IEEE Int. Conf. on Computer Design (ICCD) October 2006

Dynamic Co-Processor Architecture for Software Acceleration on CSoCs

Abhishek Mitra, Zhi Guo, Anirban Banerjee, Walid Najjar
Department of Computer Science and Engineering, University of California Riverside, CA 92521
E-mail: {amitra, zguo, anirban, najjar} @cs.ucr.edu

Abstract

By integrating one or more (hard or soft) CPU core on the chip, new generation platform FPGAs have become configurable systems on a chip (CSoC) that support a combined software and hardware execution model. More recently, FPGAs, using new design tools, have also provided support for partial reconfiguration. The system designer is left with the task of interfacing the IP Cores to the CPU and also for realizing partial reconfiguration across the cores. In this paper, we describe a software tool to automate the interface between the CPU and the reconfigurable fabric. Our tool generates hardware wrappers for the IP Cores that makes them look like a C function invocation in the source code. We also use our tool to support partial reconfiguration: the same wrapper is used for a multitude of IP Cores and the user selects the core to be invoked in the program.

1. Introduction

Modern FPGAs integrate a (hard or soft) processor core, with the reconfigurable fabric. These CSoC use the CPU to support the software execution and rely on one or more hardware cores for accelerating frequently used code segments (loop nests). These hardware cores can be either custom designed or can be selected from a library of pre-existing IP Cores. The hardware cores are tightly coupled with the CPU using very high speed, point to point links for fast data transfer in the Virtex-4 FX CSoC. CPUs such as the PowerPC 405 also support custom instructions for communicating with these co-processors. The co-processors act as accelerators for compute intensive portions of the applications such as floating point intensive calculations [17], discrete transformation algorithms (FFT, DWT, DCT, etc) [5] [7] [13], and also for custom applications such as Smith-Waterman string matching, encryption/decryption engines, etc.

A multitude of co-processing functionality can be realized using IP Cores that are highly configurable and performance optimized. Interfacing the available library of IP Cores to the on-chip processor is a time-consuming and tedious task which almost always, needs to be taken care of manually. The system designer is left with the task of interfacing each and every core to the co-processor interface [22].

Partial reconfiguration offers the system designer the possibility to leverage the same basic hardware structure for accelerating multiple tasks (programs) on the CSoC. Partial reconfiguration on the FPGA makes it possible to create a system that can enable re-configuration of pre-assigned parts of the FPGA without affecting the static parts, or inducing a system-wide reset. The high overhead of reconfiguration, at this point in its development (msec), precludes using it dynamically within the same task. It is however a very powerful tool to overcome the area limitation of a single FPGA platform across multiple applications since re-configuration can be combined together with a CPU context switch. The system designer is also left with the task of generating the interface between static and dynamic regions of the FPGA as required for partial reconfiguration [2] [9] [10] [11] [19].

In this paper we describe a software tool for automatically generating the communication interface between the software running on the CPU and a tightly coupled IP Core based co-processing system on the Virtex-4 FX FPGA. It generates hardware wrappers for the core that makes it look like a C function invocation in the source code. We extend this tool to support partial reconfiguration: the same static wrapper is used for multiple cores and the user selects the core to be invoked in the program.

Our compiler for FPGA-based reconfigurable systems, ROCCC [16] leverages the huge wealth of IP Cores by allowing the user to import these cores into the source code. The compiler automatically generates a wrapper structure that would hide the timing and stateful nature of the IP Cores and makes each available to the C language compiler, as an un-timed side-effect free function call.

We also use the ROCCC compiler approach to support run-time reconfiguration by automating the generation of the interface between static and dynamic regions of the FPGA. The user can switch between multiple functional units by calling the appropriate C function in the code, thus entailing the use of the same hardware wrapper for multiple IP Cores.

Utilizing our software tool along with the ROCCC infrastructure we have been able to automatically configure multiple IP-cores on the fabric viz. FP (Floating point) Adder, FP Multiplier, Integer divider

CORDIC engine and an FFT engine. Moreover using partial reconfiguration we have been able to overcome the area limitation of a single FPGA platform (Virtex-4 FX12), using five different IP Cores. We have allocated a region of 1800 slices for the co-processor, thus resulting in a reduction in the floor area by 2656 slices due to partial reconfiguration. Moreover the area dedicated for the hardware wrapper is no more than "14 " slices, quite miniscule, when compared to the actual IP Core area.

The rest of the paper is organized as follows: Section 2 details out the system overview and the associated software tools. Experimental details have been chalked out in Section 3, with related work in section 4. We summarize the conclusions in section 5.

2. System Overview

The target of our research emphasizes automatic wrapper generation and reconfiguration for IP Cores configured on CSoC systems. These systems are self contained embedded processing solutions often targeted for reconfigurable computing applications. The major ingredients in our system are the CSoC system, IP Core libraries and the Compiler infrastructure (ROCCC).

2.1 CSoC platform

Our CSoC system consists of a Platform FPGA, which in turn are field programmable gate array logic along with one or more (soft/hard) processors all on a single chip. The CPU on the CSoC runs an Operating System as well as application software. With the advent of higher performance FPGA fabrics it is now possible to instantiate software code accelerators on the FPGA and use it for speeding up execution on the processor. In the past, the limiting factor for speedup of these FPGA based accelerators had been the on-chip bus used for data communication between the host-code and the accelerator, since the same bus is used for various other peripherals too. Software developed around the PowerPC core on the Virtex-4 FX FPGA can communicate with fabric co-processors using point to point buffered links (also known as Fast Simplex Links) [25] hence alleviating performance based issues, present on a bus based architecture [23] [24] [25]. The Virtex-4 FX also provides a high performance bus architecture (PLB and OPB) for connection with various on-fabric peripheral controllers such as memory (DDR/SRAM) controller, Ethernet, UART, keyboard and mouse controller, Peripheral controllers are synthesized as soft cores on the FPGA fabric, thus user defined peripherals may also use this bus for communication with the CPU or other on-chip peripherals.

2.2 APU (Auxiliary Processing Unit) on Virtex-4 FX

The PowerPC 405 core on the Virtex-4 FX FPGA is a 32-bit architecture with on-chip instruction and data cache memory. An Auxiliary Processor Unit (APU) [1] controller accompanies the core to interface it to hardware accelerators on the fabric. The APU supports 32-bit custom instructions and 64-bit data. The co-processors instantiated on the Virtex-4 FX FPGA use the APU on the PowerPC for seamless communication with the FPGA fabric. Additionally there is also an option to use a bus based architecture, FCB (Fabric Co-Processor Bus) for sharing the APU with more than one co-processor.

As depicted in Figure 1, the system architecture used for our dynamic co-processor system involves a Xilinx Virtex-4 FX, the APU interface and two FSL channels. Data is sent/received over the FSL link from the Power PC to the compiler generated (APU to IP) wrapper. The wrapper parses input/output data according to the current IP Core instantiated on the dynamic fabric and maps them onto the Bus macro interface. The Bus macros interface the static wrapper to the dynamic wrapper and through it to the IP-core. Handshaking/control signals are mapped onto the Control bus and status/acknowledgment signals from the IP Core to the wrapper are mapped onto the Status bus.

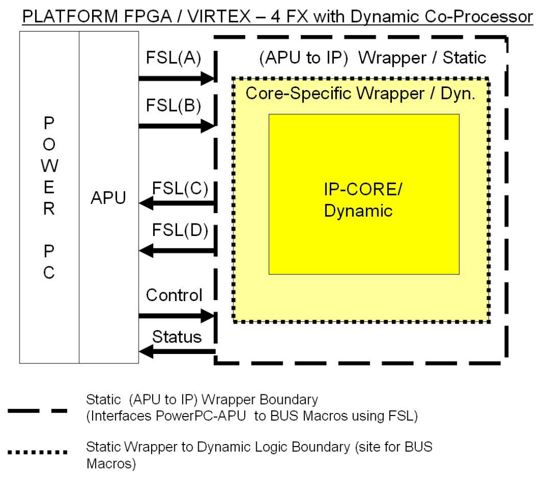

PLATFORM FPGA / VIRTEX – 4 FX with Dynamic Co-Processor
POWER PC
APU
FSL(A)
FSL(B)
FSL(C)
FSL(D)
Control
Status
(APU to IP) Wrapper / Static
Core-Specific Wrapper / Dyn.
IP-CORE/ Dynamic
Static (APU to IP) Wrapper Boundary (Interfaces PowerPC-APU to BUS Macros using FSL)
Static Wrapper to Dynamic Logic Boundary (site for BUS Macros)

Figure 1. System Architecture of the dynamic co-processor system

2.3 IP Cores

Intellectual Property cores have been available for a while for FPGA based systems. These IP-cores are highly optimized replacement for sequential software used for time-critical systems such as real time audio-video encoders/decoders, FIR filters, DSP blocks and also for highly specialized applications such as string matching based on Smith-Waterman algorithms. These IP Cores have also been used in various FPGA based applications for rapid prototyping of system accelerators and co-processors. Most IP-cores are macros for FPGA which are already mapped to the target architecture and many of them are relationally placed and routed as well. IP Cores result in higher performance designs along with lowering of the design effort for the system. Most IP Cores share a similar input/output architecture which consist of two input bus and one to two output bus, along with certain control/acknowledgement signals. Thus it is possible to encompass these interfaces into a standard I/O wrapper architecture [27] [28] which would serve as a superset for I/O interface to all possible IP-cores targeted at a particular system. Our system currently targets such compatible IP cores with future extensions planned for IP cores with arbitrary number of inputs or outputs.

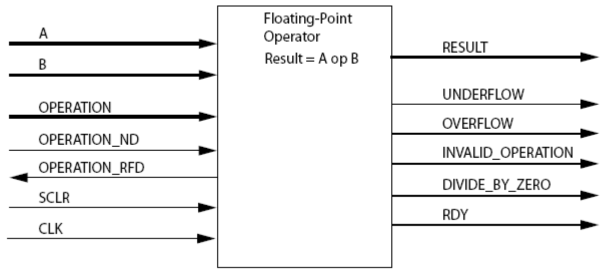


Figure 2. An example Floating Point IP Core

Depicted in Figure 2 is an example of a compatible IP Core viz. a Logicore series Floating Point unit from Xilinx and Qinetiq [26]. The input bus (A, B) and output bus (RESULT) can be configured either as 32-bit single precision or 64-bit double precision, conformant to the IEEE 754 specifications. This core can be configured for a floating point operation such as adder, subtractor, multiplier, divider, square-root, and comparator. Adder and subtractor can be combined in a single unit. Various status signals originating from the IP Core are Underflow, Overflow, Invalid operation, Divide by Zero. The OPERATION signal selects either Add / Subtract, or from a multitude of compare operations if a Comparator is configured. The computational latency of the floating point unit is 5 clock cycles.

2.4 ROCCC Overview

An overview of the ROCCC framework is depicted in Figure 3. We have separated the front and back ends to achieve modularity and eventually allow the use of a variety of front end and back end tools.

ROCCC is built on the SUIF2 [14] and Machine-SUIF [15] platforms. It compiles C code into VHDL code for mapping onto the FPGA fabric of a CSoC device. Information about loops and memory accesses is visible in our front-end IR (intermediate representation) viz. Hi-CIRRF (Compiler Intermediate Representation for Reconfigurable Fabrics). Accordingly, most loop level analysis and optimizations are done at this level. ROCCC performs a very extensive set of loop analysis and transformations, aiming at maximizing parallelism and minimizing area. The compiler also minimizes accesses to memory by effecting smart re-use of data. The compiler also performs scalar replacement at front-end. All memory loads are moved to the top of the loop body and all memory stores are moved to the bottom of the loop body.

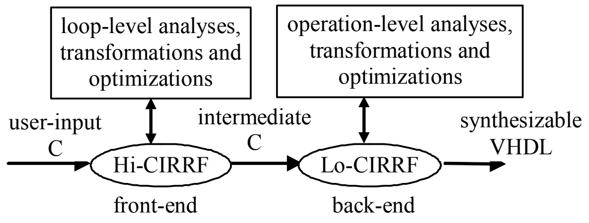

loop-level analyses, transformations and optimizations
operation-level analyses, transformations and optimizations
user-input C
intermediate C
synthesizable VHDL
Hi-CIRRF
Lo-CIRRF
front-end
back-end

Figure 3. ROCCC system overview

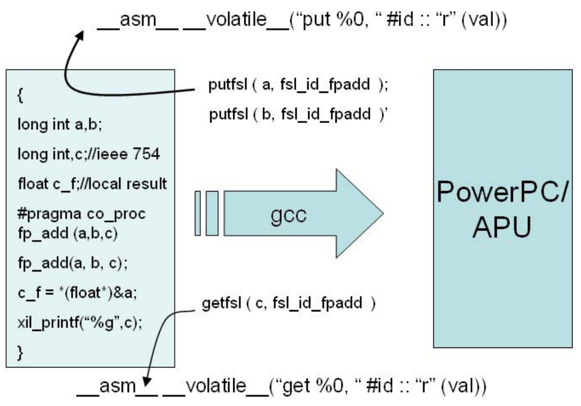
Machine-SUIF is an infrastructure for constructing the back end of a compiler. Machine-SUIF's existing passes, like the Control Flow Graph (CFG) library [30], Data Flow Analysis library [31] and Static Single Assignment library [32] provide useful optimization and analysis tools for our compilation system. We build the back-end using Machine-SUIF. The compiler's back-end Lo-CIRRF, converts the input from control flow graph (CFG) into data flow graph (DFG), and generates synthesizable VHDL codes. We rely on commercial tools viz. Xilinx XST to synthesize the generated VHDL codes for Virtex-4 FX.

2.5 Interface Synthesis

As introduced in the system overview section, the ROCCC compiler generates synthesizable VHDL code for applications written in un-timed C. In this section, we present our approach using the ROCCC system to wrap IP Cores. The compiler takes in a C-function intended for co-processing operation and automatically generates the corresponding IP Core, along with high-level abstractions. Taking the high-level wrapper abstractions as input, ROCCC generates synthesizable wrappers in VHDL separately as well as C language driver code for communication across the FSL channels. The wrappers are instantiated as components in the outer circuit and enable a seamless connectivity between the on chip CPU and the IP Cores instantiated on the fabric.

2.5.1 C language function calls. ROCCC recognizes co-processing function calls by checking a certain pragma and records this pragma into an Intermediate Representation field for further use. It inserts Assembly code required to access the FSL channels. The *putfsl* assembly call is used to write 32-bit data to the FSL, while *getfsl* call reads back 32-bit data from FSL. The software function call to the co-processor sends/receives 32-bit data through the putfsl/getfsl assembly calls as depicted in Figure 4. The APU copies the data into/from the FSL and therefore to/from the static wrapper i.e. the (APU to IP) wrapper.

The Internal Configuration Access Port (ICAP) API is used by the function call to load in a partial bitstream file in order to re-program the co-processor region with a new IP Core by making use of the OPB-HWICAP hardware.


```
__asm__ __volatile__("put %0, " #id :: "r" (val))

{
long int a,b;
long int,c;//ieee 754
float c_f;//local result
#pragma co_proc
fp_add (a,b,c)
fp_add(a, b, c);
c_f = *(float*)&a;
xil_printf("%g",c);
}

putfsl ( a, fsl_id_fpadd );
putfsl ( b, fsl_id_fpadd )'

getfsl ( c, fsl_id_fpadd )

__asm__ __volatile__("get %0, " #id :: "r" (val))
```

Figure 4. The C function call to the co-processor and the #pragma directive

2.5.2 Generation of the static (APU to IP) wrapper. The static wrapper provides an interface between the PowerPC APU and the first stage into the fabric, as depicted in Figure 5. The static wrapper uses the standard FSL interface, to provide for data input/ output and clock signals for synchronization. The static wrapper buffers the input data and presents them to the Bus macros and also buffers output data to be communicated back using the FSL channel and into the Power PC APU.

2.5.3 Dynamic wrapper. The dynamic wrapper is a second wrapper which is generated in the partial reconfigurable region of the FPGA. It is a VHDL entity which connects the 32-bit input/output signals, the control signal, and the status signal from the Bus macros onto the corresponding ports of the IP Core. We would like to emphasize that the connectivity from/to Bus macros for each IP-core is specified in its respective dynamic wrapper. Thus the dynamic wrappers present a standard interface for connectivity between Bus macros and the IP-core as shown in Figure 6.

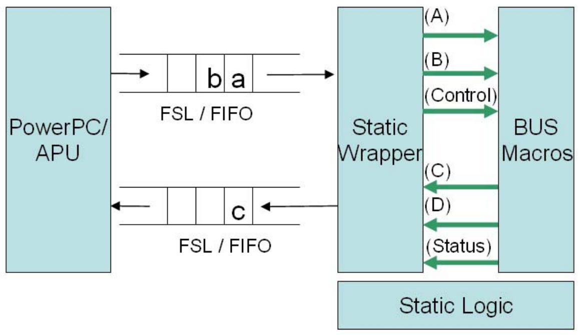


Figure 5. Data flow using FSL from the APU to the static wrapper

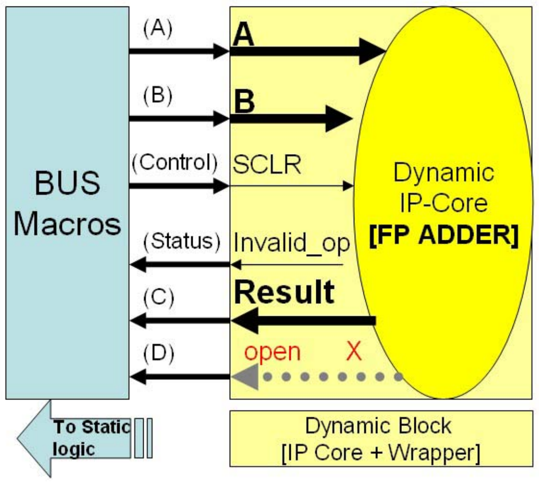


Figure 6. The dynamic block consisting of the dynamic wrapper around its accompanying dynamic IP Core

A compiler generated dynamic wrapper is depicted in Figure 7, which maps the Bus macro interface to the ports of the IP Core. The input signals A, B, and output signals C, D are connected to the Bus macros during synthesis and so are the control/status signals.

2.5.4 Dynamic Co-Processor Instantiation. We also use our tool to support dynamic partial reconfiguration. Dynamic partial reconfiguration at runtime allows re-use of FPGA resources to obtain a plurality of functionality, from the same hardware block, but at different times, and also without affecting the static parts of the device. The compiler generates the wrappers for each IP Cores that need to be dynamically reconfigured.

```
entity rmodule is
   Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
          B : in  STD_LOGIC_VECTOR (31 downto 0);
          C : out  STD_LOGIC_VECTOR (31 downto 0);
          D : out  STD_LOGIC_VECTOR (31 downto 0);
          clk : in  STD_LOGIC);
end rmodule;

architecture Behavioral of rmodule is
component cordic_module –From Logicore
                  port (
                  x_in: IN std_logic_VECTOR(15 downto 0);
                  y_in: IN std_logic_VECTOR(15 downto 0);
                  phase_in: IN std_logic_VECTOR(15 downto 0);
                  x_out: OUT std_logic_VECTOR(15 downto 0);
                  y_out: OUT std_logic_VECTOR(15 downto 0);
                  clk: IN std_logic);
end component;

u_cordic: cordic_module
                  port map (
                  x_in => A(31 downto 16),
                  y_in => A(15 downto 0),
                  phase_in => B(15 downto 0),
                  x_out => C(31 downto 16),
                  y_out => C(15 downto 0),
                  clk => clk);
end Behavioral;
```

Figure 7. A compiler generated dynamic wrapper for CORDIC engine

The design flow in Figure 8 involves the generation of the static logic along with the various partial reconfigurable logic (wrapped IP Cores). Thereafter the FPGA is floor planned to allocate a pre-determined area for the dynamic logic and the rest of the floor area is dedicated to static logic. The area dedicated to the dynamic logic, also known as the PR-Block (Partial Reconfigurable Block), is such that it may allow for the largest IP block to be placed and routed within it. I/O and communication of the static logic with the PR-block takes place using certain pre-configured hard macro blocks known as Bus macros [9], as shown in Figure 9. These Bus macros need to be manually placed around the boundary of the PR-block. We have employed the Xilinx PlanAhead 8.1 visual floorplanning tool for iterative design and placement. The final stages of the partial reconfigurable flow generates 'N' static bitstreams and 'N' partial bitstreams, where 'N' is the number of different IP Cores which are to be configured in the PR-Block. Each of the 'N' static bitstream contains the static design with the IP-Core numbered 'N' already programmed into the stream, while each of the 'N' partial bitstreams contains the logic to re-program the PR-Block with the functionality of the 'N'th IP Core. Thus the system may choose to start with one of the static bitstreams during power-up and thereafter reprogram the PR-Block with the desired functionality.

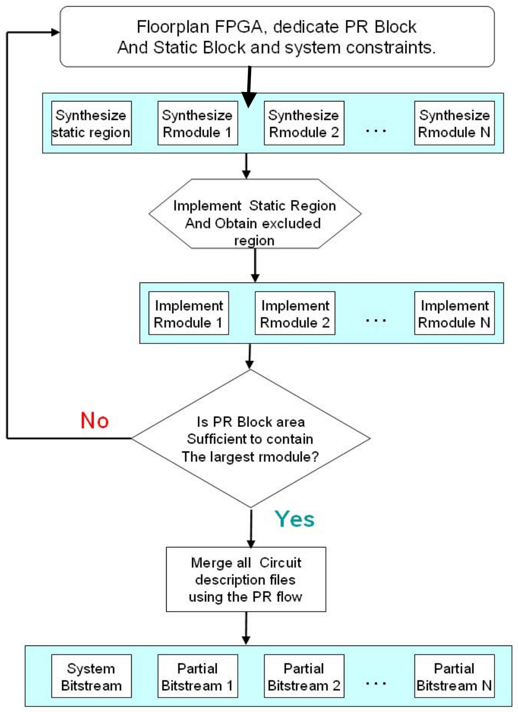

Floorplan FPGA, dedicate PR Block And Static Block and system constraints.
Synthesize static region
Synthesize Rmodule 1
Synthesize Rmodule 2
...
Synthesize Rmodule N
Implement Static Region And Obtain excluded region
Implement Rmodule 1
Implement Rmodule 2
...
Implement Rmodule N
Is PR Block area Sufficient to contain The largest rmodule?
No
Yes
Merge all Circuit description files using the PR flow
System Bitstream
Partial Bitstream 1
Partial Bitstream 2
...
Partial Bitstream N

Figure 8. The Partial Reconfiguration Flow for FPGA

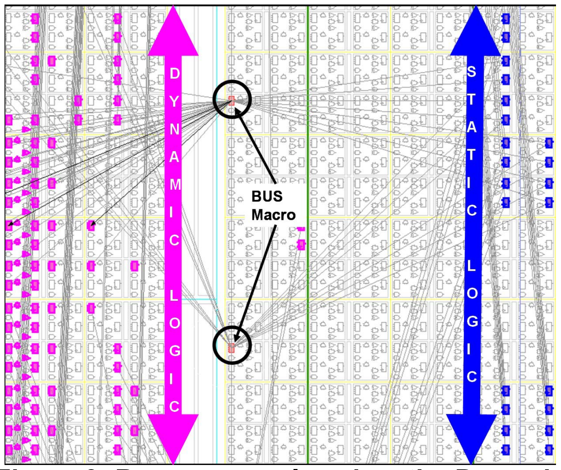


Figure 9: Bus macros placed on the Dynamic /Static logic boundary

3. Experimental Results

We have incorporated four Xilinx Logicore IP Cores in our compiler infrastructure for the purpose of conducting experiments. These cores are enumerated in, Table 1. The CORDIC (Coordinate Rotational Digital Computer) IP Core [33] performs a rectangular-to-polar vector translation. The IP Cores takes in as input the angle and magnitude in a polar coordinate and generates the equivalent vector (X, Y) in Cartesian coordinate. The CORDIC module has been configured to utilize eight DSP48 blocks for fast multiplication for calculating the new coordinates and to enable scaling.

The floating point adder, on the Xilinx Logicore IP Core [26] takes in two 32-bit single precision values conformant to the IEEE 754 standard (A, B) and outputs their sum in single precision (result). The floating point multiplier takes in two 32-bit single precision values (A, B) and outputs their 32-bit product (result). The FP multiplier [26] has been configured to utilize four DSP48 blocks for fast multiplication of the significand values from the floating point inputs.

Table 1: The area covered by the dynamic IP Cores

IP Core	Slices	DSP 48 Blks	Clk Spd MHz	Bit-Size KB	Reconfig Time	
					JTAG	Select MAP ICAP
Floating Pt Add 32bit	431	0	250	79	0.2 sec	5ms
Floating Pt Multi 32bit	189	4	218	76	0.19 sec	4.8ms
CORDIC coordinate rot. 16-bit	989	8	220	99	0.24 sec	6ms
Fixed Point Div. 32bit	1111	0	228	112	0.28 sec	7ms
FFT16, 256pt	1736	9	250	142	0.29 sec	7.1ms

The IP Core for a pipelined Integer divider [34] does arithmetic division on a 32bit dividend and a 32bit divisor thus resulting in a 32bit quotient and a 32-bit fraction value.

For calculation of FFT, we have configured the Logicore FFT IP Core [35] for 256 points, operating on 16-bit data. The core is configured for Burst I/O for non simultaneous processing and data loading/unloading. Nine DSP48 blocks have been utilized for fast multiplication operations. The static wrapper contains logic for timing and burst data loading/unloading from the FFT unit.

We have targeted the Xilinx Virtex-4 FX 12 FPGA containing 5472 slices on the ML403 Evaluation board. The design tools that we used are the Xilinx EDK 8.1 for generation of the Implementation files for the static subsystem and various wrappers for peripherals. We used Xilinx ISE 8.1i XST for synthesizing, and Xilinx PlanAhead 8.1 for floorplanning, implementing and testing the partial reconfiguration designs.

These five examples illustrate how a multitude of IP Cores can be effectively configured as co-processors on a FPGA using C based function calls. The execution time overhead at both the input side and output side for these four examples is one clock cycle except for the static wrapper for FFT engine. The configuration units (slices) dedicated to the reconfigurable block is 1800 slices as shown in Figure 10 and bus macros and wrappers account for less than 1% of slices dedicated to the PR Block (Table 2).

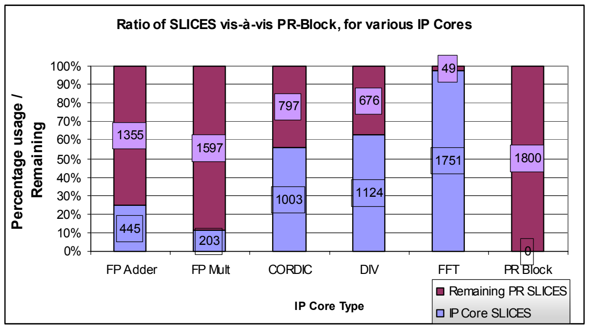


Figure 10: SLICE usage for IP Cores

Table 2: The area coverage of the IP Wrappers

Entity		**FP-Add**	**FP-Mult**	**CORDIC**	**DIV**	**FFT**
Static wrapper	SLICEs	12	12	12	12	12
	SLICE% w.r.t IP Core	2.7	6.4	1.2	1.06	0.74
	Cycle induced	1	1	1	1	1
Dyn. wrapper	Area (slice)	2	2	2	2	2
	SLICE% w.r.t IP Core	0.45	.98	0.2	0.17	0.11
	Cycle induced	1	1	1	1	1
Wrappd IP Core	Total SLICES	445	203	1003	1124	1751
	Total cycles	7	7	23	3	400
PR Blk	**1800 SLICEs** dedicated for the PR Block					

4. Related Work

Intellectual Property cores have been available for a while for FPGA based systems and have been successfully used by developers of such systems. Xilinx Logicore series of IP Cores are a library of highly available cores and have been extremely popular with designs based on Virtex series FPGA [7] [12] [13]. The XILINX IPIF module attempts to target connectivity of IP Cores to FPGA [27] [28], but does so only for the slower peripheral busses.

Targeting IP Cores to the FPGA peripheral bus using wrappers is discussed in [20] [22] [28].

Since IP-cores provide a black / gray box paradigm, system verification and integration maybe an issue. These have been documented in light of popular simulation and programming tools in [4] and the advantages and challenges in development of interface synthesis has been targeted in [5]. IP Core Reuse has been effectively discussed in light of a co-design paradigm in [3].

An automatic generator of interface synthesis for PowerPC software to custom software accelerators is detailed in [1].

Standards based IP bus interfaces such as the VSIA (Virtual Socket Interface Alliance) specify interface standards allow IP Cores to fit into "virtual sockets" [6]. However, the current condition is that numerous standards exist and no standard is adopted widely [21].

Two popular FPGA configuration mechanisms required for Partial-Reconfiguration (PR) along with their performances are discussed in [9] [11] [29].

Since development of a PR system on a FPGA entails working with a birds-eye view of the chip for layout and interface planning, thus the use of graphical environment leads to proper and efficient floor-planning and the process is documented in [10].

An early toolkit (PARBIT) targeted at the Virtex-E FPGA for enabling columnar partial-reconfiguration is treated in [2].

Reconfiguration interfaces, modules and tools have been discussed in [8] [18].

5. Conclusions and Future Work

Using our ROCCC compiler infrastructure, Xilinx Logicore IP Core library, we have been able to effectively configure a co-processor on the FPGA using a C function call in software, thus accelerating software using IP Cores. With the partial reconfiguration flow for FPGAs, we have effectively shared the reconfigurable fabric among various IP Cores, to overcome area limitation on CSoCs.

We plan to deal with the reconfiguration latency by incorporating a parallel reconfiguration engine on the FPGA. Since certain CPU cycles are always required at the initiation and the conclusion of an executing task during a context switch by the operating system, the time-penalty for re-configuration of the FPGA fabric can be minimized by parallelizing the reconfiguration with the context switch.

6. References

[1] D. Pellerin, G. Edvenson, K. Shenoy, D Isaacs, "Accelerating PowerPC Software Applications", Xilinx Xcell Embedded Magazine.

[2] E L. Horta, J W. Lockwood, "PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays FPGAs", WUCS-01-13.

[3] E. Filippi, L. Lavagno, L. Licciardi, A. Montanaro, M. Paolini, R. Passerone, M. Sgroi, A. Sangiovanni-Vincentelli "Intellectual Property Re-use in Embedded System Co-design: an Industrial Case Study," in Int. Symp. On System Synthesis (ISSS 1998).

[4] Mardav Wala, Don Bouldin, "Integrating and Verifying Intellectual Property Blocks using Platform Express and ModelSim", MWSCAS05.

[5] L. Shannon, "Impact of Intellectual Property Cores on Field Programmable Gate Array Designs", MS Thesis, Univ of Toronto

[6] http://www.vsi.org, VSIA, Virtual Socket Interface Association.

[7] Signal Proc. IP Cores, COTS Journal 09/2003, pp65-70, www.cotsjournalonline.com/pdfs/2003/09/cots09_techfocus1.pdf

[8] H Tan, R F. De Mara, A J. Thakkar, A Ejnioui, J D Sattler , "Complexity and Performance Tradeoffs with FPGA Partial Reconfiguration Interfaces" , Proceedings of RAW06.

[9] Xilinx, Inc., Two Flows for Partial Reconfiguration: Module Based or Difference Based, v1.1, Nov. 2003.

[10] N. Dorairaj, E. Shiflet, M. Goosman, "PlanAhead Software as a Platform for Partial Reconfiguration", Xilinx XCELL Journal, Art 55.

[11] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P Sundararajan, "A Self-reconfiguring Platform", Int. Conference on Field Programmable Logic and Applications (FPL 2003).

[12] Virtex-4 Multi Platform FPGA, http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/

[13] Xilinx Intellectual Property library, Logicore, http://www.xilinx.com/ipcenter/

[14] SUIF Compiler System. http://suif.stanford.edu, 2006

[15] Machine-SUIF.2006 http://www.eecs.harvard.edu/hube/research/machsuif.html

[16] Z. Guo, B. Buyukkurt, W. Najjar and K. Vissers. Optimized Generation of Data-path from C Codes for FPGAs, Int. ACM/IEEE Design, Automation and Test in Europe Conference (DATE 2005). Munich, Germany, March, 2005.

[17] J. Tripp, K. Peterson, C. Ahrens, J. Poznanovic, M. Gokhale. Trident: An FPGA Compiler Framework for Floating-Point Algorithms, Int. Conference on Field Programmable Logic and Applications (FPL 2005). Finland, 2005

[18] Michael Barr, "A Reconfigurable Computing Primer," Multimedia Systems Design, Sep. 1998, pp. 44 – 47.

[19] Xilinx ISE 8.1i Development System Reference Guide, pp130-140.

[20] R. Lysecky and F. Vahid. Pre-fetching for Improved Bus Wrapper Performance in Cores, ACM Transactions on Design Automation of Electronic Systems, Vol. 7, No. 1, pp. 58-90, January 2002.

[21] SPIRIT consortium, http://www.spiritconsortium.com/

[22] Tien-Lung Lee, Neil W. Bergmann, "Interfacing methodologies for IP re-use in Reconfigurable System on-Chip", SPIE International Symposium on Microelectronics, MEMS and Nanotechnology, Perth Australia, 12 / 2003

[23] EDK, PowerPC 405 Processor Block Reference Guide, UG018, http://www.xilinx.com/bvdocs/userguides/ug018.pdf

[24] A. Ansari, P. Ryser, D. Isaacs, Accelerated System Performance with APU Enhanced Processing, http://www.xilinx.com/publications/xcellonline/xcell_52/xc_v4acu52.htm

[25] Xilinx Fast Simplex Link v2.00a http://www.xilinx.com/bvdocs/ipcenter/data_sheet/FSL_V20.pdf

[26] Xilinx Floating point Operator v2.0, Logicore, http://www.xilinx.com/bvdocs/ipcenter/data_sheet/floating_point_ds33[5.pdf

[27] Xilinx PLB IPIF specifications DS414 http://www.xilinx.com/bvdocs/ipcenter/data_sheet/plb_ipif.pdf

[28] Xilinx OPB IPIF specifications DS414 http://www.xilinx.com/bvdocs/ipcenter/data_sheet/opb_ipif.pdf

[29] Tien-Lung Lee and Neil Bergmann. An Interface Methodology for Retargettable FPGA Peripherals. In Engineering of Reconfigurable Systems and Algorithms (ERSA), July 2003, pages 1-7.

[30] G. Holloway and M. D. Smith. Machine SUIF Control Flow Graph Library. Division of Engineering and Applied Sciences, Harvard University 2002.

[31] G. Holloway and A. Dimock. The Machine SUIF Bit-Vector Data-Flow-Analysis Library. Division of Engineering and Applied Sciences, Harvard University 2002.

[32] G. Holloway. The Machine-SUIF Static Single Assignment Library. Division of Engineering and Applied Sciences, Harvard University 2002.

[33] Xilinx CORDIC 3.0, DS 239 http://www.xilinx.com/bvdocs/ipcenter/data_sheet/cordic.pdf

[34] Xilinx Pipelined Divider v 3.0, DS305 http://www.xilinx.com/ipcenter/catalog/logicore/docs/sdivider.pdf

[35] Xilinx FFT v3.2, DS 260 http://www.xilinx.com/bvdocs/ipcenter/data_sheet/xfft.pdf