The Case for Dynamic Execution on Dynamic Hardware

Charles Ross and Wim Böhm
{rossc,bohm}@cs.colostate.edu
Computer Science Dept., Colorado State University, Fort Collins, CO

1 Introduction

We present a dynamic dataflow execution model, called the Aggregated Hierarchical Abstract Hardware Architecture (or AHAHA), for use in FPGA based applications. High level language implementations targetting FPGAs use either handshaking or control-path solutions to schedule computations. Among the handshaking variety are the Compaan VHDL Visitor [1]; the control-path solution uses a simplified form of handshaking which operates only in the same direction as the data flows and is used for example in MAP-C[2].

The SA-C* compiler developed at Colorado State University uses the AHAHA as its target architecture [3, 4]. SA-C* streams provide a flexible communication mechanism which allows multiple loops to execute concurrently in a producer / consumer relationship. The AHAHA supports enqueue and dequeue operations on variable sized buffers. When a loop operates on input and output streams, it is called a "process".

In this paper, we focus on the handshaking protocol in AHAHA, and compare its features to static timing models. We measure the overhead of handshaking in a full application in terms of area utilization. We analyze the cost of handshaking versus its benefits and show that it is desirable, and worth the minimal overhead.

2 AHAHA

AHAHA is a dataflow graph structure describing reconfigurable computing applications generated from SA-C*. Each AHAHA node encapsulates a single operation described in a VHDL component which can be optimized for the specific FPGA being targeted. The execution of the graph is governed by a section-based handshaking protocol precluding the need for static timing analysis. AHAHA incorporates both stateless combinational "unclocked" operators and stateful time-dependent"clocked" operators. Combinational (unclocked) nodes already have executable semantics inherited from hardware circuits. The AHAHA scheduling paradigm involves only the stateful nodes. A "cluster" refers to a group of input ports or output ports of a node which will always consume or produce values together. Each port of a node is contained in exactly one cluster, and no cluster may consist of both input and output ports. In the AHAHA model, unlike typical dataflow systems, clocked nodes do not fire as a complete unit; rather, the individual clusters of a node may fire separately. A "section" is a connected region of an AHAHA graph which includes a set of producing clusters, a set of consuming clusters and all of the connected unclocked nodes which lie between the producers and consumers. Conceptually, a section contains clusters (and unclocked nodes) which must fire concurrently in order to correctly execute the graph. No edges in the graph may cross a boundary between sections. Sections are constructed using a connected components algorithm. An example of a sectioned AHAHA graph is shown in Figure 1. The dashed lines indicate section boundaries (and, consequently, cluster boundaries as well). Clocked nodes are represented as rectangles, and unclocked nodes as smaller rounded rectangles. "Handshaking" is used to synchronize the nodes, and implement the firing rules of the AHAHA. Each producing cluster of a node provides a signal to indicate that the outputs in that cluster are valid and usable. Likewise, each consuming cluster of a node provides a signal which indicates that its inputs are ready to accept data. All these signals are combined with an AND gate to produce the **AllReady** signal for that section.

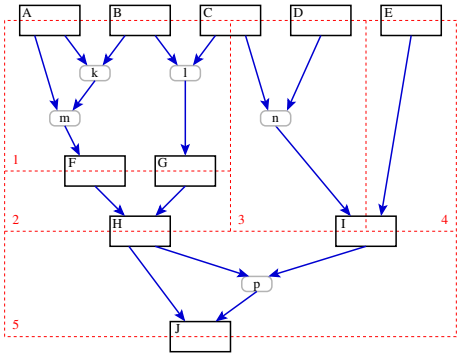
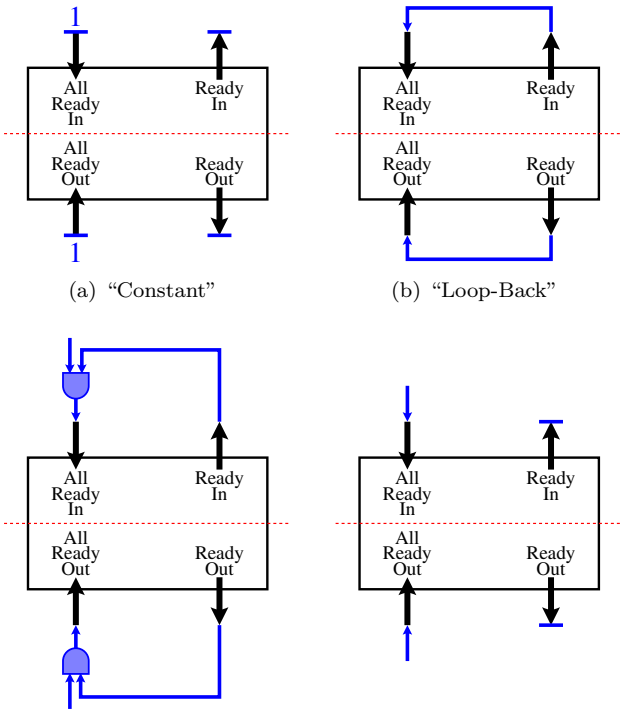


Figure 1: Some Example AHAHA Sections

0-7695-2940-2/07 $25.00 © 2007 IEEE
DOI 10.1109/FCCM.2007.10

3 Handshaking Overhead

The performance of graphs using handshaking is at least as high as statically timed models. Nodes fire as soon as they are able, and because the handshaking is purely combinational, no extra time is required. In cases with loop carried dependences which contain unbalanced conditionals, the handshaking model outperforms static timing solution. The amount of handshaking logic is proportional to the number of sections in the graph. We use a pipelined-mergesort algorithm to study handshaking overhead. A sorting network with n sorting processes sorts sequences of 2^n elements. Each merge-process in the network doubles the size of the resulting sorted sequences. This code has the highest handshaking overhead of all SA-C programs we tested. The AHAHA graph for the 16-process version of the code contains 2844 nodes, divided into 1271 sections. To quantify the amount of area used by the handshaking (as opposed to the computation itself), programs were generated without the handshaking logic included. There are several ways of removing handshaking logic, shown in Figure 2.



(a) "Constant" (b) "Loop-Back"

(c) "And" (d) "External"

Figure 2: AHAHA Handshaking removal methods

Handshaking involves creating the Ready In and Out signals, the AND gate that creates the AllReady signal, and logic to react to Ready and AllReady signals. Often the logic for the actual function of the node and handshaking is mixed. Constant supplies a `1` to the `AllReady` signals and sinks the Ready signals. It has the deficiency that actual node functionality will be dissolved. For example, a Buffer node will never have more than one element in it because it can always fire. Therefore, Constant is unacceptable, and only included for completeness. Loop-back connects `Ready` to `AllReady`. The And method causes each cluster to fire when it is ready, and an external signal indicates that it should fire, as when a finite state machine governs execution. The External method would be used in a fully static timing solution with no Ready feedback from the nodes.

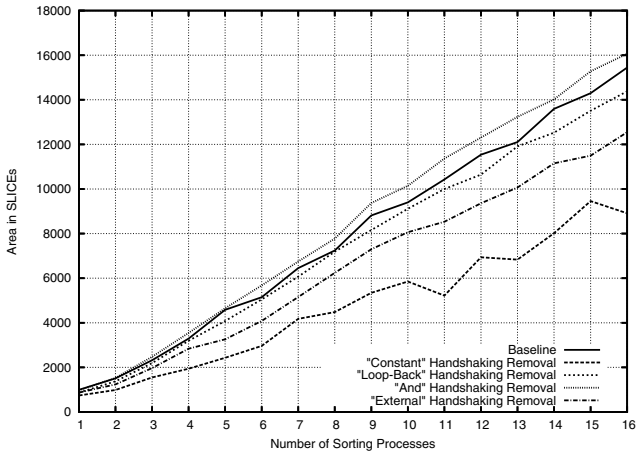


Figure 3: Comparison of handshaking removal methods

From the graph, it is clear that the dominating factor in every case is the size of the program being implemented. The type of handshaking removal has only a small effect on the slope of the line. The baseline lies between the External method and the And method, both of which would require additional logic to function properly. The Loop-back method does not assume the existence of extra logic and is the most accurate in ascertaining a less than 7% handshaking overhead in the worst case. The benefits in efficiency and expressiveness of handshaking outweigh this modest area requirement.

References

[1] B. Kienhuis, E. Rijpkema, and E. Deprettere, "Compaan: Deriving process networks from matlab for embedded signal processing architectures," in *8th International Workshop on Hardware/Software Codesign*, (San Diego, USA), pp. 13–17, 2000. `citeseer.ist.psu.edu/kienhuis00compaan.html`.

[2] "SRC." `www.srccomputers.com`.

[3] W. Böhm, J. Hammes, B. Draper, M. Chawathe, C. Ross, R. Rinker, and W. Najjar., "Mapping a single assignment programming language to reconfigurable systems," in *Supercomputing Journal 21*, pp. 117–130, 2002.

[4] M. Chawathe, *Hardware Compilation of Streams and Processes*. PhD thesis, Colorado State University, 2006.