An Automated and Power-Aware Framework for Utilization of IP Cores in Hardware Generated from C Descriptions

Alex Jones and Prith Banerjee

Technical Report No. CPDC-TR-2002-04-02
©2002 Center for Parallel and Distributed Computing
April 2002

Center for Parallel and Distributed Computing
Department of Electrical & Computer Engineering
Northwestern University
Technological Institute
2145 Sheridan Road, Evanston IL-60208

An Automated and Power-Aware Framework for Utilization of IP Cores in Hardware Generated from C Descriptions

Alex Jones Prith Banerjee
Center for Parallel and Distributed Computing
Department of Electrical and Computer Engineering
Technological Institute, Northwestern University
2145 Sheridan Road, Evanston, IL 60208-3118
Phone: (847) 491-3641 Fax: (847) 491-4455
Email: {akjones, banerjee}@ece.northwestern.edu

ABSTRACT
Use of hand optimized Intellectual Property (IP) logic cores is prolific in hardware design. These IP cores range from rather complicated signal processing transforms and filters to arithmetic operators. While IP cores remain a standard way to utilize the improvement in chip fabrication technology and contend with time to market pressure through reuse, popularity of tools generating hardware descriptions from high level languages is increasing in popularity. PACT HDL attempts to combine these two methods within a power-aware framework. The PACT HDL compiler generates power optimized VHDL/Verilog from a C language description. The work presented here describes an automated framework to incorporate arbitrary IP logic cores corresponding to C intrinsic operators. The logic cores to be used may be specified by the user through compiler directives or compiler command line options. The framework is power-aware through using a clock-gating technique to "turn off" the IP cores when not in use. The validity of the approach is demonstrated using several image and signal processing benchmarks.

Keywords
High-level synthesis, C/C++, VHDL/Verilog, power-aware, signal and image processing, FPGA, ASIC, SoC

1. INTRODUCTION
Recently, a lot of attention has been placed on automated generation of hardware descriptions of based on algorithms written in high-level languages such as C/C++. There are several reasons for this trend: 1) It is much easier to find programmers who are knowledgeable about C/C++ compared to programmers knowledgeable about industry standard hardware design languages (HDLs) such as VHDL and Verilog. 2) Many computationally complex algorithms that would benefit from hardware implementations are available or being developed in C/C++. 3) Fabrication technology continues to improve, leading to an incredible number of transistors in semiconductor dies and reprogrammable devices with high gate counts. Hardware designers need to take advantage of the amount of logic available while decreasing the overall design time.

While traditional hardware design tools trade off two main design constraints, area and delay, a third important metric is being relatively ignored. Design power consumption is growing in importance, particularly as fabrication technology improves and logic density increases. Chips with smaller feature sizes allow more on-chip logic and the potential for more power consumed. As a result, it seems logical that next generation hardware design tools must continue to target high-level language descriptions of algorithms and consider delay, area, and power consumed as constraints during synthesis.

Many vendors supply Intellectual Property (IP) logic cores specific to different hardware targets such as different makes of FPGAs or an ASIC process. These IP cores are generally hand optimized for different common occurring calculations from arithmetic operations to more complex DSP functions. Often these IP cores have several different implementations of a particular operation with different features including delay, area required, and power consumed. While these IP cores are generally intended for a hardware designer working directly with HDLs, an effective synthesis tool could benefit from an automated method for their inclusion in a generated hardware design.

This paper describes an automated, power-aware framework for the inclusion of vendor IP cores in PACT HDL, a C to HDL compiler that targets Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). The compiler uses techniques to "turn off" the IP cores through clock gating to allow the compiler to reduce power consumed.

1.1 PACT
PACT HDL is part of the PACT (Power Aware Architecture and

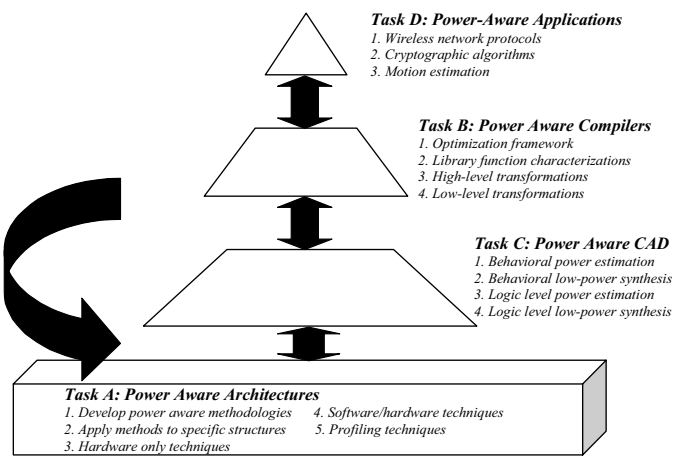

Figure 1.1. PACT Project Overview

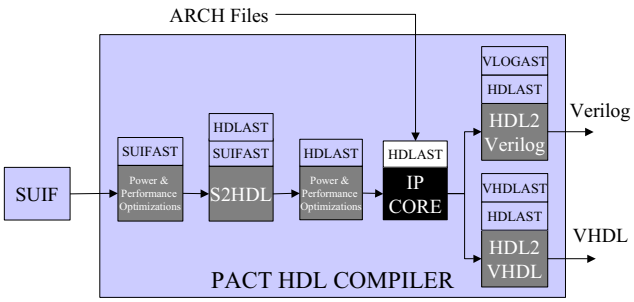

Figure 1.2: PACT HDL Compiler Flow

Compilation Techniques) project. The objective of PACT is to develop power-aware architectural techniques and associated compiler and CAD tool support that will take applications written in C and generate efficient code that runs on power-aware systems. The end goal is to generate power savings at all levels in the design process toward an overall high aggregate power savings. An overview of PACT is displayed in Figure 1.1.

More specifically, PACT HDL is a component of the PACT compiler (Task B). The PACT compiler targets a System on a Chip (SoC) style architecture consisting of reprogrammable components (FPGAs), application-specific components (ASICs), and a general-purpose processor (ARM). PACT HDL generates a power and performance optimized synthesizable Register Transfer Level (RTL) HDL code from a C description. This code is run through commercial synthesis tools to generate bitstreams for FPGAs or fabricatable hardware for ASICs. A detailed discussion of PACT HDL may be found in[6] with some initial power and performance optimizations discussed in[7].

1.2 Related Work

IP cores have been an important tool to hardware designers working directly with VHDL and Verilog for some time. This can be seen by the prolific vendors supplying cores such as Altera[1] and Xilinx[14] for FPGAs and Synopsys[12] for ASICs. IP cores are also a key factor in most design reuse schemes attempting to shorten time to market and move toward the SoC[8],[4].

A significant emphasis has been placed on low power design since the middle 1990s[11]. Clock gating in particular has been one of the earliest and most successful techniques for reducing power in digital circuits[2],[13]. There has also been work on targeting functional units as significant power drains and reducing their consumption[10].

As synthesis from high level programming languages becomes more popular, work in automated IP core use has started to become available. Accelchip (formerly Mach Design Systems) and Northwestern University have incorporated various IP cores for FPGAs into its MATLAB compiler[5] to attack area and performance issues. The ER group at UCLA has used a pattern recognition technique to utilize IP cores within automatically generated codes[3] also to improve performance and minimize area.

In contrast PACT HDL uses operator cores through user directives and compiler options. The use of architecture files makes the IP scheme generic so that as new cores become available they may be quickly utilized by the compiler. The PACT HDL compiler considers power consumed as well as area and performance as benchmarks to be considered by the framework. The PACT HDL contribution is the automated power-aware IP core framework through use of generic architecture file interface and automatic utilization of the clock gating technique for power savings.

1.3 Outline

The remainder of the paper is organized as follows: Section 2 describes the PACT HDL IP core framework including the use of architecture files and the clock gating technique. Results with image and signal processing benchmarks are discussed in Section 3. Section 4 draws some conclusions. Section 5 relates some on-going future work. References are listed in Section 6.

2. IP CORE FRAMEWORK

PACT HDL generates RTL HDL codes in VHDL and Verilog using a Finite State Machine (FSM) style. The PACT HDL Compiler flow is depicted in Figure 1.2. These codes use intrinsic operators to represent calculations such as addition, subtraction, and multiplication. The output HDL codes are passed to commercial synthesis tools that generate the gate level hardware descriptions. Each intrinsic operator is replaced with a hardware implementation of the calculation by the synthesis tool. Unfortunately, by leaving this decision to the synthesis tool, the gate-level instantiation may not be appropriate for the desired constraints, particularly those relating to power consumed. The synthesis tools tend to use combinational implementations that are area and power hungry. In some cases, the tool may not be able to instantiate the appropriate logic, such as the division operator, at all.

By binding operations to specific IP cores rather intrinsic operators different tradeoffs between power consumed, area required, and delay required may be explored. Instantiating IP cores in the compiler to replace intrinsic operators allows the compiler to specify how to reuse or replicate functional units which may have a significant impact on the design characteristics. The IP Core module has been highlighted in the flow shown in Figure 1.2.

2.1 Architecture Files

PACT HDL uses the concept of architecture files to allow the automated integration of hardware components external to the design. These components may consist of on or off chip RAMs, bus interfaces, functional units, and so on. The architecture files are designed to be easy to work with from a user standpoint and easy for the compiler to parse efficiently. These files describe a profile of the core, the interface signals, and component behavior

```
# 32 bit sequential integer multiplier
# built from 8-bit parallel multipliers
# from the Xilinx Core Generator
operation mul # multiplication operator
input 0 32 # first operand precision
input 1 32 # second operand precision
output 64 # output precision
static 3.767 # power in mW
dynamic 368.74 # power in mW
area 113 # area in CLBs
cycles 6 # clock cycles
```

Figure 2.1: Example for profile.ARCH

in a FSM style. By using this scheme, the compiler can easily assimilate the latest IP core technology that becomes available. In effect the architecture files model the timing behavior of the IP core much like the timing diagram in Figure 2.3.

The architecture file concept is necessary to the automated framework of IP core inclusion because it must be general enough to handle cores from different vendors with different naming schemes and assumptions on use. PACT HDL assumes that the IP core will be used in the HDL by component instantiation only due to its multiple HDL targets. The core itself must match the requirements of the operation in the code such as same number of inputs and outputs, sufficient precision, and so on.

2.2 IP Core Manager

The first stage of the IP Core module discovers the different components that are available to the compiler. This is done by parsing the profile.ARCH file for each component. An example profile.ARCH for a Xilinx LogiCore Multiplier[14] is listed in Figure 2.1. The operation line tells the compiler what kind of operation the IP Core performs. Each core input is declared on an input line with an index and precision parameter. The output line declares the output bit-width of the core. The last four lines included profiling information for the core.

In the example there is information for static and dynamic power, area, and number of clock cycles per execution included. The units are not specified to the compiler, in the example they are included only in comments. The compiler assumes that all profiles use the same units, but these units may be arbitrary. This assumption is necessary since units may not be compatible such as CLBs versus nanometers for area of an FPGA or ASIC respectively. In order to allow maximum generality, only relative values need be compared.

All of this information is stored in an IP Core Manager within the compiler. A scheduler or a core expansion phase may refer to this profile information or link to locations of other architecture files from the manager. The actual signals and behavior of the compiler are not stored in the manager.

2.3 Binding IP Cores

Upon completion of determination of available cores, a binding and expansion step ensues. Currently, the user determines the binding of IP cores statically either by command line compiler options or by directives in the C code. A default operator may be specified for all operations of a certain type (e.g. multiplication, division, etc) through a directive or specific operations may be bound to particular cores. The command line option may only specify a default operator. The default operator is helpful for implicit operations that may not be easily specified by the user, such as multiplications and additions generated by array address calculations. If no binding is specified for a particular operator, the standard intrinsic RTL operator is used. If more than one core is specified for any particular operation, only the first core specified is used.

```
COMPONENT:
      clk : 1 : in : clock
      a : 32 : in
      b : 32 : in
      q : 64 : out
      nd : 1 : in
      rdy : 1 : out
```

Figure 2.2: Example for signals.ARCH

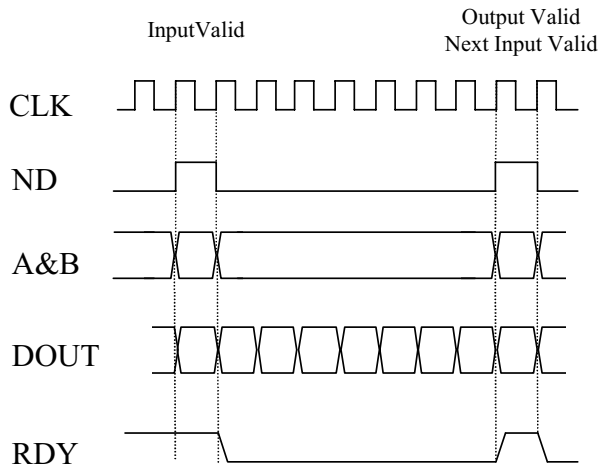

Figure 2.3: An example timing diagram of a IP Core functional unit

During the HDL AST traversal, at the first occurrence of an operation with an associated IP core an HDL component and instance are created. At this point, the signals.ARCH file is parsed so that appropriate signals may be created. Figure 2.2 contains an example signals.ARCH file for a Xilinx LogiCore Multiplier[14]. Each line in the file declares a signal used within the core component. The first field is the formal signal name, the second field is the bit-width of the signal, and the third field is the direction of the signal either in, out, or inout. An optional fourth field may specify if the parameter is a clock. A colon delimits each of the fields. The compiler automatically generates all of the appropriate instantiation signals based on these formal parameters to the IP core. The component clock is hooked up to the global clock and the other signals are available for use in the design.

2.4 IP Core Expansion

After the components and instantiations have been created, a third step is required for expanding these IP cores into the HDL code. The behavior of an IP core is generally described in a timing diagram. An example timing diagram for a two input single

```
STATE1:
      a : INPUT0
      b : INPUT1
      nd : 1
      STATE2

STATE2:
      nd : 0
      if rdy = 1 then STATE3 else STATE2

STATE3:
      OUTPUT : q
```

Figure 2.4: Example for comp.ARCH

```
when main_state21 =>
  main_tmp_t4_7 := main_tmp_t4_7 *
  main_tmp_t4_8
  main_ctrl <= main_state31;
```
(a)

```
when main_state21_1 =>
  mul32_par8_a_inst1 <= main_tmp_t4_7;
  mul32_par8_b_inst1 <= main_tmp_t4_8;
  mul32_par8_nd_inst1 <= '1';
  main_ctrl <= main_state21_2;

when main_state21_2 =>
  mul32_par8_nd_inst1 <= '0';
  if mul32_par8_rdy_inst1 = '1' then
    main_ctrl <= main_state21;
  else
    main_ctrl <= main_state21_2;
  end if;

when main_state21 =>
  main_tmp_t4_7 :=
  mul32_par8_q_inst1(31 DOWNTO 0);
  main_ctrl <= main_state31;
```
(b)

Figure 2.5: IP core expansion example (a) original with intrinsic operator (b) post expansion with IP core

output core is shown in Figure 2.3. This timing diagram is based on the timing diagram for a Xilinx LogiCore Multiplier[14]. In PACT HDL comp.ARCH file describes the behavior of the core in an FSM style. This file is derived from the timing diagram. An example of this is displayed in Figure 2.4 based on the diagram from Figure 2.3.

The comp.ARCH file is expanded into the HDL code using information from the original operation. In this example three states are created. In the first state, the parameters a and b are assigned inputs selected from the operation being replaced, the nd flag is set high, and the control is told to move to the next state. In the second state, the nd flag is set low and when the rdy signal is high it moves to the next state. Finally, the output is assigned signal q in the third state. The newly created states are then merged into the HDL AST.

The VHDL code for an actual expansion is exhibited in Figure 2.5 with the original pre-expansion state shown in (a) and the expanded code in (b). The main state ID is retained while sub IDs that correspond to the state numbers in the architecture file are appended. The example shows how the INPUT and OUTPUT terms in the architecture file are replaced by variables in the HDL AST used in the operation.

2.5 Clock Gating

In order to make the IP core inclusion power-aware, a clock gating mechanism was implemented within the framework. At the time of instantiation discussed in Section 2.3, a clock gate is also created if the core is clocked. This clock gate allows the clock to be enabled at the time of computation and disabled after the result has been received in effect turning it on only for computation. Clock gating may be enabled by the user through a command line compiler option.

To create the clock gate, separate component clock and component enable signals were instantiated. The gate logic is

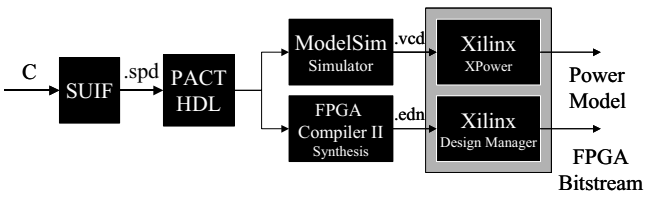

Figure 3.1: Tool flow for Xilinx FPGAs using PACT HDL

actually described as an and gate between the global clock and the component enable. In HDL this was implemented with a process model using the global clock and enable as a sensitivity list and a behavioral and statement between the two parameters that is assigned to the component clock. Because the logic is simple, it is expected that the design complexity increase will be negligible. The gate provides a potential for significant power savings.

3. RESULTS

The PACT HDL compiler with the automated IP core framework has been tested with several benchmarks and has been shown to generate correct VHDL and Verilog code. A benchmark suite of appropriate image processing codes and kernels was selected for testing and profiling of optimizations. These codes correspond to functions a hardware-software partitioner might select for execution on a reconfigurable or application specific hardware resource in a SoC environment. The benchmarks include:

- FIR Filter
- Matrix Multiplication
- Laplace Transform
- Sobel Transform
- Least Mean Squares

The Finite Impulse Response (FIR) filter is a band pass filter used in image and signal processing. Matrix multiplication is an $O(n^3)$ operation used in a variety of image and signal processing operations. The Laplace transform benchmark uses the Laplace operator $\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ to determine the gradient used for edge detection in images. The Sobel transform is another method to determine the gradient using two-dimensional linear convolution. Least mean squares (LMS) is adaptive filtering technique used for system convergence. It is used in many different signal processing applications such as voice over IP. LMS is an important benchmark because it includes division, which is not directly synthesizable in traditional hardware synthesis tools.

3.1 Tool Flow

The IP cores profiled in this study were from the Xilinx CoreGen tool[14]. All the benchmarks were simulated for correctness using the ModelSim Simulator from Model Technologies[9]. The RTL codes were synthesized targeting the Xilinx Virtex XCV400 using FPGA Compiler II from Synopsys[12] with a request clock frequency of 50MHz. The synthesized codes were placed and routed using the Xilinx Design Manager. A simulation force file was generated using random input values to simulate an actual data set. The resulting timing simulation file from Design Manager is simulated at 50MHz with the ModelSim using the generated force file. The output of this simulation is a file

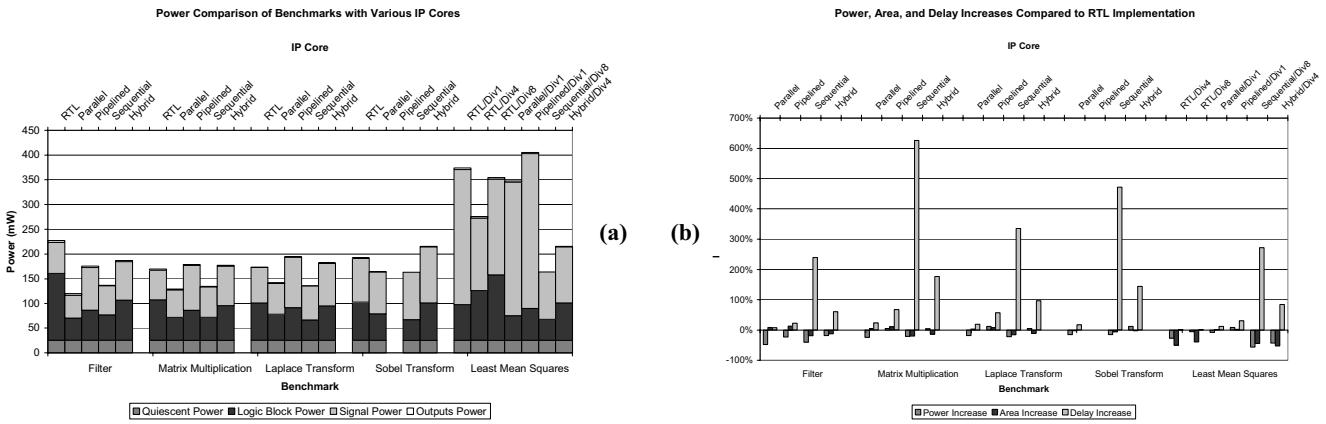

Figure 3.2: Results for benchmarks with various IP cores targeting a Xilinx Virtex XCV400 FPGA running at 50MHz with Vdd of 2.5V (a) power comparison of benchmarks in mW (b) percentage increase of power, area, and delay compared with RTL design

representing the actual gate level behavior of the system for the random dataset. Finally this file is used in combination with the placed and routed design in the Xilinx Xpower tool for the estimated power consumed by the design running at 50MHz with Vdd of 2.5V. The Design Manager and ModelSim simulator reports area and timing data, respectively.

3.2 Results for IP Core Inclusion

Each of the benchmarks was profiled using a direct RTL translation without IP cores, a parallel (combinational) multiplier, a pipelined multiplier, a sequential multiplier, and a hybrid combinational/sequential multiplier. For all versions the inputs were 32-bit and the output was 64-bit. In addition, the LMS algorithm used three different pipelined dividers. For each divider the latency was on the order of 32 cycles while the throughput was varied between one, four, and eight cycles. Figure 3.2 shows the power, area, and delay results for the various benchmarks.

In the case of the first four benchmarks the RTL version is compared with parallel, pipelined, sequential, and hybrid IP core implementations of a multiplier. For the LMS algorithm, the RTL multiplier is first combined with each of the three implementations of a pipelined divider. For this design, parallel refers to the parallel multiplier with the single cycle throughput divider, pipelined refers to the pipelined multiplier with the single cycle throughput divider, sequential refers to the sequential multiplier with the eight cycle throughput divider, and hybrid refers to the hybrid multiplier with the four cycle throughput divider.

In Figure 3.2 (a) it is shown that the IP core selected for a design can have a significant impact on the power consumed by the design. As expected, designs using the combinational multiplier implemented by the synthesis tool consume some of the most power of all the designs using the IP core multipliers. Surprisingly, the pipelined and hybrid multipliers tend to require similar amounts of power and in some cases more than the RTL implementation. As expected, there is significant power variation among the IP core multipliers. The parallel multipliers tend to consume the least power compared to the other multipliers on the order of the 32-cycle sequential multiplier. Notably, the power results do not always have the expected trends concerning chip area required (e.g. chip area and power consumed trend in similar directions).

Figure 3.2 (b) shows the percent increase in the IP core based designs of power, area, and delay. The parallel multiplier shows a power improvement over the RTL design despite the tendency to increase the area. The delay of the design also increases slightly in all cases. The parallel multiplier does require two additional states over the RTL intrinsic operator, which is responsible for the area and delay increases. The pipelined multiplier requires more area than the parallel core, has a longer delay and consumes significantly more power, making it a bad choice overall. The sequential and hybrid multipliers tend to reduce the area but the delay increase is rather high, reaching over 400% increase in the matrix multiplication. The power savings versus delay tradeoff would need to be evaluated to determine if the sequential or parallel multiplier should be used for the design.

3.3 Results for IP Cores with Clock Gating

As described in Section 2.5 clock gating techniques were applied to the cores included in the design. The hope was to decrease power consumed by the designs without any significant area increase or performance degradation. A comparison between the

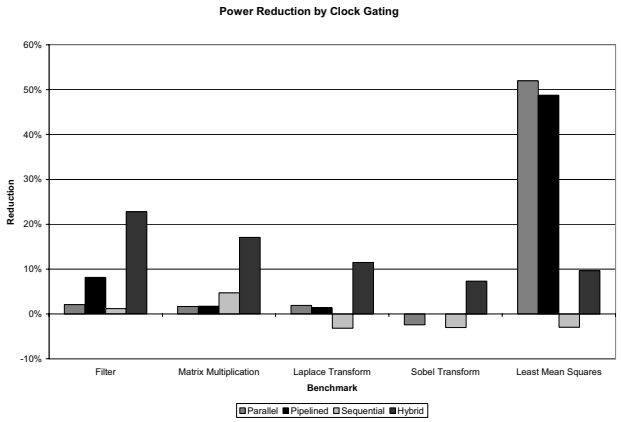

Figure 3.3: Percent power reduced by clock gating over original design

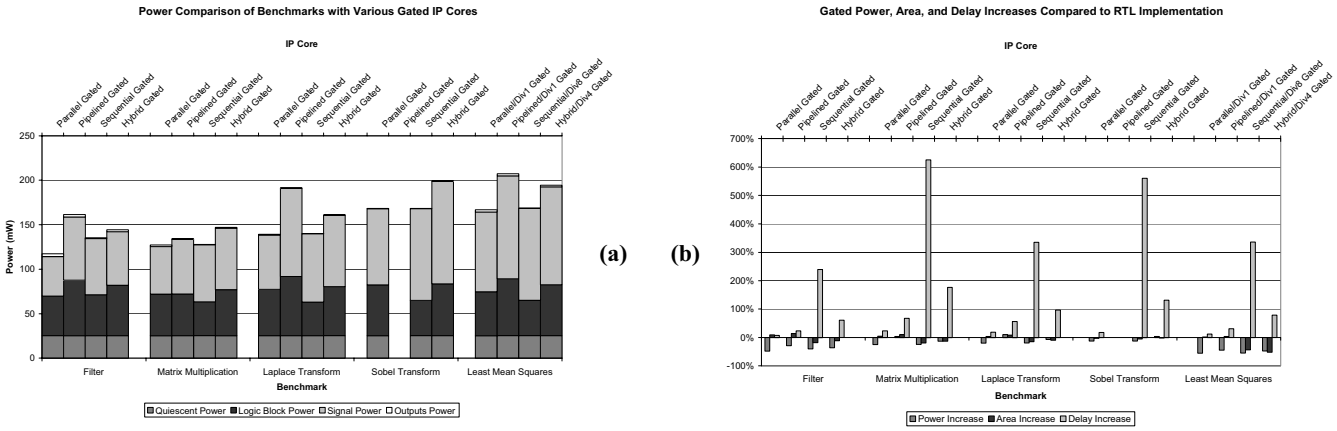

Figure 3.4: Results for benchmarks with various gated IP cores targeting a Xilinx Virtex XCV400 FPGA running at 50MHz with Vdd of 2.5V (a) power comparison of benchmarks in mW (b) percentage increase of power, area, and delay compared with RTL design

non gated results shown in Figure 3.2 and described in Section 3.2 and clock gated results is shown in Figure 3.3.

In general, the hybrid multiplier shows the biggest improvement amongst the first four benchmarks and a reasonable improvement in the LMS algorithm. The pipelined multiplier tends to show more modest reductions with some consistency. The parallel and sequential multiplier savings are either very modest or in some cases there was actually an increase in consumption versus non-gated results. These results may be due more to variations in the synthesis results rather than the effect of the clock gating on the design. The results of the first four benchmarks and the LMS algorithm are quite different.

The LMS shows around a 90% reduction for the parallel and pipelined multiplier with gating and the single cycle throughput divider. Since this gated multiplier did not have a significant impact on the other designs, it reasonable to assume that the power saved is from the divider core, particularly since both designs use the single cycle throughput divider. The sequential multiplier with the 8 cycle throughput divider show a slight power increase over the non gated design. Figure 3.2 (a) shows the power consumed by the non-gated design, which is already rather low. In this case it appears the power consumed did not have much room for improvement.

Figure 3.4 shows characteristics of the gated IP cores similar to the results for the non-gated cores shown in Figure 3.2. Figure 3.4 (a) shows how the clock gated codes tend to normalize the results for all the benchmarks to between the 150 to 200 mW range. The data in Figure 3.4 (b) shows how the power has changed from the gated over the non-gated designs but the area required and delay have not changed significantly, which was the hope from Section 2.5.

4. CONCLUSIONS

This work demonstrates automated framework in PACT HDL to include IP logic cores directly into the resulting hardware codes from a C description. The results from the benchmark suite show the potential for significant power savings when selecting the appropriate IP core for use in the design see Figure 3.2 and discussion in Section 3.2. Unlike other work in this area, the PACT HDL IP core framework is power-aware through use of the clock gating technique, allowing a functional unit to be "turned off" when not in use. The initial results show promising results that clock gating is a valid technique for reducing power in hardware designs using clocked IP cores, see Figure 3.3 and discussion in Section 3.3.

5. FUTURE WORK

Currently, the automated IP core framework depends on user directives to make decisions about where to use operators in the code. Since the IP Core Manager described in Section 2.2 contains information about expected power consumed, area required, and required delay, and automated scheduler could make decisions in the absence of compiler directives. The compiler could attempt to optimize power under delay and area constraints. It is possible that the compiler's decisions could significantly improve the power consumed over the static, user defined model.

Another improvement might be to extend the IP core framework to recognize codes fragments for which a core exists rather than just simple operator replacement. There has been some work already in this area discussed in Section 1.2.

6. REFERENCES

[1] Altera *Intellectual Property Catalog*, www.altera.com

[2] Benini, L., Siegel, P., De Micheli, G., *Saving Power by Synthesizing Gated Clocks for Sequential Circuits*, DATC 1994.

[3] Bozorgzadeh, E., Memik, S., Kastner, R., Sarrafzadeh, M., *Pattern Selection: Customized Block Allocation for Domain-Specific Programmable Systems*, FPGA 2002.

[4] Halambi, A., Grun, P., Tomiyama, H., Dutt, N., Nicolau, A. *Automatic Software Toolkit Generation for Embedded Systems-On-Chip*, ICVC 1999.

[5] Haldar, M., Nayak, A., Choudhary, A., Banerjee, P., *A System for Synthesizing Optimized FPGA Hardware from MATLAB*, ICCAD 2001.

[6] Jones, A., Bagchi, D., Pal, S. Banerjee, P., Choudhary, A. "PACT HDL" appears in *Power Aware Computing*. Graybill, R., Melhelm, R. Kluwer, Boston.

[7] Jones, A., Bagchi, D., Pal, S., Tang, X., Choudhary, A., Banerjee, P. *PACT HDL: A C Compiler Targeting ASICs and FPGAs with Power and Performance Optimizations*. Submitted to ICCD 2002.

[8] Keating, M., Bricaud, P., Reuse *Methodology Manual for System-on-a-Chip Design*, Kluwer, 1998.

[9] Model Technology ModelSim www.model.com

[10] Musoll, E., Cortadella, J. *High-level Synthesis Techniques for Reducing the Activity of Functional Units*, ISLPED 1995.

[11] Rabaey, H., Pedram, M. ed. *Low Power Design Methodologies*, Kluwer 1996.

[12] Synopsys *Design Compiler, FPGA Compiler and DesignWare IP*, www.synopsys.com

[13] Tellez, G., Farrahi, A., Sarafzadeh, M. *Activity-Drive Clock Design for Low Power Circuits*, ICCAD, 1995.

[14] Xilinx *ISE Foundation*, www.xilinx.com