

High-level Synthesis of Designs for Partially Reconfigurable FPGAs *

Satish Ganesan, Abhijit Ghosh, and Ranga Vemuri

Laboratory for Digital Design Environments
University of Cincinnati

Email: [satish](mailto:satish@ececs.uc.edu), [ranga](mailto:ranga@ececs.uc.edu)

Tel: 513 556 3025 Fax: 513 556 7326

1. Introduction

Partially reconfigurable devices provide a unique ability by which a part of the device can be reconfigured while other parts are still operational. It provides an exciting opportunity to reduce reconfiguration overhead by overlapping it with execution time. This feature enables designs that run on such hardware to gain considerable speed-up when compared to hardware that support only total reconfiguration. To take advantage of dynamic reconfiguration, a design environment for partially reconfigurable devices must integrate this capability into the CAD framework. In this paper, we present a CAD framework that accepts a behavioral specification, performs high-level synthesis and implements a methodology to capture the dynamic reconfiguration capability of the device. Logic elaboration is achieved using the commercial tool, VELAB and layout synthesis is done using the XACT6000 tool. A host controller manages the device and loads and removes blocks as required.

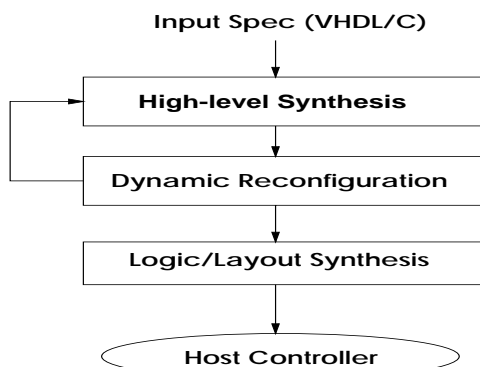


Figure 1: Tool flow

2. Input Specification

The system accepts a single-threaded behavioral description specified in a subset of either VHDL or C. This description is transformed by a front-end translator into a block graph format, which represents the input as a Control data flow graph (CDFG). Each block in the block graph is a DFG of a set of computations and the block hierarchy represents the control flow between the blocks. Branching of blocks occur when a conditional statement is encountered in the specification or a when there is a change in execution type from compute to IO or vice-versa. A huge block of straight line code can also be broken down into multiple blocks based on user specified break points. In the limiting case, each block will just perform a single operation. Currently, execution of loops is not supported by the framework.

3. High-level Synthesis with a Dynamic Reconfiguration Strategy

Each block of the block graph is separately synthesized into a structural VHDL description using a high-level synthesis tool, which is a part of the tool suite. The synthesized RTL design generates a *done* signal denoting the

*This work is supported in part by the US Air Force, Wright Laboratory, WPAFB, under contract number F33615-97-C-1043.

Design	Front end Translator	High-level Synthesis		Logic Synthesis	Layout Synthesis	
	Time (sec)	# blocks	Time (sec)	Time (sec)	# cells	Time (sec)
Image Smoothing	0.145	6	2.72	5.33	1016	301.3
Image Morphing	0.10	4	3.42	4.18	624	249.41

Table 1: Results for tool suite

end of execution, and a *start* signal for every block. The *done* signal will be used by the dynamic reconfiguration strategy as a control signal to enable block substitution. The *start* signal of a block indicates the start of execution of a block. Since the blocks are pre-synthesized and the next step in the design flow is logic elaboration using VELAB, the area of the block can be obtained from the RT level design.

Since the logic elaborator VELAB accepts only purely structural specifications, the outputs of the HLS tool are modified by scripts and other wrappers to make them compatible with VELAB. The synthesized blocks with their corresponding areas and the DFG structure of the block graph are then fed to the block substitution phase. The aim is to reduce the overall execution time by overlapping configuration time of a block with the execution time of another as much as possible. This is achieved with the help of a novel dynamic block substitution strategy. The output of this phase can be viewed as a sequence of temporal segments, each segment having atleast one block that is executing and one block that is being reconfigured. A temporal segment ends when the host receives a done signal from an executing block. The host can poll on any signal on the device because architectures like the XC6200 allow interaction of the host with any device register. For a single thread of execution, blocks get *done* in the order that they are loaded.

Initially, the first block is loaded and begins execution. A second block is configured while the first block is executing. The first temporal segment contains these two blocks. The first block upon completion generates a done signal. The host on receiving the done signal loads the next ready block, replacing the *done* block. In the case of a conditional block, the value evaluated by the condition is used to select one of it's children. Once a block is done, the next (second, in this case) temporal segment begins. Physical block substitution is done by a block-level placement tool. An intermediate register block is introduced to store the data output by the *done* block of the first temporal segment. The register block belongs to the second temporal segment and is not removed until the start of the third temporal segment. Meanwhile, once the second block has been configured, it waits for its start signal. When it receives the signal, it begins execution, reading it's inputs from the intermediate register block. When the reconfiguration time of the newly loaded block is comparable to the execution time of the replaced block, reconfiguration overhead is almost completely absorbed. In the worst case that block branching occurs only when conditions are evaluated, the performance is comparable to that of total reconfiguration.

The above substitution cycle is repeated until the entire design has been loaded. A single threaded specification simplifies the block substitution process to a great extent. However, this can be extended to a multi-threaded specification where the implementation becomes non-trivial.

5. Results

A few small examples (simplified and small versions of well-known algorithms) were specified in behavioral VHDL and taken through the tool suite down to the Wildfire RC in order to verify the design flow (no reconfiguration was performed). The WildFire RC consists of one XC6200 device. Results for this are presented in Table 1. The time taken for each step in the design flow for these examples for a single configuration of the device is provided. Additional examples are being synthesized and will be included in the final version of the paper.