

# Dynamic Reconfiguration to Support Concurrent Applications

Jack Jean<sup>1</sup>, Karen Tomko, Vikram Yavagal, Robert Cook, and Jignesh Shah

Department of Computer Science and Engineering  
Wright State University, Dayton, OH 45435  
{jjean, ktomko, vyavagal, bcook, jshah}@cs.wright.edu

## Abstract

The proposed dynamically reconfigurable system can support multiple applications running concurrently. An FPGA resource manager is developed to allocate and de-allocate FPGA resources and to pre-load FPGA configuration files. For each individual application, different tasks that require FPGA resources are represented as a flow graph which is made available to the resource manager so to enable efficient resource management and pre-loading. The impact of supporting concurrency and pre-loading in reducing application execution time is demonstrated.

## 1. Introduction

A dynamically reconfigurable system allows hardware reconfiguration while part of the reconfigurable hardware is busy computing. This paper describes the system software development of a dynamically reconfigurable system that can support multiple applications running concurrently. A block diagram illustrating such a system is shown in Figure 1 where each application consists of a program to be executed on the host machine and a flow graph to be executed on the FPGA resources. The host program is responsible for starting the execution of graph nodes through the resource manager. With the information of multiple flow graphs, one for each application, the resource manager allocates and de-allocates FPGA resources so that new nodes may be loaded into the system while other nodes are being executed. In addition, a speculative strategy is adopted by the resource manager in the “pre-loading” of FPGA configuration files so to reduce and hide the reconfiguration overhead. The FPGA architecture is *modular* in the sense that the FPGA resources consist of a number of hardware units and each graph node uses an integer number of hardware units. Because of the resource manager and the speculative loading policy, multiple applications may share the FPGA resources effectively, very much analogous to a virtual memory

<sup>1</sup> This research is supported by DARPA under Air Force contract number F33615-97-1-1148, an Ohio State investment fund, and an Ohio State research challenge grant. Xilinx Inc. donated an FPGA design tool and FPGA chips on XMODs.

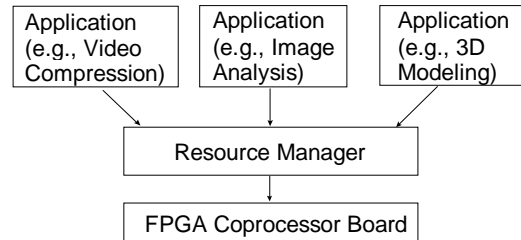


Figure 1 - The dynamic reconfiguration system

system. The RAGE project [1] is similar to our own, but emphasizes partial reconfiguration. It does not support pre-loading of configurations.

## 2. Resource Manager Design

The platform used is a 180 MHz Pentium-pro PC hosting a PCI-bus based G900 FPGA board manufactured by Giga Operations Corporation. The board contains eight XMODs where each XMOD contains two XC4020E FPGA chips, 8 MB DRAM, and 256KB SRAM. The resource manager uses sockets to communicate to individual applications. Internally it is a multi-threaded design with a main thread which basically performs initialization, listens on a socket, and spawns an application service thread whenever there is a new application running. In this system, the number of application service threads of the resource manager is the same as the number of applications. The main thread also spawns a scheduler thread to allocate XMODs either on demand or speculatively. Communication among multiple threads of the resource manager is based on shared memory.

## 3. Performance Results

To test the system operation a program was artificially synthesized with a flow graph of four nodes, each exhaustively solving an NP-complete satisfiability problem (see Figure 2). The flow graph with weighted edges is shown at the right hand side of Figure 3. A larger weight indicates a higher chance of using the node and only nodes 0 and 3 of the graph are really needed for computation. Two different sets of tests were performed.

**TEST 1: Running eight multiple-satisfiability applications with or without the resource manager**

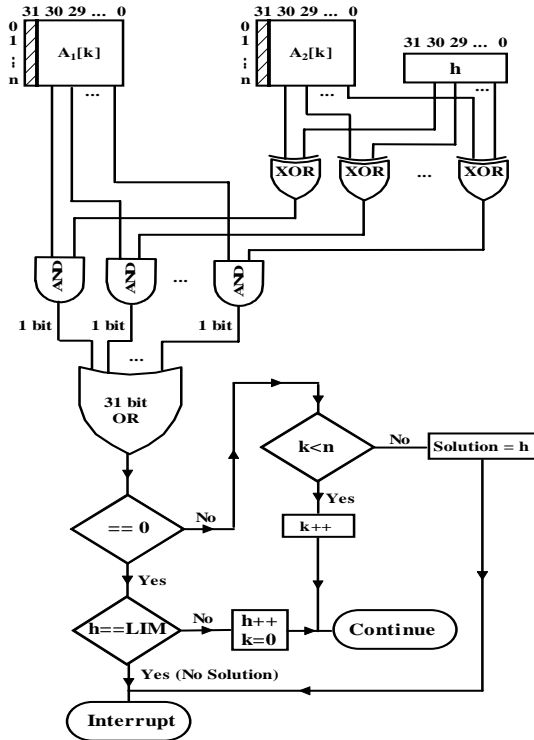


Figure 2 - Satisfiability FPGA design

When there is no resource manager, each application needs to initialize the board itself and in that initialization process each application gets to attach the PCI bus address space for the G900 board to its own address space. As a result the applications cannot run concurrently and have to run sequentially. The timing measurement was based on the average of three independent runs. The resource manager does not use pre-loading in this test.

Table 1 summarizes the average execution times for different cases in TEST 1. The overhead time includes the board initialization (in average takes about 2.31 seconds) and the loading of a configuration file (takes about 0.37 seconds). The speedup of the concurrent case versus the

	Total Time	Overhead Time	“Real” Part
Sequential (Without Resource Manager)	74.17	24.4	49.77
Sequential (With Resource Manager)	70.15	5.92	64.23
Concurrent (With Resource Manager)	11.23	5.92	5.31

Table 1 - Average Times for TEST 1 in seconds

no-resource-manager sequential case is 9.37 (= 49.77/5.31). The reason for the super-linear speedup is not clear since there are only eight XMODs.

**TEST 2: Running multiple-satisfiability applications concurrently on top of the resource manager with or without the speculative pre-loading**

The edge weights of the application flow graph was set up so that, if pre-loading occurs after the first graph node, the right one would be pre-loaded. Afterwards, if there are still extra XMODs, the other two nodes would be pre-loaded even though they would not be used at all. The computation time of each graph node is in the range of two to four seconds.

When there is only a single application, it took on average 8.77 seconds without the speculative pre-loading. With pre-loading, it took 8.44 seconds instead. Since the application needs to load two configuration files during the execution, the second file is apparently pre-loaded and the saving is 0.33 seconds.

When there are four concurrent applications, it took in average 9.30 seconds with the speculative pre-loading and 9.88 seconds without pre-loading. The detailed timing with pre-loading is shown in Figure 3 where the loading of node 0 and node 3 are labeled as L0 and L3, respectively.

**Reference**

[1] J. Burns, A. Donlin, J. Hogg, S. Singh, and M. Wit, “A Dynamic Reconfiguration Run-Time System,” in Proc. of FCCM, pp. 66-75, 1997.

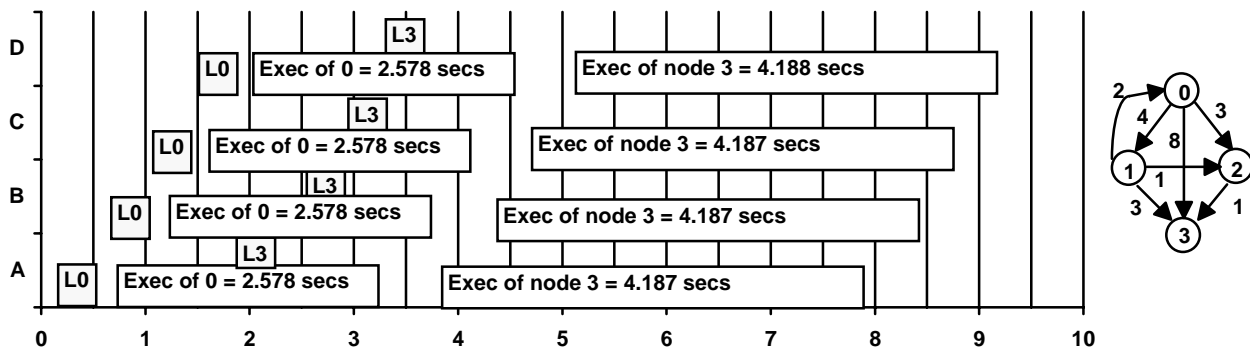


Figure 3 - Four concurrent applications, A, B, C, and D, running with pre-loading