

Preemptive Multitasking on FPGAs*

L. Levinson[‡], R. Männer[†], M. Sessler[†], H. Simmler[†]

[†]University of Mannheim, B6, 26; 68131 Mannheim, Germany

[‡]Weizmann Institute of Science, Rehovot, Israel 76100

Abstract

In exploring the efficacy of reconfigurable computing, one of the dimensions is the possibility for multitasking on an FPGA-based processor. Conventional computers and operating systems have demonstrated the many advantages of sharing computational hardware by several tasks over time. The ability to do run-time configuration and readback of FPGAs in a coprocessor architecture enables exploring the problems of implementing realistic multitasking.

1 Introduction

FPGAs for custom computing machines have shown remarkable speedups for several classes of algorithms in the past years. Reasons for high speedups can be seen in deep pipeline stages and the usage of parallelism for the algorithm execution.

One ongoing field of research is the run time reconfiguration (RTR) of FPGAs. Most RTR approaches use FPGA coprocessors like Pamette or microEnable [1] as a base platform, due to the tight coupling needed for the FPGA control. RTR makes use of the reconfigurability of the FPGAs. Algorithms which use more FPGA resources than available can be split into subparts which are then executed sequentially [2]. Another possibility is to execute several algorithms, that do not require all FPGA resources, in parallel in one FPGA, i.e. true multitasking [3]. Due to the multiple usage of the CPU-to-FPGA coprocessor connection for the parallel executed tasks, the overall performance for each task is reduced and logic must be added to each task to arbitrate the shared resource. A third possibility is the execution of several tasks with a batch operating system [4]. Each task is executed completely, before the next scheduled task is prepared for execution. This method provides exclusive access to any external device and the full CPU-to-FPGA communication bandwidth.

The cited RTR approaches used either total or partial overlay techniques to manage the tasks. However long tasks may block execution for minutes or even hours. A task manager must be able to suspend the ongoing execution of one task to avoid these blocking sit-

uations. Such a preemptive multitasking environment works like modern operating systems and must be able to extract and reconstruct the status of the FPGA designs for each task swap.

2 Task Switching

Essential for implementing multitasking on FPGA coprocessors is the ability to suspend the execution of an ongoing task and to restore a previously interrupted task.

Such a task switch can be compared to a task switch on modern CPU's where only some specific registers are saved during task switch. These registers describe the current process status and are used for task reconstruction. In case of an Intel Pentium II 104 bytes must be saved and restored.

A FPGA design normally does not have a code or data descriptor like a CPU. Rather, an FPGA holds data in several registers scattered over the FPGA. Therefore, all used registers, latches and internal memory of an FPGA design must be saved to enable later restoration of the task. This can require up to 350kbytes for a modern FPGA device such as the Xilinx XCV1000.

2.1 Requirements

Preemptive multitasking on FPGAs requires some features of the FPGA, the coprocessor board and the FPGA designs. These features are listed below:

- The FPGA and the board must provide a configuration and readback ability. The readback bitstream is used to determine the status and must contain the status of all internal registers and RAMs.
- All registers and internal memory bits must be either preset or reset after configuration. This initial state control allows to restore the current state of the task.
- There has to be a complete clock control on the board. Stopping the clock allows freezing the task at a current state.

*Work supported by the German-Israeli Foundation for Scientific Research and Development

- Multiple clocks are constrained to be integer multiples of the slowest clock and they have to be in phase. Tasks must be interrupted only on the slowest clock to ensure that no clocks are lost and all relative timings are preserved.
- FPGA design signaling to indicate secure switching times. This is necessary when external RAM is used where the address and the corresponding data are transferred in different clock cycles (see Figure 1).

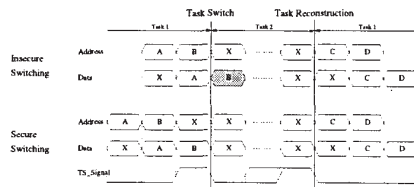


Figure 1: External RAM access during a task switch.

2.2 Design State Extraction and Reconstruction

State extraction of a FPGA design and reconstruction of this state are the two key features to enable task switching on FPGAs.

State extraction of a stopped FPGA design is done by filtering all status information bits out of the readback bitstream. The readback bitstream contains information about the configuration and the status of the internal registers or RAMs. For the state extraction only these state bits are important. This selection reduces the amount of bits that have to be stored by $\approx 90\%$ ¹. In order to extract these state bits, their bit positions within the readback stream must be known.

Task reconstruction is done by correctly setting the state of each single register and each RAM bit in the configuration bitstream. This is done by directly manipulating the configuration bitstream to avoid the lengthy place and route procedure. This can be done because a task switch does not change any logic functionality or connections in the FPGA design and the initialization information is directly coded by single bits in the configuration bitstream. All bit positions for each initialization bit in the bitstream must also be known to enable this direct and fast manipulation of the initialization states.

3 Client Server Model

All the state extraction and bitstream manipulation must be done by a central **Hardware Management**

¹ For the Xilinx XCV300 device and 91% for the XCV1000.

Unit (HMU) in the host software system. A client-server model including this HMU was built for proof-of-principle. The implementation was done for WinNT and uses the microEnable FPGA coprocessor with a XC4028EX device.

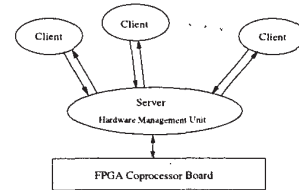


Figure 2: Client-Server model architecture.

The multithreaded server handles all communication between the clients and the hardware and manages the extraction and restoration of each task switch. The possibility of task switching was successfully shown with this client-server model for two exemplary designs. Design extraction and state reconstruction for the complete FPGA was performed in 18ms and 13ms respectively on a 166MHz Pentium.

4 Conclusion

The client-server model runs successfully and shows that task interruption and reconstruction can be done with FPGAs to enable task switching. For a fast task switch time configuration and readback have to be as fast as possible and any memory swapping has to be avoided. These are the most essential features for a multitasking FPGA board.

These and the other mentioned requirements are fulfilled by a new coprocessor board that provides hardware features especially for supporting task switching. This board is currently under development and results will be presented soon.

References

- [1] Optimagic WebSite: Board List, 2000.
<http://www.optimagic.com/boards.html>
- [2] R. Hudson, et.al.: A Run-Time Reconfigurable Engine for Image Interpolation, IEEE Symposium on FPGAs for CCMs, 1998. Page 88-95.
- [3] G. Brebner: The Swappable Logic Unit: A Paradigm for Virtual Hardware, IEEE Symposium on FPGAs for CCMs, 1997. Pages 77-86.
- [4] J. Jean et.al.: Dynamic Reconfiguration to Support Concurrent Applications, IEEE Symposium on FPGAs for CCMs, 1998. Pages 302-303.