Methodology and Experimental Setup for the Determination of System-level Dynamic Reconfiguration Overhead

Kyprianos Papadimitriou [*] Antonis Anyfantis Apostolos Dollas
Department of Electronic and Computer Engineering
Technical University of Crete
GR73100 Chania, Crete, Greece
{kpapadim, aanyfantis, dollas}@mhl.tuc.gr

1. Introduction

Dynamic reconfiguration is gaining popularity [2], [4] but it may cause degradation of overall execution time due to the time to download the bitstream before an application starts execution of the new configuration. Thus evaluation of its performance becomes an interesting area [3]. In this work we include an analysis of the reconfiguration time by defining the delays that add up to it. An experimental setup is deployed that can be used for performance evaluation of applications implemented with dynamic reconfiguration, as well as of mechanisms developed to reduce reconfiguration overhead.

The configuration memory of Xilinx Virtex-II is arranged in vertical frames. In a self-reconfigurable system [1] where the PowerPC reconfigures the FPGA through the Internal Configuration Access Port (ICAP), once data are available in the configuration cache HWICAP BRAM the nominal time to reconfigure a single frame is 12.5 μs for 66 MHz. However, this is not the only aspect in the reconfiguration process. Several physical components of the system add significant delays causing reconfiguration time to increase more than three orders of magnitude as compared to the above time.

2. Experimental Setup

The setup consists of a Digilent XUPV2P platform, a logic analyzer and a PC for evaluation. During the first phase of experimentation, two configuration partial bitstreams are placed in a compact flash memory and a PowerPC processor loads them to the Virtex-II XC2VP30 FPGA according to a user input. The ACE Controller supervises the transfer of data from the compact flash to the FPGA. The processor and several peripherals have been configured in the FPGA. A push button fires partial reconfiguration at any time during operation. Four dip switches control the functionality of an FPGA peripheral. A UART sends status messages to the PC. A few signals of the FPGA peripherals along with some other internal signals are monitored with the logic analyzer.

[*]Funded with a Ph.D fellowship by the Greek Ministry of National Education and Religious Affairs under the program Heraklitus, EPEAEK II, grant number 88727/11

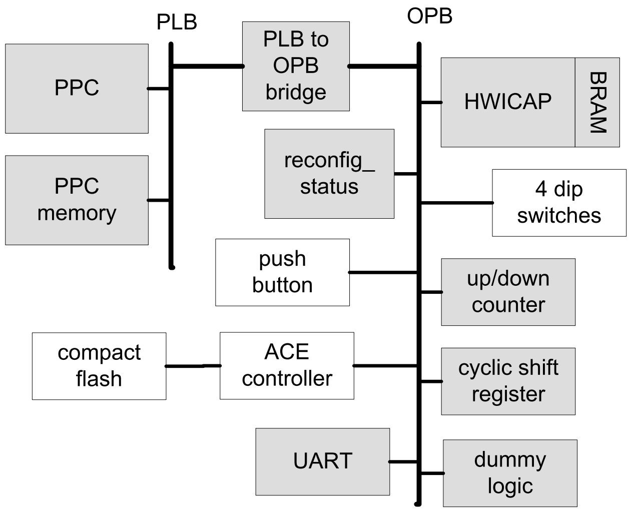

Figure 1. Shadowed boxes represent internal components of the FPGA system. White boxes are the parts of the XUP platform that are connected to the FPGA.

2.1. The FPGA System

The FPGA system is illustrated in Figure 1. An up/down counter which can be partially reconfigured and a cyclic shift register which remains unchanged during operation have been implemented. Ten dummy_logic peripherals have been implemented as partially reconfigurable modules. The push button forces the processor to request a partial bitstream from the compact flash. Configuration data are then written in the PPC memory, and subsequently transmitted to the HWICAP BRAM. The latter writes the FPGA configuration memory. The above operation is due to the HWICAP API that needs a configuration data chunk to be loaded in processor memory before transfer to the HWICAP BRAM takes place. The reconfig_status indicates the duration of reconfiguration process. The dummy_logic slave peripherals are added in experiments of reconfiguration of more than one peripherals in which measurements with the logic analyzer were conducted.

2.2. Reconfiguration Time Breakdown

We define the following delays that add up to the reconfiguration time:

- CF-PPC: is the time to transfer one configuration data chunk from the compact flash (CF) to the PPC memory.
- PPC-HWICAP: is the time to transfer configuration data from the PPC memory to the HWICAP BRAM. The maximum size of the data transferred per transmission equals to one BRAM, i.e., 512 words.
- HWICAP BRAM-CM: is the time to transfer data of one HWICAP BRAM to the configuration memory of the FPGA.

These delays correspond to processor code functions and are measured with software timers. Their values are sent to the PC through a UART peripheral.

3. Conducting the Experiments

Several parameters of the processor were configured before entering the experimental phase. The processor memory was chosen to be 48 Kbytes, the stack was 6000 bytes and the buffer cache where the configuration data are loaded on, varied between 512 and 4096 bytes. The buffer cache is located in the processor memory and it acts like a normal cache where the data are loaded on prior processor main memory. We also varied the processor array size, i.e., processor memory array in which configuration data chunks are loaded from the compact flash, from 512 to 4096 bytes. In particular, transactions in units of several multiples of one sector were conducted per processor request. A sector is the smallest unit the compact flash is organized in, and is equal to 512 bytes.

4. Results and Analysis

Initially, we measured piece-wise delays during reconfiguration of one frame using the software timers. The corresponding bitstream size was 636 words. Table 1 contains software measurements for all buffer cache sizes up to 4096 bytes and all processor array sizes up to 4096 bytes. The measured delays do not correspond to the time to transmit the entire frame. The processor array size puts the upper limit on how many bytes are transferred per transaction. According to the definition of software measured delays, we measure delay per transaction.

Table 1. Piece-wise delays. Processor array size is in multiples of one sector. Size is in bytes and time in msecs

buffer cache	processor array	CF-PPC	PPC-HWICAP	HWICAP BRAM-CM
512-4096	1×512	1.5	0.42	0.02526
512-4096	2×512	2.94	0.83	0.02526
512-4096	3×512	4.35	1.25	0.02526
512-4096	4×512	5.79	1.67	0.02526
512-4096	5×512	7.24	1.67	0.02526
512-4096	6×512	7.24	1.67	0.02526
512-4096	7×512	7.24	1.67	0.02526
512-4096	8×512	7.24	1.67	0.02526

When the buffer cache was varied from 512 up to 4096 bytes in multiples of 512, and the processor array size was kept constant, no variation on CF-PPC and PPC-HWICAP delays was observed. By contrast, while we were varying processor array size, the delay increased up to the point the processor array size was set to 2560. This increase is normal as more data were allowed to be loaded per transaction. When the processor array size was set equal to or greater than 2560 no difference in the delays was recorded. This is because any processor array size above this value can accommodate the entire bitstream, which is equal to 2544 bytes. The PPC-HWICAP delay was stopped increasing after the processor array size was set to 2048. That is due to the HWICAP BRAM size which can not hold more than 2048 bytes. The HWICAP BRAM-CM delay was found to be 0.02526 ms.

In cases where the processor array is larger than the bitstream, the latter is transferred with one processor request and stored in the processor memory in its entirety prior transmission to the HWICAP. From the HWICAP part, configuration cache size is 2048 bytes and can not be changed, thus inhibiting transfer and accommodation of the entire bitstream at once. However, in our system HWICAP is not the bottleneck as the time for the PPC to load 2048 bytes to HWICAP BRAM is larger than the time the HWICAP needs to configure the FPGA.

References

[1] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan. A Self-reconfiguring Platform. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 565–574, 2003.

[2] C. Kachris and S. Vassiliadis. Performance Evaluation of an Adaptive FPGA for Network Applications. In *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping*, 2006.

[3] H. Tan, R. F. DeMara, A. J. Thakkar, A. Ejnioui, and J. D. Sattler. Complexity and Performance Evaluation of Two Partial Reconfiguration Interfaces on FPGAs: a Case Study. In *Proceedings of the ERSA*, 2006.

[4] Xilinx Inc. Press Release: ISR and Xilinx Roll Out Ready-to-Wear SDR. Xilinx Inc., San Jose, CA., 2006. www.fpgajournal.com.