Rapid C to FPGA Prototyping with Multithreaded Emulation Engine

Shin-Kai Chen, *Bing-Shiun Wang*, *Tay-Jyi Lin*, and *Chih-Wei Liu*

Department of Electronics Engineering
National Chiao Tung University, Taiwan

Abstract – **FPGA prototyping is preferred over software simulations for its more convincing & realistic behaviours and fast simulation time. However, it is usually possible after the RTL design is done, which prevents extensive design space exploration. This paper describes an early-stage FPGA prototyping flow, which starts from C sources, through hardware/software partitioning with transaction-level modelling (TLM), to the RTL design. We also propose a FPGA-customized multithreaded emulation engine for TLM prototyping. Compared with the OpenRISC core, the proposed engine saves 43.08% datapath complexity while improving the operating frequency by 60.67%. Moreover, our FPGA prototype for JPEG at TLM can compress 37.16 color QCIF frames per second, which is 4.5X faster than SystemC simulation on a 3GHz PentiumD PC.**

I. INTRODUCTION

Recent advances in fabrication allow engineers to design large amount of transistors in a single chip. More and more functions are integrated together to support novel applications. Large-scale designs lead to long develop time and bring difficulties to verification. The poor test speed of traditional software simulation is really annoying to designers. FPGA are generally used for system prototyping. It provides high speed functional verification with accurate hardware behavior, gives designer more confidence to the products. With this useful tool, bugs in the design are eliminated before costly silicon implementation. Unfortunately, FPGA prototyping is only possible after the RTL design implementation. It takes a long time for RTL construction from C model. The lacks of hardware verification system makes the construction inefficient. In this paper, we propose a rapid prototyping system using a multithreaded emulation engine. The emulation engine is designed and optimized on FPGA, provides high speed TLM modeling. A simulation kernel is built on the multithreaded emulation engine. The prototyping system enables FPGA prototyping at very early stages, even when hardware/software partition has not been done. Besides concept proving and demonstration, the rapid prototyping can further migrate to RTL implementation following original design flow.

The rest of this paper is organized as follows. Section 2 introduces the C to FPGA design flow and some related work. Section 3 presents the proposed rapid prototyping scheme, multithreaded emulation engine, and simulation kernel. A JPEG encoder example and some comparisons between multithreaded emulation engine and an open source processor, OpenRISC, will appear in Section 4 to demonstrate the prototyping system. Finally, Section 5 concludes this paper.

II. PRELIMINARY

C language was developed in the early 1970s, and has become the most familiar language to software programmers. Traditionally, application implementations generally start from C model as figure 1. Relying on experienced engineers, the application is directly mapped into RTL model, which contains all the implementation information. With CAD tools, Silicon implementation can be obtained from RTL model automatically.

This research was supported in part by the National Science Council under Grant NSC95-2220-E009-013 and the Ministry of Economic Affairs under Grant 96-EC-17-A-01-S1-034.

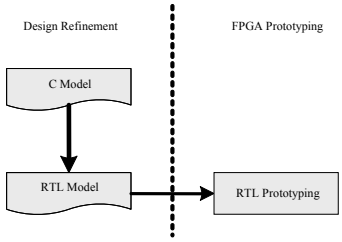

Fig. 1 Conventional design flow

The C language is originally designed for programmable processors, suggesting sequential execution, flattened share memory, and implicit data communication while hardware owns concurrency, complex memory layout, and explicit data transfer. These inconsistencies have led to huge gap between C and hardware, making hardware implementation a difficult job, especially for today's large scale designs. To overcome the gap, several related works propose additional abstraction levels between C and hardware to smooth the design flow. SystemC [1] and Handel-C [2] provide TLM abstraction to address the problem. Figure 2 shows the TLM flow. First of all, concurrency and interconnection are extracted from C model using processes and channels to form untimed TLM model. Hardware/Software partition are then performed, refine untimed TLM model to bus functional model. Concurrency explorations can be performed here in this TLM abstraction. The components in TLM model are then mapped to RTL individually. Finally, silicon implementation is achieved through RTL model. The TLM abstraction bridges C and hardware, smoothes design flow, and also provides opportunities for automation.

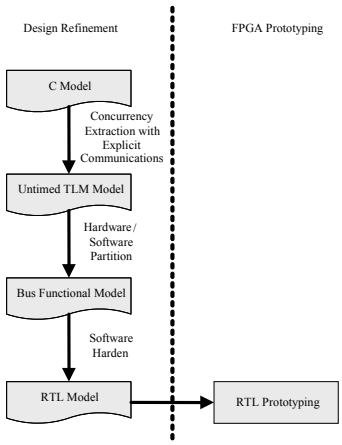

Fig. 2 Modern SoC design flow

Normally, FPGA prototyping can be obtained using RTL model for function verification and concept proving. It helps designers to avoid expensive failure in silicon implementation, and demonstrate the circuit earlier. Some researches [3][4] have noticed the requirement for rapid prototyping. Most of them follow above design flow. They start from well-defined synthesizable TLM model, rely on tools to generate RTL or

FPGA netlist automatically for prototyping. Unfortunately, synthesizable TLM model construction is still difficult and time consuming. Also, the automation is not complete yet; several steps still require manual assistances.

III. PROPOSED METHOD

In this chapter, we'll introduce proposed rapid prototyping flow, multithreaded emulation engine optimized for FPGA, and simulation kernel.

A. Rapid prototyping flow

The proposed flow is as figure 3. Similar to the TLM flow, the proposed flow introduce TLM models, which comprises tasks and queues. Tasks represent concurrency while queues establish data communication between them.

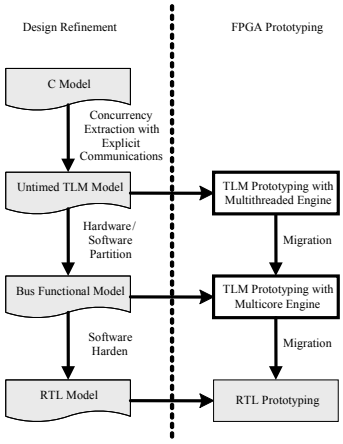

Fig. 3 Refined design flow with proposed early-stage FPGA prototyping

The task/queue model is built using simple C language. The structure of a task is restricted as figure 4. Inside the infinite loop, function of the task is assigned between the two APIs, which read input from input queue and write output to output queue. The queues are simply FIFO buffers which expose data communication between tasks explicitly.

```
task()
{
    // Variable Declaration
    for(;;)
    {
        Read_Input_Queue();
        // Task Code
        Write_Output_Queue();
    }
{
```

Fig. 4 Task structure

Using above structure, the task/queue model can be easily built without implementation details. The rapid prototyping can be quickly realized using this model with multithreaded emulation engine. After hardware/software partition, multi-core engine will take the responsibility for bus functional model prototyping.

These emulation engines are not only for TLM prototyping, but are applied during the software harden step toward RTL construction. When some components are already fulfilled by RTL, it can be downloaded onto FPGA, co-simulate with original prototyping system. Keeping move components out of prototyping system to RTL implementation, the prototyping system will naturally migrate to conventional RTL prototyping system as complete RTL model is constructed.

B. Multithreaded emulation engine

The multithreaded emulation engine is basically a 4-thread programmable RISC processor. The instruction set architecture is MIPS-compatible [5], simple and compiler-friendly. It essentially follows classical 5-stage pipeline [6] – instruction fetch, instruction decode, execution, memory access and write back, provides 32 32-bit general purpose registers for each thread. As the shrinking in technology, interconnection become critical issue for design. This problem already affects FPGA implementation seriously for there is poor routing resource in FPGA. Global routing are usually threats to performance. In the pipelined processor, global interconnection normally comes from forwarding and interlocking. These circuits are set for possible data hazard between instructions, which gather information from several pipeline stages and send control signals back.

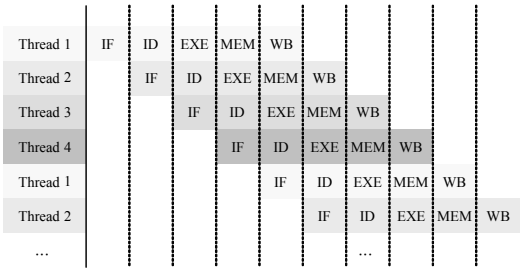

Fig. 5 Interleaved multithreading of 4 hardware threads on a classical 5-stage pipeline

Multithreaded processors support multiple instruction streams on the same hardware. To achieve this, multiple hardware content should be maintained in the machine. Multithreaded machines expose thread-level parallelism, provides latency tolerance. To address the interconnection issue, the emulation engine takes interleaved multithreaded strategy. In this machine, 4 independent threads are executed simultaneously using single datapath. One PC register and register file are required to hold the context for each thread. As figure 4, the 4 threads are issued interleavingly, thus, no data dependencies would exist between adjacent instructions reside right in the datapath anytime. In this case, forwarding and interlocking unit can be discarded, which greatly alleviates pressure from global routing. Besides, interleaved multithreaded architecture provides numerous optimization opportunities. Every instruction include branch and jump can have 3 cycle latencies now. Calculation performed in the execution stage can now extend into the memory access stage, which shortens the critical path in the execution stage. Such optimization burst clock rate of the multithreaded emulation engine while no overhead occurs in software development.

Interrupt mechanisms are implemented only on the first thread, called system thread. Tasks running on the system thread may be interrupted for system services. In contrast, other threads are referred as computation thread, and they are QoS (Quality of Service) guaranteed. System thread responds to outward stimulus as general purpose processor. Once an exception occurs, the ongoing task is suspended. After context switch, an interrupt service routine comes for event handling. Computation threads have deterministic behavior. Once a task is assigned on computation thread, the task will be guaranteed to finish at a

predictable time. Context switches will never take place in these threads regardless of any interrupt. The QoS property is useful for real-time system. Time critical tasks can be assigned to computation threads, which concentrate on computation. Interactive tasks like operating system can be processed on the system thread, which support complete interrupt handling. Using the deterministic property of computation threads, real-time scheduler can be implemented on the system thread, master computation thread, form a real-time guaranteed system.

C. Simulation kernel

The multithreaded emulation engine can only provide up to 4 time-multiplexing concurrencies by hardware techniques. Unfortunately, there are generally more than 4 tasks in applications. A multitasking simulation kernel brings more concurrency and flexibility to the emulation engine. In this multitasking kernel, the 4 hardware threads are used in different manner. The system thread in the emulation engine is used for system management as the others perform computation tasks. The system thread takes responsibility for resource management, task scheduling, task dispatching, and data communication. It takes control of other computation threads. Some hardware circuit may be added into the system as hardware threads. Once the system starts, a ready list is maintained in the kernel. If there are any computation threads or hardware threads idling, the kernel will assign an appropriate task from ready list. In the system, all data communication is processed in system thread as software FIFO queue. Using communication APIs, interconnections between tasks are easily fulfilled by software mechanism.

D. Migration

The prototyping system helps migration from untimed TLM model to RTL model. After hardware/software partition, data communications in untimed TLM model are replaced with hardware queues. In bus functional model, tasks are transferred from multithreaded emulation engine to individual processor. As each task is fulfilled using RTL, the processor is replaced by physical function blocks. Finally, RTL prototyping will be seamlessly constructed.

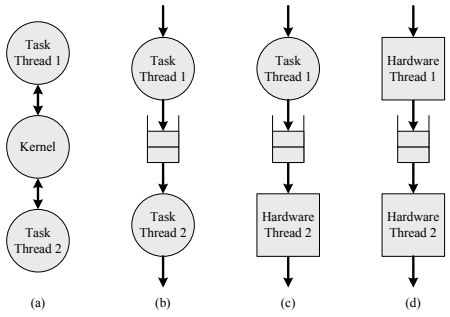

Fig. 6 Design migration: (a) Untimed TLM model (b) Bus functional model (c) One task migrates to hardware (d) A

Figure 6 shows an example for migration. The example system contains 2 tasks, and there's data communication from task 1 to task 2. Initially, untimed TLM model of 6(a) is built; both tasks are executed in computation threads on emulation engine, data transfer between tasks are performed through the kernel. 6(b) shows bus functional model, data communication is replaced by hardware queue here, and both tasks are performed by programmable processors. After task 2 is implemented in RTL, the prototyping system becomes 6(c). After all tasks are translated into RTL, the emulation engines are eliminated eventually like 6(d).

IV. Experimental Results

In this section, a JPEG encoder will be used as a design example to demonstrate the proposed rapid prototyping flow. Also, comparisons between the multithreaded emulation engine and OpenRISC core will present here.

A. JPEG Encoder

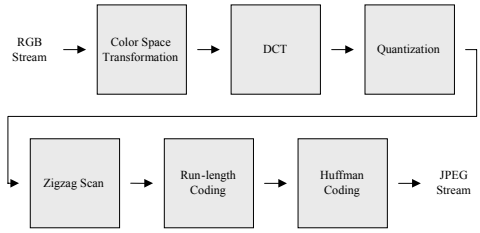

Fig. 7 JPEG encoder

JPEG [7] is a well-known standard, and it's broadly used around the world for picture compression. The flow of JPEG encoder is as Figure 7. It takes RGB data as input, and output JPEG stream. Data communication among the system is based on a macroblock, which comprises 16x16 pixels. After receiving input, a color space transformation translates RGB input to YCbCr data by 3x3 matrix multiplication. YCbCr is an orthogonal domain suitable for compression. DCT further transform the data from time domain to frequency domain, most information now concentrate to lower frequency. During the quantization, coefficients shrink to smaller value. Zigzag scan records the coefficients from low frequency to high frequency. Run-length coding pacts the resulting streams into value and length of zero run. Finally, Huffman coding produces JPEG streams after several look-up tables. Modifying JPEG encoder into TLM model using proposed task/queue model, the result is shown in figure 8. In the JPEG encoder flow, quantization, zigzag scan, and run-length coding acquire little computation, and the data communication between them is consistent. Thus these steps are combined in a single task. Data communication between tasks is modeled by FIFO queues.

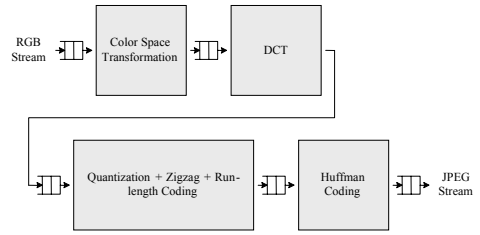

Fig. 8 The TLM model for JPEG encoder

Using the multithreaded emulation engine, the above TLM model for JPEG encoder can be quickly prototyped on FPGA platform. We set up software simulation and hardware prototyping environments for C, TLM, and RTL JPEG encoders to process QCIF image. Software simulations are realized on a Pentium D machine operating at 3 GHz. The TLM model simulation follows SystemC's simulation framework, and RTL simulation is done by ModelSim SE simulator. Hardware prototyping systems are implemented on a Xilinx Virtex II 6000 [8] FPGA board, which can sustain large scale designs. The FPGA board only supports 35 MHz clock signal, logic must operate at multiple of 35 MHz on this platform. A 70 MHz RISC processor is used for C prototyping. This RISC processor has the same ISA with the multithreaded emulation engine, it implements forwarding unit and interlocking mechanism for

data hazard. TLM model is prototyped by proposed multithreaded emulation engine which can run 140 MHz on this FPGA board. A JPEG encoder RTL is directly downloaded on FPGA, operating at 35 MHz. The result is listed below in Table I. Software simulation only shows advantage on C modeling, while hardware prototyping provides high speed verification on TLM and RTL modeling. As the design abstraction goes down following modern SoC design flow, software simulation provides poor simulation speed. The RTL simulation takes more than 3 minutes to encode a QCIF frame. Hardware prototyping system can't get up with software simulation for C model, but the processing speed improves as designs are modified toward real hardware. In the intermediate abstraction, the TLM layer, the JPEG encoder generates 37.162 QCIF JPEG frames per second on our prototyping system, while the software SystemC simulator can only offer 8.253 QCIF frames per second.

Table I PERFORMANCE EVALUATION

	Model	Target Platform	Simulation Speed (FPS)
Software Simulation	C	Pentium D@ 3 GHz	77. 576
	TLM		8. 253
	RTL		0. 005
FPGA Prototyping	C	Classical5-stage RISC Processor@ 70 MHz	23. 728
	TLM	Multithreaded Emulation Engine@ 140 MHz	37. 162
	RTL	Direct Implementation on FPGA@ 35 MHz	84. 250

B. FPGA implementation

OpenCores is a repository for open source, freely available IP (Intellectual Property) core. They hold projects to build IPs for various purposes. The most famous project among them is the programmable processor cores, OpenRISC [9]. OpenRISC is basically a single-threaded 32-bit RISC processor which also has a MIPS-compatible instruction set architecture. Many properties including register count can be adjusted according to usage. OpenRISC is programmed and optimized by people from world wide around for it's an open source project, everyone can contribute to the OpenRISC project. We port this free core on Xilinx Virtex II 6000 using Xilinx ISE v6.2, with speed grade -6, and all optimization effort levels are set to high, the same configuration as proposed multithreaded emulation engine.

Table II shows the difference between the two processors. The proposed engine provides 4 thread as OpenRISC is single-threaded. Among the 4 threads, at least 3 computation threads are utilized for computation. In OpenRISC, Instructions are pipelined to different number of stages. Computation operations are pipelined into 4 stages, additional one stage is required by memory access operations. Register files are not included in the datapath complexity comparison, because the resources are not compatible between the two processors. The OpenRISC processor consumes more area for those interconnections-centric forwarding and interlocking unit. Also, the critical path starts from write back stage, through forwarding unit, back to execution stage. The elimination of global interconnections brings revenue to multithreaded emulation engine, with 43.08% profit in complexity, and gain 60.67% performance.

Table II OPENRISC V.S. MULTITHREADED EMULATION ENGINE

	OpenRISC	Multithreaded Emulation Engine
# Hardware Threads	1	3 + 1
# Pipeline Stages	4 ~ 5	5
Functional Unit	32-bit ALU	
# Registers	32*32-bit	4*32*32-bit
Datapath Complexity	1,300 CLBs	740 CLBs
Operating Frequency	89 MHz	143 MHz

V. CONCLUSION

In this paper, we propose a processor-based system on FPGA to provide hardware prototyping. The proposed system provides FPGA prototyping in the early design stage, even before hardware/software partition has not been done. This rapid prototyping system supports high speed verification, much faster than traditional software simulation on desktop machines. Multithreaded emulation engine and simulation kernel form the main framework. The multithreaded emulation engine is a 4-thread RISC processor with MIPS-compatible ISA. Thanks to interleaving multithreading, the datapath of emulation engine gets rid of global interconnection, and is optimized for FPGA. Compared with OpenRISC, a well-known open source processor, it runs 60.67% faster using about half hardware resource. In addition to concept proving and demonstration for TLM model, the system can further help co-verification of numerous modules from different abstraction layer. Following the TLM flow, this proposed system can seamlessly migrate to RTL FPGA prototyping.

REFERENCES

[1] *SystemC Version 2.0 User's Guide*, Open SystemC Initiative (OSCI), 2001.

[2] *Handel-C Language Reference Manul*, Celoxica Corp., 2003.

[3] W. Klingauf and R. Gunzel, "From TLM to FPGA: rapid prototyping with SystemC and transaction level modeling," in *Proc. ICFPT*, 2005, pp. 285-286.

[4] M. Vasilko, L. Machacek, M. Matej, and S. Holloway, "A rapid prototyping methodology and platform for seamless communication systems," in *Proc. RSP*, Jun. 2001, pp. 70-76.

[5] D. Sweetman, *See MIPS Run*, 1st Edition, Morgan Kaufmann, 2002.

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, 3rd Edition, Morgan Kaufmann, 2002.

[7] W.B. Pennebarker and J. L. Mitchell, *JPEG Still Image Compression Standard*, Van Nostrand Reinhold, 1993.

[8] *Virtex-II Platform FPGA User Guide*, Xilinx Inc., Mar. 23, 2003.

[9] D. Lampret, *OpenRISC 1200 IP Core Specification*. Rev. 0.7, Preliminary Draft. OpenCores, Sep. 6, 2001.

[10] S. A. Edwardws, "The challenges of hardware synthesis from C-like languages," in *Proc. DATE,* 2005, pp. 66-67.

[11] M. Stoian, and G. Stefan, "A multithreading architecture for low power processors," in *Proc. CAS*, 2006, pp. 387-390.

[12] J. J. Labrosse, *MicroC/OS-II The Real-Time Kernel*, 1st Edition, CMP Books, 1999.

[13] T. Grotker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Kluwer, 2002.