor is used to control the power supply to a *bucket*, which is defined as the basic granularity level for power management. This class of techniques often requires the modification of FPGA architectures as well as the underlying hardware implementations. In this paper, we focus on energy reduction techniques that can be directly applied to existing FPGAs. Specifically, we study the potential of partial reconfiguration to reduce energy and compare partial reconfiguration to clock gating.

2.2 Partial Reconfiguration

Runtime partial reconfiguration (PR) is a special feature offered by Xilinx FPGAs that allows designers to reconfigure certain portions of the FPGA during runtime without influencing other parts of the design. This feature allows the hardware to be adaptive to a changing environment. First, it allows optimized hardware implementation to accelerate computation. Second, it allows efficient use of chip area such that different hardware modules can be swapped in/out the chip at runtime. Last, it may allow leakage and clock distribution power saving by unloading hardware modules that are not active. One major issue of PR is the configuration speed because the reconfiguration process incurs performance and power overhead. By maximizing the configuration speed, these overheads can be minimized.

In [8], to improve the reconfiguration speed, Liu *et al.* proposed to use direct memory access (DMA) techniques to directly transfer configuration data to the Internal Configuration Access Port (ICAP). They reported to have achieved 82 Mbytes/s ICAP throughput using this approach. In addition, they placed a block RAM (BRAM) cache next to ICAP so as to increase the ICAP throughput to 378 Mbytes/s. However, since on-chip storage resources are precious and scarce, putting a large BRAM next to ICAP is not a practical approach. Similarly, in [9], Claus *et al.* also designed a DMA engine to provide high configuration throughput and they reported to have achieved 295 Mbytes/s on the Virtex-4 chip.

According to [10], on the Virtex-4 chip, the ICAP can run at 100 MHz and in each cycle, it is able to receive 4 bytes, thus the idealized ICAP throughput is 400 Mbytes/s. In section 4 of this paper, we propose a DMA engine design to approach this ideal throughput and study how the configuration throughput affects the energy reduction potential of partial reconfiguration techniques.

2.3 The Virtex-4 FPGA

The Virtex-4 FPGA consists of two layers. The first layer is the logic and memory layer: it contains the reconfigurable hardware including logic blocks (CLBs), block RAMs (BRAMs), I/O blocks, and configurable wiring resources. The second layer contains the configuration memory as well as additional configuration and control logic which handle the configuration bitstream loading and the configuration data distribution. The smallest piece of reconfiguration information that can be sent to the FPGA is called a frame. A frame contains the configuration information needed to configure blocks of 16 CLBs.

Dynamic partial reconfiguration is one of the key features of Virtex-4 FPGA, such that at runtime a hardware accelerator can be loaded onto or unloaded from the FPGA chip. The Internal Configuration Access Port (ICAP) allows internal access to read and write the FPGA's configuration memory, thus it allows self-reconfiguration of Xilinx Virtex devices. On the Virtex-4 chip, the ICAP is able to run at 100 MHz and in each cycle it is able to consume 4 bytes of configuration data, thus the ideal ICAP throughput is 400 Mbytes/s. Also, we store the configuration data in the external SRAM, which runs at 100 MHz and at its maximum speed, it is able to output 4 bytes per cycle. Thus the SRAM also has a maximum throughput of 400 Mbytes/s.

Clock gating can be implemented on Virtex-4 FPGAs in a straightforward manner. The Virtex-4 FPGA architecture consists of 12 clock regions, each region is 16 CLBs tall and spans over half of the chip. The clock distribution network has a branch entering into each clock region. To reduce the dynamic power consumption of portions of a design, such as a hardware accelerator, we can shut down one or more branches of the clock network. To achieve this, we can locate the accelerator in one or more clock regions and connect the accelerator clock to the output of a BUFMUX resource. There are two inputs to this BUFMUX resource, one input is the global clock whereas the other input can be tied to ground. Then at runtime, we can manipulate the select signal to the BUFMUX to turn on/off the accelerator clock.

2.4 The eMIPS System

The eMIPS system is based on a dynamically extensible architecture [11]. The eMIPS architecture allows additional logic to interface and interact with the basic data path at all stages of the pipeline. The additional logic, which is termed Extensions, can be loaded on-chip dynamically during execution by the processor itself. Thus, the architecture possesses the unique ability to extend its own ISA at run-time. In the eMIPS system, the pipeline stages, general purpose register file, and memory interface match those in the classic MIPS RISC processor. The eMIPS system augments the basic MIPS architecture to include all the facilities for self-extension, including instructions for loading, unloading, disabling, and controlling the unallocated blocks in the microprocessor.

The partially reconfigurable Extensions distinguish the eMIPS architecture from the conventional RISC architecture from which it is derived. Through the Extensions the processor overcomes two major shortcomings of the RISC architecture, namely, inflexibility and inability to evolve with changing needs. Using the partial reconfiguration design flow, the eMIPS system can be partitioned into fixed and reconfigurable regions such that the TISA is included in the fixed region, whereas the Extensions are included in the reconfigurable regions. Extensions have been used for accelerating application execution, for implementing plug and play on-chip peripherals, for monitoring and model-checking applications, and for debugging application software during development.

In this paper, we use the eMIPS system to study the impact of partial reconfiguration on the system energy consumption behavior. Specifically, we implement a fully streaming DMA design to reduce the overheads of the configuration process. In addition, we use the Extension "mmldiv64," which is a reconfigurable hardware module designed to accelerate the 64-bit division operations. It has been shown to achieve a 2x speedup on a simple test application.

3. ANALYTICAL MODELS

In this section, we provide analytical models to answer the three questions raised in the introduction. Through this analysis, we are able to identify the conditions under which run-time partial reconfiguration (PR) leads to energy saving.

3.1 Can Acceleration Hardware Lead to Energy Reduction?

The purpose of a hardware accelerator is to accelerate program execution, and we pay extra power for it. In this subsection, we consider the case where the hardware accelerator is always loaded such that it always consumes clock as well as static power even if it is not active.

The core question can be translated into equation 1, where E_{prah} represents the energy consumption with the reconfigurable acceleration hardware turned on and E_{bl} represents the energy consumption of the baseline design without the acceleration hardware. Energy is a product of power and time, thus equation 1 can be expanded into equation 2, where P_{prah} represents the power consumption of the design with the acceleration hardware. The acceleration hardware incurs extra power, thus P_{prah} should be higher than the baseline power $P_{bl.}$ On the other hand, t_{prah} is the time for the design with acceleration hardware to finish the program; it should be lower than the baseline time t_{bl} because the acceleration hardware is meant to accelerate program execution.

In order to simplify the conditions under which the acceleration hardware leads to overall energy saving, we define two parameters: speed-up, *SU*, is the ratio of the baseline execution time, t_{bl}, over the execution time of the design with the acceleration hardware, t_{prah}; and power-up, *PU*, the ratio of the power of the design with the acceleration hardware, P_{prah}, over the baseline power, P_{bl}. By using *PU* and *SU* in equation 2, we derive equation 3, and finally arrive to the conclusion: equation 4 shows that PR hardware accelerator leads to energy reduction if the ratio of *PU* over *SU* is less than 1.

(1) $E_{prah} < E_{bl}\,?$

(2) $P_{prah} \times {}_{prah} < P_{bl} \times {}_{bl}$

(3) $(P_{bl} \times PU) \times ({}^{t_{bl}}\!/_{SU}) < P_{bl} \times {}_{bl}$

(4) ${}^{PU}\!/_{SU} <$

3.2 Can Run-Time PR Lead to Energy Saving?

In this subsection, we consider the idea of blanking the region that hosts the accelerator when it is not active, using dynamic partial reconfiguration. By blanking, we mean writing a *blank* bitstream file to the region in order to unload the accelerator from the design. As shown later in this paper, in this way, we are able to reduce both the clock and static power of the region when the acceleration hardware is not needed. For this, we have to pay the energy dissipation overhead during runtime reconfiguration.

The core question can be translated into equation 5, where E_{pr} represents the energy consumption during the partial reconfiguration process; and E_{saving} represents the resulting energy saving by unloading the acceleration hardware. In equation 6, E_{pr} is actually a product of the power consumption during partial reconfiguration, $P_{pr,}$ and the time taken for partial reconfiguration, t_{pr}. Similarly, E_{saving} is a product of the power of the hardware accelerator, $P_{ext,}$ and the time during which the acceleration hardware is not being used, $t_{inactive}$.

Note that during partial reconfiguration, we are sending a configuration bitstream file to the ICAP port, which then writes the configuration data to the configuration memory. Thus, the parameter P_{pr} depends on the architecture and hardware design of the chip. Similarly, the parameter P_{ext} depends on the hardware design of the hardware accelerator; and the parameter $t_{inactive}$ depends on the application. Our experiments show that reconfiguration time t_{pr} is actually a pure function of the size of the bitstream file, S_{bf}, over the transfer throughput to ICAP, T_{icap}. Because we have little control over the hardware design parameters P_{ext} and $P_{pr,}$ or the application-dependent parameter $t_{inactive}$, the key to reduce the energy overhead of partial configuration in a general way is to increase the throughput of the configuration data transfer. Equations 7 and 8 capture these relationships and identify the lower bound of the ICAP throughput that is necessary for the runtime partial reconfiguration technique to introduce energy reduction.

(5) $E_{pr} < E_{saving}\,?$

(6) $P_{pr} \times {}_{pr} < P_{ext} \times {}_{inactive}$

(7) $P_{pr} \times ({}^{S_{bf}}\!/_{T_{icap}}) < P_{ext} \times {}_{inactive}$

(8) $T_{icap} > {}^{P_{pr} \times S_{bf}}\!/_{P_{ext} \times {}_{inactive}}$

3.3 Can Run-Time PR Outperform Clock Gating?

As indicated in the previous subsection, partial reconfiguration can reduce both clock and static power when the hardware accelerator is not being used, but at the same time the reconfiguration process introduces some energy overhead. On the other hand, clock gating introduces little overhead since no partial reconfiguration is required; however, it reduces only the clock distribution power but not the static power. In this subsection, we consider the idea of turning off the branch of the clock network that goes into the hardware extension, when the accelerator is not active.

The core question can be translated into equation 9, where E_{saving} and E_{pr} are defined in the previous subsection; and $E_{saving\text{-}cg}$ represents the energy saving as a result of the clock gating technique. As shown in equation 10, we decompose the accelerator power P_{ext} into static power P_{static} and clock power P_{clock}, and the energy saving incurred by the clock gating technique is just a product of the clock power P_{clock} and the time during which the hardware accelerator is not being used, $t_{inactive}$.

By simplifying equation 10 we derive equations 11 and 12, which identify the lower bound of ICAP throughput that is necessary for the runtime partial reconfiguration technique to produce more energy reduction than the clock gating technique. Note that in equation 12, P_{static} replaces the term P_{ext} as in equation 8. Since P_{static} is only a fraction of P_{ext}, equation 12 actually imposes a tighter bound on the ICAP throughput compared to equation 8.

$$(9)\quad E_{saving} - E_{pr} > E_{saving-cg}\ ?$$

$$(10)\quad (P_{static} + P_{clock}) \times t_{inactive} - P_{pr} \times ({S_{bf}}/{T_{icap}}) > P_{clock} \times t_{inactive}$$

$$(11)\quad P_{static} \times t_{inactive} > P_{pr} \times {S_{bf}}/{T_{icap}})$$

$$(12)\quad T_{icap} > {P_{pr} \times S_{bf}}/{P_{static} \times t_{inactive}}$$

4. STREAMING DMA ENGINES FOR THE ICAP PORT

As shown in the analytical models presented in the previous section, the key for the runtime partial reconfiguration technique to lead to energy reduction is the configuration data transfer throughput to the ICAP. In this section, we design and implement a direct memory access (DMA) engine to establish a direct transfer link between the external SRAM, where the configuration files are stored, and the ICAP. We demonstrate that our DMA design achieve close to ideal performance.

4.1 Design of the Streaming DMA Engines

Figure 1 shows our system design for partial reconfiguration. In the original design, the ICAP Controller contains only the ICAP and the ICAP FSM, and the SRAM Controller only contains the SRAM Bridge and the SRAM Interface. Hence, in the original design, there is no direct memory access between SRAM and ICAP and all configuration data transfers are done in software. In this way, the pipeline issues one read instruction to fetch a configuration word from SRAM, and then issues a write instruction to send the word to ICAP; instructions are also fetched from SRAM, and this process repeats until the transfer process completes. This scheme is highly inefficient because for the transfer of one word it requires tens of cycles, and the ICAP transfer throughput of this design is only 318Kbytes/s. In order to achieve close to ideal ICAP throughput, our streaming DMA design provides three features: master-slave DMA engines, a FIFO between the two DMA engines, and burst mode to support data streaming.

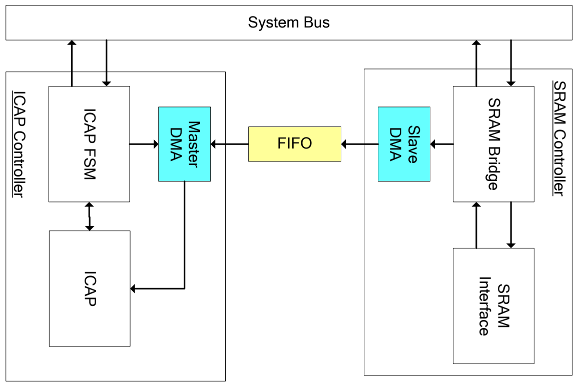

Figure 1: Structure of the Master-Slave DMA for PR

4.1.1 Adding the master-slave DMA engines

First, we implemented the master-slave DMA engines. As shown in figure 1, the master DMA engine resides in the ICAP controller and interfaces with the ICAP FSM, the ICAP, as well as the slave DMA engine. The slave DMA engine resides in the SRAM Controller, and it interfaces with the SRAM Bridge and the master DMA engine. When a DMA operation starts, the master DMA engine receives the starting address as well as the size of the DMA operation. Then it starts sending control signals (read_enable, address *etc.*) to the slave DMA engine, which then forwards the signals to the SRAM Bridge. After the data is fetched, the slave DMA engine sends the data back to the master DMA engine. Then, the master DMA engine decrements the size counter, increments the address, and repeats the process to fetch the next word. Compared to the baseline design, adding the DMA engines avoids the involvement of the pipeline in the data transfer process and it significantly increases the ICAP throughput to about 50 Mbytes/s.

4.1.2 Adding a FIFO between the DMA engines

Second, we modified the master-slave DMA engines and added a FIFO between the two DMA engines. In this version of the design, when DMA operation starts, instead of sending control signals to the slave DMA engine, the master DMA engine forwards the starting address and the size of the DMA operation to the slave DMA engine, then it waits for the data to become available in the FIFO. Once data becomes available in the FIFO, the master DMA engine reads the data and decrements its size counter. When the counter hits zero, the DMA operation completes. On the other side, upon receiving the starting address and size of the DMA operation, the slave DMA engine starts sending controls signals to the SRAM Bridge to fetch data one word at the time. Then once the slave DMA engine receives data from the SRAM Bridge, it writes the word into the FIFO, decrements its size counter, and increments its address register to fetch the next word. In this design, only data is transferred between the master and slave DMA engines and all control operations to SRAM are handled in the slave DMA. This greatly simplifies the handshaking between the ICAP Controller and the

SRAM Controller, and it leads to a 100 Mbytes/s ICAP throughput.

4.1.3 Adding burst mode to provide fully streaming

The SRAM embedded in the ML401 FPGA board actually provides burst read mode such that we can read four words at a time instead of one. Burst mode reads are available on DDR memories as well. There is an ADVLD signal to the SRAM device. During a read, if this signal is set, then a new address is loaded into the device. Otherwise, the device will output a burst of up to four words, one word per cycle. Therefore, if we can set the ADVLD signal every four cycles, each time we increment the address by four words, and given that the synchronization between control signals and data fetches is correct, then we are able to stream data from SRAM to the ICAP.

We implemented two independent state machines in the slave DMA engine. One state machine sends control signals as well as the addresses to the SRAM in a continuous manner, such that in every four cycles, the address is incremented by four words (16 bytes) and sent to the SRAM device. The other state machine simply waits for the data to become ready at the beginning, and then in each cycle it receives one word from the SRAM and streams the word to the FIFO until the DMA operation completes. Similarly, the master DMA engine waits for data to become available in the FIFO, and then in each cycle it reads one word from the FIFO and streams the word to the ICAP until the DMA operation completes. This fully streaming DMA design leads to an ICAP throughput that exceeds 395 Mbytes/s, which is very close to the ideal 400 Mbytes/s number

4.2 Performance of the Streaming DMA Engines

In order to gauge the performance of our fully streaming DMA design, we tested it with five bitstream files with varying sizes and complexities: bitstream file test 1 implements a chain of counters that span one clock region; test 2 is a blanking bitstream for the eMIPS extension; bitstream file test 3 implements a hardware debugger; bitstream file test 4 is similar to test 1 but it spans half the chip; bitstream file test 5 implements the eMIPS design. We measured the time taken to complete the DMA operation and the results are summarized in table 1: in all five cases, our full streaming DMA design achieved an ICAP throughput that was higher than 399 Mbytes/s. This high ICAP throughput would enable the runtime partial reconfiguration technique to become an efficient energy reduction technique.

Table 1: ICAP throughput measurement

Bitfile	size (bytes)	time (seconds)	throughput (Mbytes/s)
TEST 1	83360	0.00020857	399.67
TEST 2	103152	0.00025805	399.74
TEST 3	130816	0.00032721	399.79
TEST 4	512016	0.00128021	399.95
TEST 5	976000	0.00244017	399.97

5. EXPERIMENTS AND RESULTS

We have performed experiments on the Virtex-4 FPGA to study the FPGA energy consumption behaviors. In this section, we introduce our experiment methodology, present our experiment results, and extrapolate the experiment results to show the ability of partial reconfiguration in energy reduction under different situations.

5.1 Methodology

5.1.1 Measurement of chip energy consumption

As shown in figure 2, in order to measure the chip power consumption, we tapped into the 1.2 V core power supply rail on the ML401 board. We placed a 0.033 Ohm shunt resistor on the supply rail and measured the voltage drop across the shunt resistor with a multimeter. Then from the voltage drop we could derive the current that was drawn into the Virtex-4 FPGA chip, and we could multiply this current by 1.2 V to derive the chip core power consumption under different configurations. A separate amperemeter measures the overall power to the entire board. In addition, we utilized the timer embedded in the eMIPS system to measure the program execution time, then multiplied the execution time by the chip core power consumption to get the chip energy consumption.

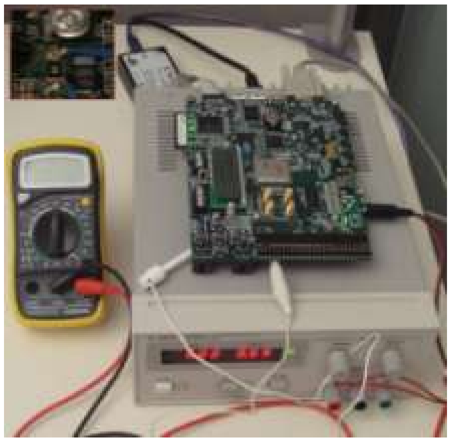

Figure 2: Experiment Setup

5.1.2 Test system: eMIPS

We performed the energy consumption experiments on the eMIPS system. Figure 3 shows the layout of the eMIPS system on the Virtex-4 chip. The system consists of two parts: the highlighted region of the chip hosts a partially reconfigurable extension (in this case a 64-bit division accelerator). Note that the extension spans two clock regions: when using clock gating, we shut down the clock distribution network to both two clock regions. The second part of the design is the static data path, or TISA, it is spread throughout the chip other than the extension region.

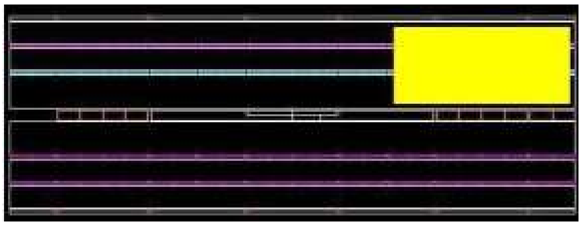

Figure 3: eMIPS layout

Table 2 shows the chip power consumption under different configurations, the first column shows the voltage drop across the shunt resistor on the 1.2 V chip power supply rail, the second column shows the current flowing into the chip, the third column shows chip power consumption. First, during partial reconfiguration (pr), the static TISA is trying to reconfigure the extension, during this time the chip consumes 0.607 W of power. Second, if only the static TISA is active and the extension region is not loaded (no ext), the chip power consumption is 0.604 W. Note that when the extension region is not loaded, it consumes neither static nor dynamic power, and the clock network to the region is shut down. Third, the chip power consumption becomes 0.611 W if the extension is loaded and we use clock gating to shut down the clock network in the extension (ext+cg). In this case, the extension consumes only static power. Finally, if the extension is loaded and active (ext), the chip power consumption is 0.625 W. Therefore, the total power consumption of the extension is 0.021 W: 0.014 W is dynamic/clock power, and the rest 0.007 W is static power.

Table 2: chip power consumption

	volt (mV)	current (A)	power (W)
Pr	16.7	0.506	0.607
no ext	16.6	0.503	0.604
ext+cg	16.8	0.509	0.611
ext	17.2	0.521	0.625

5.2 Basic Properties of the Virtex-4 FPGAs

In this subsection, we study the basic properties of the Virtex-4 FPGA systems. It is essential that we understand these basic properties before going into the core experiments.

5.2.1 Configuration time is a pure function of the bitstream file size

Theoretically, the ICAP throughput can reach 400 Mbytes/s, but this is achievable only if the configuration time is a pure function of bitstream file size. In order to find out whether this theoretical throughput is achievable, we performed experiments to configure different regions of the chip, to repeatedly writing NOPs to ICAP, and to stress the configuration circuit by repeatedly configuring one region. During all these tests, we found out that ICAP always ran at full speed such that it was able to consume four bytes of configuration data per cycle, regardless of the semantics of the configuration data. This confirms that configuration time is a pure function of the size of the bitstream file.

5.2.2 Partial reconfiguration reduces static power

Recall that the study in [3] indicated that the polarity of the inputs of outputs in FPGA hardware structures may significantly impact leakage power consumption. We would like to find out whether the Virtex-4 design utilizes this property such that when the *blank* bitstream is loaded to wipe out an accelerator, the circuit is set to a state to minimize the leakage power consumption.

In order to achieve this, we implemented eight partial reconfigurable (PR) regions on the Virtex-4 chip, with each region occupying a configuration frame. These eight PR regions did not consume any dynamic power, as we purposely gated off the clock to these regions. Then we used the *blank* bitstream files to wipe out each of these regions and observed the chip power consumption behavior. The results indicated that for every four configuration frames that we applied the *blank* bitstream on, the chip power consumption dropped by 0.01 W. This study confirms that partial reconfiguration indeed leads to static power reduction and suggests that the Virtex-4 design may have utilized the polarity property to minimize leakage power.

5.3 Energy Reduction by Acceleration Extension

In this subsection, we try to answer the question of whether partially reconfigurable acceleration hardware leads to energy reduction. In the experiment, we compare the energy consumption of using only the baseline TISA to that of using the baseline TISA plus the acceleration extension. The extension is always loaded regardless of whether it is active or not. The test program had two parts: the first part consisted of only 64-bit division operations, and the second part consisted of only 32-bit computations. We fixed the workload in the 32-bit computation part and gradually increased the workload in the 64-bit division part. To give a reference for performance comparison, the 32-bit computation part took 6.7 seconds to finish. For the 64-bit division, it took the baseline version about 1 second to complete 1000 operations, whereas with the extension loaded, this time dropped to around 0.5 second.

The results are summarized in figure 4: the x-axis represents the 64-bit division workload, whereas the y-axis shows the chip energy consumption. It shows that the acceleration extension almost always leads to energy reduction due to its low power consumption and its ability to significantly reduce execution time. As more 64-bit division operations are required in the program, the energy consumption gap between these two configurations also widens. When there are 50,000 64-bit divisions, this part consists of 90% of the total program workload, and the chip energy consumption is almost halved by using the accelerator.

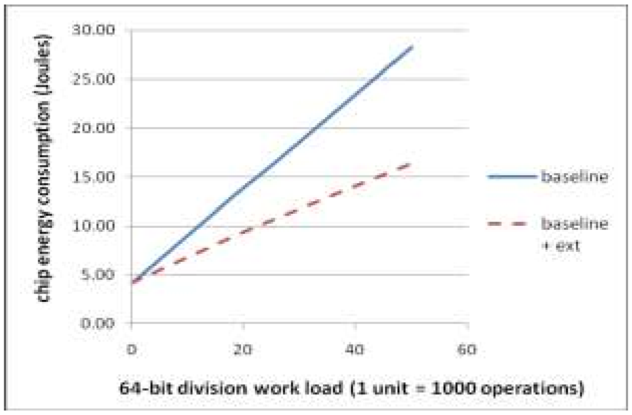

Figure 4: energy efficiency of acceleration extensions

In figure 4, each increment of 64-bit division workload indicates 1000 operations, which roughly equals to 1/7 of the 32-bit operation workload. Even if the 64-division workload is only 1000 operations, we can still observe energy reduction by using the accelerator. To find out when the accelerator starts to reduce energy, we performed another experiment and mapped the analytical model from subsection 3.1 to these results. As shown in table 3, the first column shows the number of 64-bit division operations; the second column shows the program speed-up (SU), or the baseline execution time over the execution time of the baseline with the accelerator; the third column shows the power-up (PU), or the chip power consumption with accelerator loaded over that of baseline only; the fourth column is the ratio of power-up over speed-up; finally, the fifth column shows the energy consumption delta, or the energy consumption of the baseline minus the energy consumption with the accelerator loaded. A positive number on the fifth column indicates that the accelerator led to energy reduction.

Note that in table 3 PU is a constant number whereas SU increases as the amount of 64-bit division workload increases. As predicted by equation 4, when the ratio PU/SU is less than 1, then using the accelerator leads to energy reduction. This prediction is verified by the experimental data in table 3. Therefore, we conclude that the ability of the acceleration extension to reduce energy consumption depends on its ability to accelerate the program execution, the fraction of the program that can be accelerated, as well as how much extra power it consumes compared to the baseline. The first two factors contribute to SU, whereas the last factor contributes to PU. In the eMIPS design, due to it is low power-up (1.050), the 64-bit division acceleration extension can lead to energy reduction even if the 64-bit division workload is as low as 8% of the total program workload. Thus, the acceleration extensions not only accelerate program execution but reduce energy consumption as well.

Table 3: energy reduction with acceleration extensions

div64_op	SU	PU	PU/SU	E delta (J)
0	1.000	1.050	1.050	-0.172
1	1.000	1.050	1.050	-0.172
10	1.001	1.050	1.049	-0.170
100	1.006	1.050	1.044	-0.148
1000	1.053	1.050	0.997	0.046
10000	1.352	1.050	0.777	2.026

5.4 Runtime Partial Reconfiguration for Energy Reduction

In this subsection, we study two techniques, partial reconfiguration (PR) and clock gating (CG), to further reduce energy consumption. We compare these two techniques and we try to identify the best conditions for each technique. Recall that CG shuts down the clock network in the accelerator and leads to elimination of dynamic power; since CG takes only two cycles to complete, it introduces negligible overhead. On the other hand, PR reduces both static and dynamic power by reconfiguring the accelerator but it may introduce significant energy overhead due to the comparatively long reconfiguration time.

For this study, we consider two extreme types of program behaviors. These two types of program behaviors have no impact on the CG technique but a large impact on the PR technique. As shown in the upper part of figure 5, all 64-bit division operations are concentrated into one portion of the program, thus with PR, the accelerator only has to be loaded once to accelerate the program, then unloaded once to reduce power. On the other hand, as shown in the lower part of figure 5, the 64-bit division operations and the normal operations appear in an interleaving fashion. In this case, with PR, before each chunk of the 64-bit division operations starts, the accelerator have to be loaded; after the chunk of 64-bit division operations completes, the accelerator have to be unloaded to make room for other application. A typical program behavior will fall somewhere in between these two extremes.

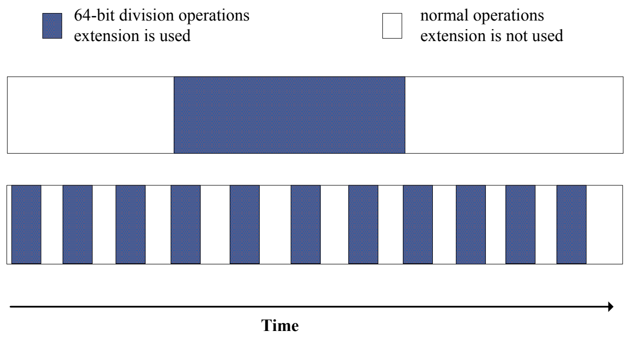
64-bit division operations extension is used
normal operations extension is not used
Time

Figure 5: program behaviors

In the first experiment, we used a program with the first type of behavior, namely, all 64-bit division operations were concentrated in only one part of the program. We fixed the amount of 64-bit division to 100 operations, which took the design with accelerator about 0.05 seconds to complete. Then we increased the amount of workload that did not require the acceleration extension; while executing this workload, the accelerator is inactive under both CG and PR schemes. We compared the energy consumption of four different schemes: PR with 318 Kbyte/s ICAP throughput, PR with 50 Mbytes/s ICAP throughput, PR with 400 Mbytes/s ICAP throughput, and CG.

The results are summarized in figure 6: the x-axis represents the workload that does not require the accelerator; each unit represents about 2.5 second of execution time. The y-axis represents the energy reduction incurred by the selected scheme: it is derived by subtracting the energy consumption of the selected scheme from the scheme that leaves the accelerator active throughput program execution.

The first observation from figure 6 is that if the ICAP throughput is 318 Kbyte/s, then it is not worthwhile to use PR because it consumes more energy than just leaving the accelerator on the whole time. This is because the low ICAP throughput results in a very high energy overhead during reconfiguration. In this particular case, the size of the bitstream files was 190 Kbytes, and it took about 0.5 seconds to complete the configuration process, during which 0.35 J of energy was dissipated. The second observation is that as the non-extension workload increases, the energy reduction brought by both PR and CG techniques increases. Also, the high-throughput PR techniques resulted in more energy reduction than CG because PR techniques reduced both static and dynamic power. The third observation is that there is little difference between the scheme of 50 Mbytes/s ICAP and that of 400 Mbytes/s ICAP. This is because the extension was only configured twice (to load and unload the extension), thus in the first type of program, configuration time was negligible with either high throughput ICAP DMA.

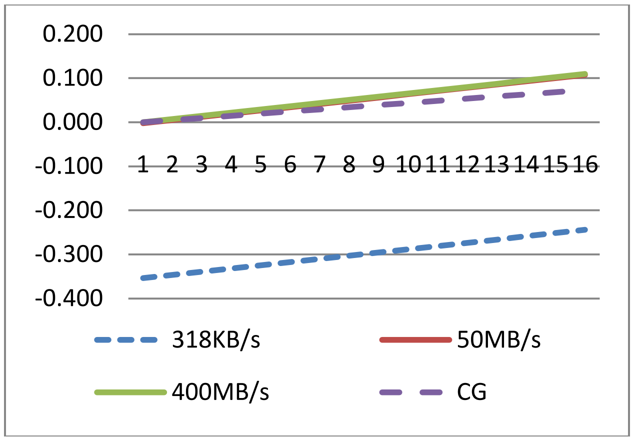

Figure 6: energy reduction by PR

In the second experiment, we used a program with the second type of behavior, namely, the 64-bit division operations interleave with the non-extension operations, such that with PR the accelerator needs to be loaded and unloaded many times during execution. This program contained a 100-iteration main loop, in each iteration we fixed the amount of 64-bit division to 10 operations and increased the amount of non-extension workload. In this case, we only compared the energy consumption of three schemes: PR with 50 Mbytes/s ICAP throughput, PR with 400 Mbytes/s ICAP throughput, and CG.

The results are summarized in table 4: the first column shows the time during which the accelerator is not active in each iteration, the number is shown in scientific notation; the second, third, and fourth columns (PR50, PR400, CG) show the energy reduction produced by the PR scheme with 50 MB/s and with 400 MB/s ICAP throughput, and by the CG scheme respectively. The fifth and sixth columns show the energy reduction differences between the PR and CG schemes. A positive number in the fifth or sixth columns shows the advantage of PR over CG in terms of energy reduction.

The first observation is that due to its negligible overhead, CG almost always results in energy reduction compared to leaving the accelerator active the whole time. The second observation is that the PR schemes produce energy reduction even when the granularity of non-extension work in each iteration is at the millisecond range. Particularly, a higher ICAP throughput enhances the ability of the PR technique to produce energy reduction. In general, the PR techniques have an advantage over the CG techniques in terms of energy reduction as the non-extension work load increases, this trend has been observed in figure 6 and table 4.

Table 4: energy reduction by CG and PR

t_inactive (s)	PR50 (J)	PR400 (J)	CG (J)	PR50 - CG (J)	PR400 - CG (J)
5.00E-05	-0.218	-0.028	0.000	-0.218	-0.028
5.00E-04	-0.216	-0.026	0.001	-0.217	-0.027
5.00E-03	-0.202	-0.012	0.010	-0.213	-0.023
5.00E-02	-0.062	0.128	0.104	-0.166	0.024
5.00E-01	1.340	1.530	1.038	0.302	0.491
5.00E+00	15.352	15.542	10.380	4.972	5.162

In addition, by using the analytical model presented equations 8 and 12 on the eMIPS design, we derived that with a 400 Mbytes/s throughput ICAP, PR would bring energy reduction compared to leaving the accelerator active all the time if the execution time of the non-extension workload in each iteration exceeds 9.1 milliseconds; and PR would bring energy reduction compared to CG if the execution of the non-extension workload in each iteration exceeds 27 milliseconds, and we verified this experimentally. If the ICAP throughput is 50 Mbytes/s, these two numbers become 72.8 milliseconds and 216 milliseconds. Furthermore, if the ICAP throughput is only 318 Kbytes/s, these two numbers become 11.7 seconds and 34.8 seconds.

To conclude, the conventional wisdom is that clock gating is an effective fine-grained energy reduction technique, whereas partial reconfiguration should not be used to reduce energy consumption unless the accelerator would be inactive for a very long period of time. Our study has showed that the ICAP throughput is critical to the performance of PR as an energy reduction technique: a high ICAP throughput enables PR to reduce energy consumption even if the execution time of the non-extension workload is in the range of milliseconds.

5.5 Extrapolation

In the eMIPS design, the power consumption of the 64-bit division acceleration extension is only 5% of the power consumption of other parts of the system. We are interested in finding out the effectiveness of partial reconfiguration in reducing energy consumption when the accelerator consumes a more significant part of the chip power. To achieve this, we extrapolated the experiment results shown in figure 4 by assuming that the power consumption of the accelerator was 50% of that of other parts of the system. Note that in this case, all other conditions remain the same as the original results. The extrapolated results are summarized in figure 7. It shows that when the 64-bit division workload is small, the energy consumption of the design with accelerator is much higher than that of the baseline, implying that a high energy overhead as a result of program acceleration. However, with CG and PR (400 Mbytes/s ICAP throughput), we are able to accelerate program execution and at the same time reduce energy dissipation. Furthermore, since the energy consumption of the accelerator is a significant part of the chip energy consumption, PR is able to reduce a significantly higher amount of energy compared to CG.

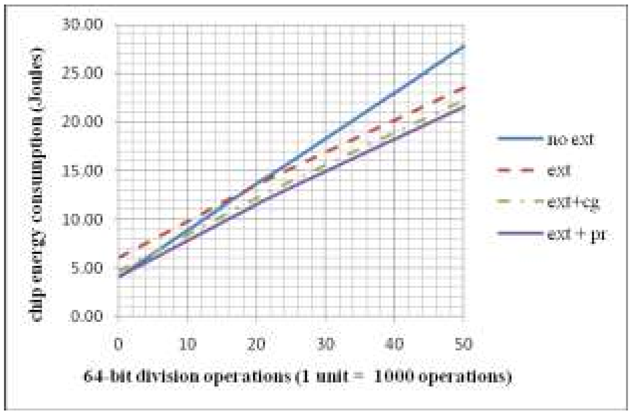

Figure 7: energy reduction by PR

6. CONCLUSIONS

In this paper, we studied whether partial reconfiguration can be used as an effective energy reduction technique in reconfigurable computing systems. To approach this problem, we first identified the analytical models that capture the necessary conditions for energy reduction under different system configurations. The models show that increasing the configuration throughput is a general and effective way to minimize the partial reconfiguration energy overhead. Therefore, we designed and implemented a fully streaming DMA engine that nearly saturates the configuration throughput.

The findings of this paper provide answers to the three questions raised in the introduction: first, although we pay extra power to use an accelerator, depending on the accelerator's ability to accelerate the program execution, it will result in actual energy reduction. The necessary condition for energy reduction is given in equation 4. The experimental results on eMIPS demonstrate that due to its low power overhead and excellent ability to accelerate 64-bit division operations, having the 64-bit division acceleration extension can lead to both program speedup and system energy reduction.

Second, it is worthwhile to use partial reconfiguration to reduce chip energy consumption if the energy reduction can make up for the energy overhead incurred during the reconfiguration process; and the key to minimize the energy overhead during the reconfiguration process is to maximize the configuration speed. The minimum configuration throughput required for partial reconfiguration to produce energy reduction is modeled in equation 8. Our experimental results demonstrate that our fully streaming DMA design is able to achieve >399 Mbytes/s configuration throughput. With our DMA design, the configuration process takes less than one millisecond instead of a fraction of a second. This short configuration time makes the configuration energy overhead negligible, and thus enabling partial reconfiguration to be a highly effective energy reduction technique.

Finally, clock gating is an effective technique in reducing energy consumption due to its negligible overhead; however it reduces only dynamic power whereas partial reconfiguration reduces both dynamic and static power. Therefore, partial reconfiguration can lead to a larger energy reduction than clock gating, provided the extra energy saving on static power elimination can make up for the energy overhead incurred during the reconfiguration process. The minimum configuration throughput required for energy reduction is defined in equation 12. Although the conventional wisdom is that partial reconfiguration is only useful if the accelerator would not be used for a very long period of time, the experimental results indicate that with the high configuration throughput delivered by our fully streaming DMA engine, partial reconfiguration can outperform clock gating in energy reduction even if the accelerator inactive time is in the millisecond range.

The results of the extrapolation study indicate that partial reconfiguration is even more beneficial when the accelerator consumes a significant fraction of the system power. In this situation, when the accelerator is loaded, it accelerates program execution but meanwhile it increases system energy consumption. By using partial reconfiguration to reduce the power consumption of the accelerator when it is inactive, we can accelerate program execution and at the same time reduce the overall energy consumption by half.

REFERENCES

1. A. Telikepalli, "power vs. performance: the 90nm inflection point," Xilinx White Paper 223.
2. T. Tuan and B. Lai, "leakage power analysis of a 90nm FPGA," in proceedings of IEEE Custom Integrated Circuits Conference, 2003.
3. J.H. Anderson and F.N. Najm, "active leakage power optimization for FPGAs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, No. 3, March 2006.
4. Q. Wang, S. Gupta, and J. Anderson, " clock power reduction for Virtex-5 FPGAs," in proceedings of ACM/SIGDA International Symposium on FPGAs, 2009.
5. K. Paulsson, M. Hubner, S. Bayar, and J. Becker, "exploitation of run-time partial reconfiguration for dynamic power management in Xilinx Spartan III-based systems," in proceedings of International Conference on High-Performance Embedded Architectures and Compilers, 2008.
6. A. Gayasen, Y. Tsai, N. Vijaykrishnan, M Kandemir, M.J. Irwin, and T. Tuan, "reducing leakage energy in FPGAs using region-constrained placement," in proceedings of ACM/SIGDA International Symposium on FPGAs, 2004.
7. R.P. Bharadwaj, R. Konar, D. Bhatia, and P. Balsara, "FPGA architecture for standby power management," in proceedings of IEEE International Conference on Field-Programmable Technology, 2005.
8. M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "run-time partial reconfiguration speed investigation and architectural design space exploration," in proceedings of IEEE International Conference on Field Programmable Logic and Applications, 2009.
9. C. Claus, B Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "a multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in proceedings of IEEE International Conference on Field Programmable Logic and Applications, 2008.
10. Virtex-4 FPGA User Guide: http://www.xilinx.com/support/documentation/user_guides/ug070.pdf
11. The eMIPS Project: http://research.microsoft.com/en-us/projects/emips/default.aspx

12. D.B. Thomas and W. Luk, “multivariate gaussian random number generation targeting reconfigurable hardware,” ACM Transactions on Reconfigurable Technology and Systems, vol. 1, no. 2, June 2008.

13. T. Fry, S. Hauck, "SPIHT Image Compression on FPGAs", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, No. 9, pp. 1138-1147, September 2005.

APPENDIX 1: LOW-LEVEL PARTIAL RECONFIGURATION AND POWER STUDIES

In ***ICAP_NOP***:

Place a BRAM next to the ICAP Controller to repeatedly write different commands to the ICAP and observe the behavior of the ICAP port.

In ***POWER_STUDY***:

Implement a set of counters on the FPGA chip and observe how the power consumption changes as the number of counters increases.

In ***PR_STUDY_MULTIPR***:

Based on ICAP_NOP, build a BRAM next to the ICAP controller to store bitstream files. Build multiple PR regions and repeated configure these PR regions and observe the effect on the performance of the ICAP.

Power and Configuration Experiments

<u>P1: Measure changes in power due to device utilization.</u>

Description: Build bit streams containing counters of varying widths (0,1,2…1024) and measure the power consumed by the FPGA under each bit stream.

Claims: Bit steams containing more counters should consume more power than those with less. If the change in power is significant, power savings may be achieved by compacting area of designs and deactivating underutilized areas when not in use.

Results:

Note: the following results are obtained when the clock is running at 100 MHz								
		Board-Level Measurement			Chip-Level Measurement			
number of counters	voltage (v)	current (A)	Board Power (W)	Chip Power (W)	shunt (mV)	core power (W)		
0	5	0.54	2.7	0	5.4	0.20	//only control	
1	5	0.57	2.85	0.15	5.8	0.21		
2	5	0.57	2.85	0.15	5.9	0.21		
4	5	0.57	2.85	0.15	5.9	0.21		
8	5	0.57	2.85	0.15	6.2	0.23		
16	5	0.57	2.85	0.15	6.3	0.23		
32	5	0.58	2.9	0.2	6.6	0.24		
64	5	0.58	2.9	0.2	6.8	0.25		
128	5	0.58	2.9	0.2	7.2	0.26		

256	5	0.59	2.95	0.25	8.3	0.30		
512	5	0.6	3	0.3	9	0.33		
1024	5	0.62	3.1	0.4	11.4	0.41		

When we get to 1024 counters, the chip is almost filled			
logic	utilization		
FF	76%		
LUT	9%		
logic distribution			
slices	81%		

Note: measurement with multimeter = > eMIPS chip power			
eMIPS version 1.1 with mmldiv64 extension			
Configuration	Volt drop (mV)	Current (A)	Power (W)
full-config	15.6	0.47	0.57
blank bitstream	15.2	0.46	0.55
PR bitstream	15.6	0.47	0.57

P2: Measure changes in power due to activity.

Description: Build a bit stream of a design that would allow us to dynamically change the clock frequency driving a 1024 bit counter in an FPGA. Measure the power consumed as the frequency is changed.

Claims: Running the FPGA at a higher frequency (ie at a higher activity level) consumes more power than with less activity. If the change in power is significant, power savings may be achieved by reducing the overall activity level of the FPGA by turn off activity to regions that are not currently in use through clock gating, reconfiguration or some other method.

Results:

Note: measurement with multimeter = > chip power			
1.2V Chip Core Power Supply R182: 0.033 R			
Experiment 1: with 1024 counters running at different freqs			
freq (MHZ)	Volt drop (mV)	Current (A)	Power (W)
100	15.5	0.47	0.56
200	22	0.67	0.80
300	28.3	0.86	1.03
400	34.5	1.05	1.25

P3: Measure changes in power using modular partial reconfiguration

Description: Build a partially reconfigurable project using an FPGA with multiple partially reconfigurable regions containing counters. Measure the power consumed with all the pr regions containing counters and as pre regions are disabled using blanking bit streams. Sweep clock frequency from 100 to 400 MHz to reflect different levels of activity.

Claims: If the power consumption is reduced by disabling regions of the FPGA using the blanking pr bit streams, it might be possible to us pr to eliminate the power consumed by certain regions of the chip at times when it is not needed.

Results:

Note: measurement with multimeter = > chip power				
5 modules each with 100 counters: can use this for extrapolation				
voltage drop across the shunt at different freq and different configuration				
	100 MHz (mV)	200 MHz (mV)	300 MHz (mV)	400 MHz (mV)
full	11.3	14.5	16.9	21
minus ext0	10.6	13.2	15	18.4
minus ext1	10.1	11.9	13.1	15.9
minus ext2	9.5	10.6	11.7	13.4
minus ext3	9	9.6	10.2	11.1
minus ext4	8.4	8.6	8.8	9
chip core power consumption at different freq and different configuration				
	100 MHz (W)	200 MHz (W)	300 MHz (W)	400 MHz (W)
full	0.41	0.53	0.61	0.76
minus ext0	0.39	0.48	0.55	0.67
minus ext1	0.37	0.43	0.48	0.58
minus ext2	0.35	0.39	0.43	0.49
minus ext3	0.33	0.35	0.37	0.40
minus ext4	0.31	0.31	0.32	0.33

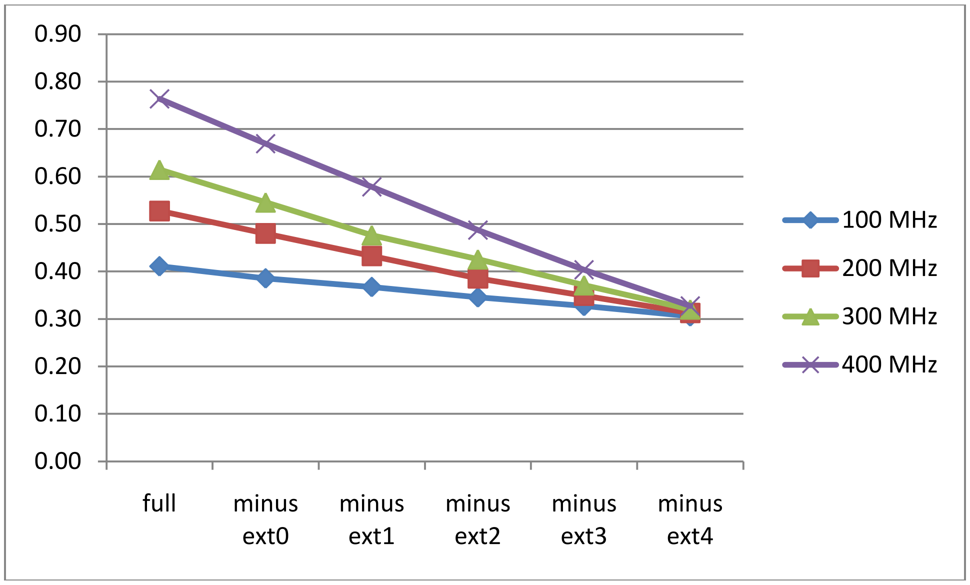

Figure 1: changes in power at various configuration and frequencies (y-axis: power in Watts)

P4: Power consumed by Configuration

Description: Perform configuration using the ICAP using different bit streams including manufactured bit streams containing regular patterns of bits. Measure power consumed during configuration operation and compare relative to power consumed before and after configuration operation.
Claim: This is to identify any potential power overhead configuration may incur in a system that performs reconfiguration regularly and frequently.
Results:

1. During full-chip configuration---voltage drop across shunt 1.6 mV => 0.06 W. I have tested this with various configuration bitstream files (including eMIPS, power_counter, etc.)
2. Put partial bitstream in BRAM and continuously perform reconfiguration, difference of voltage drop across shunt before and during the configuration process 0 mV

C1: Partial Reconfiguration with ICAP

Description: This experiment provides a starting testing platform for further experimentation. Using the ICAP controlled by a FSM, we use partial reconfiguration to modify a region of the FPGA using a partial bit stream stored in Block ram on chip.
Claims: These experimental setups removes all external factors that could impact the performance of the configuration process and allow us to focus on the hardware mechanics of the configuration interface itself. This work also means to verify performance numbers reported by previous work.
Results:

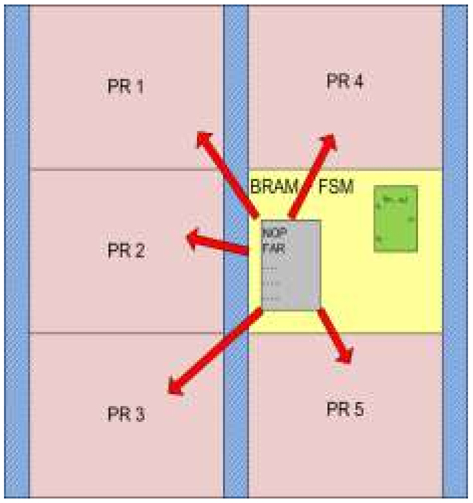

Figure 3: experiment setup of ICAP PR study

In this study, we have implemented a BRAM next to the ICAP. The BRAM is used to store the bitstream for partial reconfiguration. We have also implemented a FSM such that once it receives a command from the serial interface, it starts writing the bitstream to ICAP. When it finishes, it returns the number of cycles taken to write the bitstream to ICAP, as well as the number of cycles ICAP_BUSY signal is set.

The result is that ICAP ***always runs at full speed***: it is able to write one word (4 bytes) to ICAP at each cycle (at 100 MHz), the ICAP_BUSY signal is set for one cycle at the end of the bitstream writing.

The conclusion is that ICAP is able to support the maximum throughput 400 MB/s, the bottleneck of fast configuration is actually at how fast we can fetch the configuration data from memory. In our next stage design, we plan to implement a DMA engine to establish direct communication between the ICAP and the configuration memory. The implementation quality of DMA will decide how fast we can do configuration.

C2: Bit stream manipulation: Configuration CRC test.

Description: Change CRC value of a bit stream to determine behavior of device in the case of a CRC error.

Claims: Previous work reflected ambivalence to the value of the CRC when configuring the FPGA. This experiment is meant to concretely verify this observation. If the correctness of the CRC has no effect on the behavior of the device it becomes possible for us to manufacture bit streams for experimentation without having to know how to generate correct CRC values.

Results:

1. Modify the CRC word (using a wrong CRC value): once a CRC error is detected, ICAP stops functioning and stops taking new commands.
2. Remove the CRC command: ICAP still functions, and the PR bitstream gets written to ICAP as usual. After the bitstream is written to ICAP, the new PR region starts functioning.
3. Disable CRC using COR: in this case, the CRC checking function is disabled, an attempt to write any CRC value to ICAP results in ICAP stop functioning.

Conclusion: ***CRC is optional***. We do not have to put the check CRC command (30000001) in the bitstream. But if we put that into the bitstream and the CRC value is incorrect, then ICAP stops functioning.

<u>C3: Bit stream manipulation: Remove NOPs</u>

Description: The FPGA tools insert many NOPs throughout the bit stream. We would like to understand if these are necessary and if so why. To begin to understand this we will remove the NOPs from the bit stream and attempt to configure the FPGA with a modified bit stream.
Claims: If the modified bit stream works as before, the NOPs are indeed that and can be removed to improve performance. If not, the configuration mechanism has requires them to provide delay.
Results:

1. I remove all NOPs from the bitstream, after writing the bitstream to ICAP, the PR region is not functioning.
2. If a NOP is spotted in a bitstream, I replace it with a meaningless word (e.g. 00000000). After doing this, the PR region still functions.
3. If a NOP is spotted in a bitstream, I stall ICAP for several cycles and jump to the next command. In this way, we do not write NOP to ICAP but instead give several cycles of delay to ICAP. After doing this, the PR region still functions.

Conclusion: ***NOP is a dummy command to add delays*** to the ICAP to allow time for the commands to finish.

Situations where NOPs are inserted:

1. After each write to CMD "30008001" has finished
2. After each write to FAR "30002001" has finished
3. If 30002001 (write to FAR) is followed by 30008001 (write to CMD), then NO NOP is inserted after 30002001.
4. If 30002001 (write to FAR) is followed by 30014XXX (MFWR), then no no-op is inserted after 30002001.

<u>C4: Bit stream manipulation: Determine latency of common commands (ICAP_BUSY)</u>

Description: The ICAP interface is made of several command words used to control the state of the configuration fabric. These commands are stored with the configuration data in the bit stream. In the ICAP interface there is output called BUSY. Without any documentation, we assume this indicates that the ICAP is performing some function and cannot take commands or data at that time. We will write various patterns of commands and data to an attempt to force the ICAP to raise this BUSY signal.
Claims: If the ICAP can be busy in this way, under circumstances will this occur? Is it possible for some sequences of commands to raise this BUSY and thus require a halt to the streaming of configuration data to the ICAP. If this is so it would possible to explaining why in some cases it appears that theoretical maximum bandwidth cannot be achieved.
Results:

1. Issue 20 registers reads --- ICAP_BUSY never set

2. Issue a read immediate after a write to the same register, and repeat this 20 times ---- ICAP_BUSY never set
3. Select several frame addresses within the region and write data frames to them in pseudo random pattern—ICAP_BUSY never set.
4. Write the same partial bit stream to the ICAP over and over—ICAP_BUSY never set.
5. Write nothing but NOPs to ICAP-- ICAP_BUSY never set.
6. Repeat set frame address followed by same address—ICAP_BUSY never set.
7. Repeat set frame address followed by different address in a pattern—ICAP_BUSY never set.
8. Select a data frame from one of your bitstream and write that repeatedly—ICAP_BUSY never set.
9. Take a working bitstream and replace all the frame data with zeros—ICAP_BUSY never set.
10. ICAP_BUSY is set when I turn off the write_enable signal and turn it back on, it causes 6 cycles of ICAP_BUSY.

Conclusion: ***ICAP_BUSY is only set when we manipulate the write_enable signal***.

C5: Configuration Test: Repeat Re-Configuration
Description: Repeatedly write the same bit stream to configure the same region over and over. Measure the number cycles between last byte written and rising of DONE signal and the number of cycles between the last byte and the first byte of the next pass.
Claims: If the time between the end and start of configurations is a significant overhead, then configuring multiple regions or changing the configuration of a region frequently would have to overcome that cost in performance acceleration.
Results: as shown in experiment C4, ICAP_BUSY is never set during reconfiguration regardless of what bitstream is used. Thus, configuration should always be able to run at full speed.

C6: Configuration Test: Re-Configure multiple Regions
Description: Repeatedly write different bit streams sequentially in a pseudo random pattern. Measure the number cycles between last byte written and rising of DONE signal and the number of cycles between the last byte of one configuration and the first byte of the next.
Claims: This is to simulate the activity of an OS scheduler reallocating the available pr regions to better match the current work load. If the time between the end and start of configurations is a significant overhead, then configuring multiple regions or changing the configuration of a region frequently would have to overcome that cost in performance acceleration.

Results: as shown in experiment C4, ICAP_BUSY is never set during reconfiguration regardless of what bitstream is used. Thus, configuration should always be able to run at full speed.

C7: Bit stream manipulation: Don't Write NOPs

Description: Write a partial bit stream for a pr region to the ICAP from a BlockRAM. If the word read from the BlockRAM is a NOP, don't write to the ICAP. Otherwise write it to the ICAP. This will all the cycle of delay that NOP allows without actually writing the NOP.
Claims: By allowing the delay and not writing the NOP we can determine if inclusion of the NOPs are only to provide statically scheduled flow control in the form of delays from the NOPs instead to some other underlying mechanism such as pipelining or clocking the ICAP using the write enable. Given the number of NOPs in the bit steams, a redefinition of the bandwidth metric should be considered from the number of bytes per second to configuration data or command byte per second (exclude NOPs) or chip area configured per second.
Results: as shown in C3, ***NOP is a dummy command to add delays*** to the ICAP to allow time for the commands to finish.

C8: blanking bitstream

Description: to study what a blanking bitstream does. A blanking bitstream is supposed to wipe out an extension, but what exactly does it do? To find out, we study the details of the blanking bitstream files.

Results:
Blank bitfile basically keeps writing 0's to the unused PR region

APPENDIX 2: Power Consumption Experiments

The eMIPS System Power Consumption Based on XPOWER Simulations

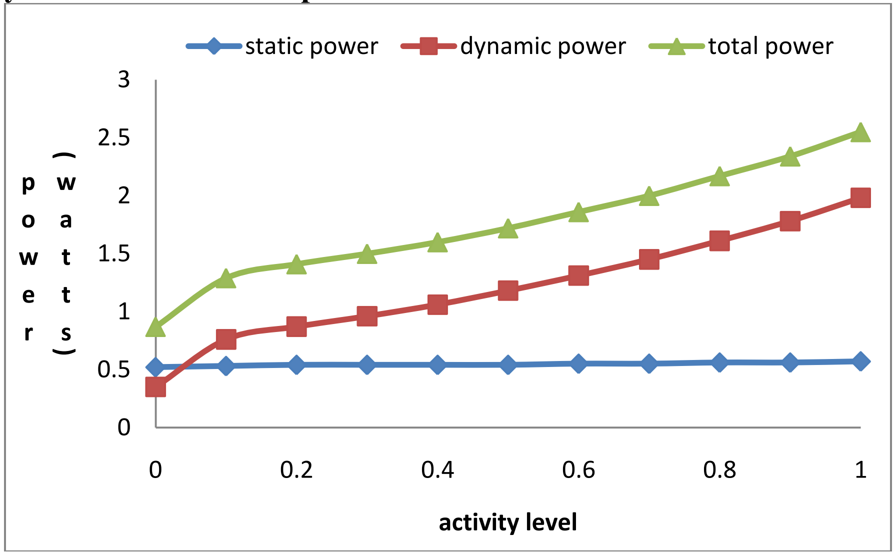

Figure 1: total power consumption as a function of activity level

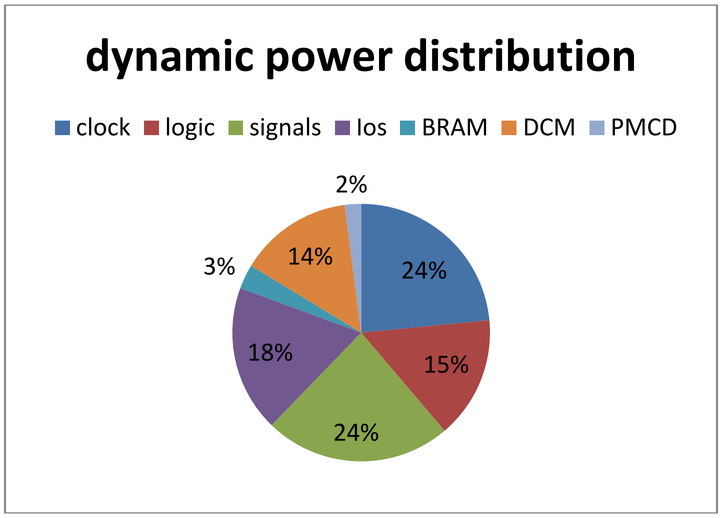

Figure 2: component-based dynamic power distribution

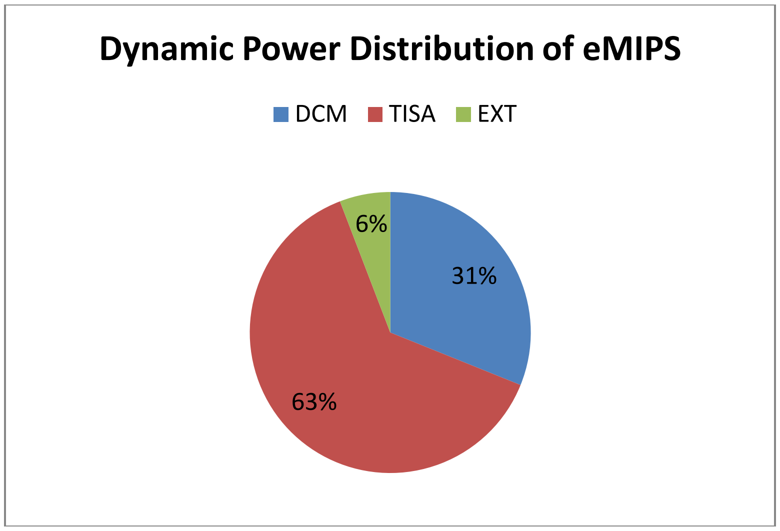

Figure 3: module-based dynamic power distribution

Chip Power Consumption Measurement Setup

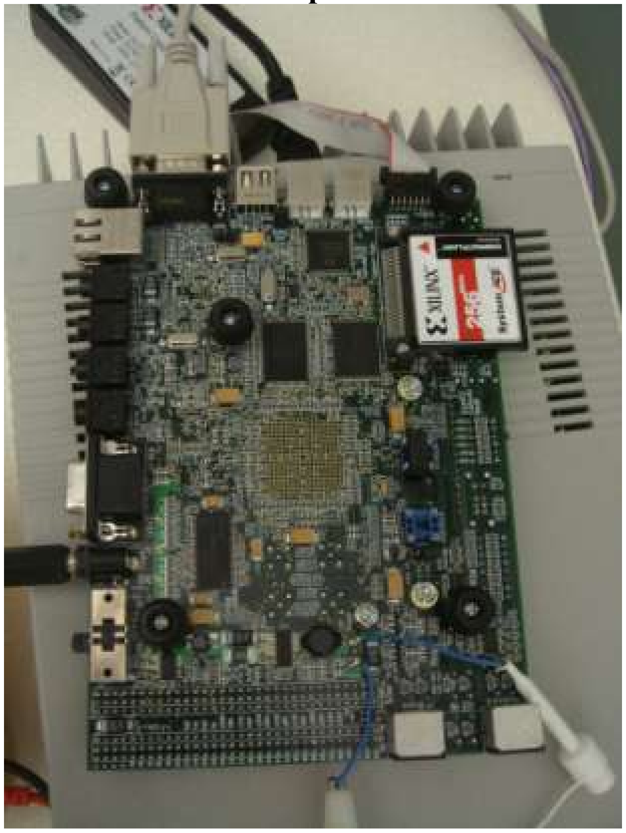

Figure 4: place 0.033 R shunt resistor on the 1.2 V chip power supply

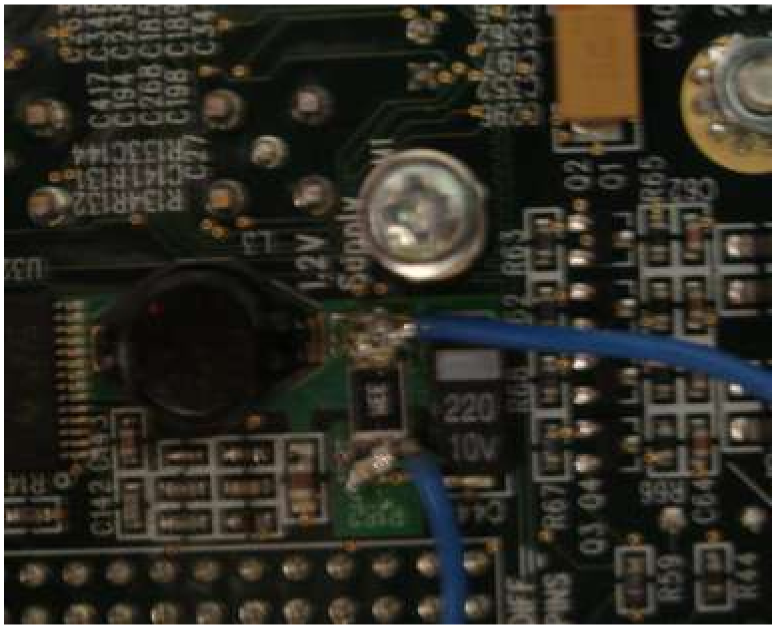
Figure 5: close view of the shunt resistor

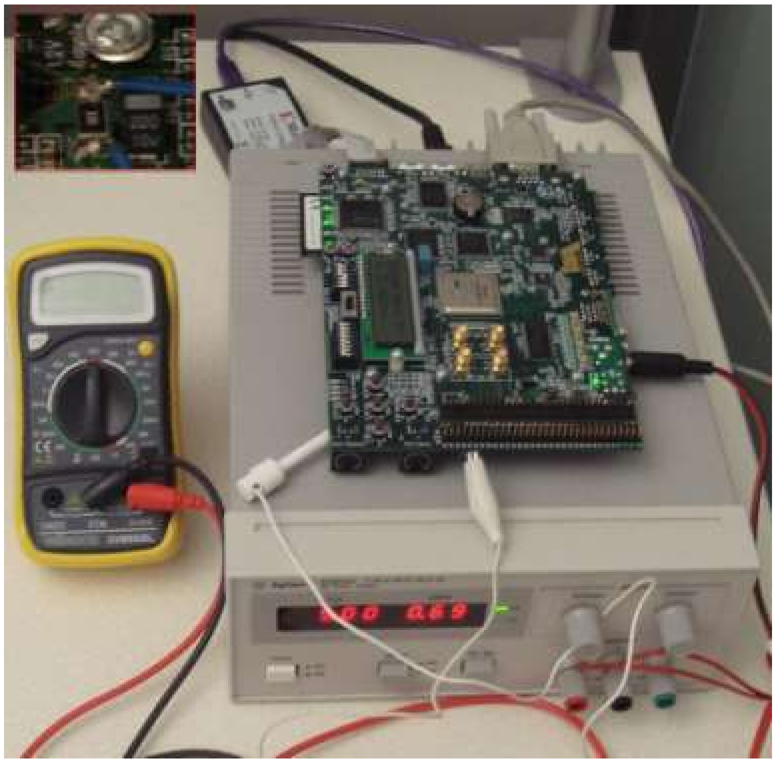
Figure 6: measurement equipments