Rapid Construction of Reconfigurable Computing Fabrics for Systems on a Programmable Chip

Christophe Wolinski
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.
IRISA, IFSIC France
wolinski@lanl.gov

Maya Gokhale
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.
maya@lanl.gov

Kevin McCabe
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.
kmccabe@lanl.gov

ABSTRACT

A reconfigurable computing fabric based on a System on a Programmable Chip (SoPC) is a parameterized cellular architecture in which an array of computing cells communicates with an embedded processor through a global memory [2]. This architecture is customizable to different classes of applications by functional unit, interconnect, and memory parameters. In previous work [1], we have demonstrated the advantage of reconfigurable fabrics for image and signal processing applications. In this paper, we describe Fabric Generator (FG), a Java-based toolset that greatly accelerates construction of the fabrics presented in [1]. A module-generation library is used to define, instantiate, and interconnect the cells' datapaths. Other tools generate customized sequencers for individual cells or collections of cells.

FG has been used to create fabrics for the Altera Excalibur ARM SoPC. We describe the FG tool set as well as a representative application generated using FG on the Altera. This application, a matched filter, achieved 4.5Gop/s (where a op is an 8/16-bit multiply-accumulate) on the Exalibur ARM, a 48X speedup over a 1.7GHz Pentium Xeon.

Keywords

Computer Aided Design, EDA, reconfigurable computing, FPGA, Configurable System on a Chip, Cellular Array, Computing Fabric

1. INTRODUCTION

Systems on a Programmable Chip (SoPC) are gaining popularity as a means of combining on a single chip the advantages of embedded processors for control and housekeeping tasks with the performance of application-specific reconfigurable logic. Several general purpose SoPC products have been offered recently by Altera[3] and Xilinx[4] in addition to application-specific SoPC being developed for telecommunications applications[5, 6].

In previous work ([7], [2], [1]), we have experimented with hardware/software co-processing on the Altera Excalibur ARM device. We have found that high performance can be obtained with a *fabric-based* approach, in which application-specific, interconnected compute cells are tiled across the reconfigurable logic, and the cells communicate with the embedded processor over a distributed dual-port memory. The memory is used to initialize each cell's data and program and to monitor the state of the fabric. We have manually developed Fabric-Based Systems that deliver up to 1.4GMACs (8-bit data, 16-bit arithmetic) at 33MHz on a 1M Gate Altera Excalibur ARM device.

Recently we have developed a Java-based toolset to automate fabric creation. Using these tools, it is possible to define low-level modules, populate a datapath with the modules, instantiate datapath cells in a fabric, and compile a microcoded program for datapath control into a sequencer.

In this paper we describe the fabric creation toolset and illustrate its operation on a Matched Filter application.

2. RELATED WORK

Fabric-based architectures have been popular since the invention of cellular automata. They are attractive because small, localized cells with small degree interconnect are efficiently implemented in VLSI or Programmable Logic Devices (PLDs), and for some application classes, exceed conventional processors' or DSPs' performance by orders of magnitude. Recent fabric-based architecture proposals include [8], [9], [10], [11], [12], and [13].

Our architecture design has many similarities to these proposals. The novel aspects of our approach are that in our parameterized cellular architecture, the cell, the interconnect, and the memory architecture all can be customized to the application. In addition, an embedded processor is an integral part of our model. The processor loads data and program, synchronizes the cells, and can inspect and retrieve data from the cells' memories.

The Fabric Generator (FG) is similar to JHDL[14], LDG[15], PamDC[16], and other CAD tools embedded in high level programming languages. With these tools, it is possible to write a high level language program to describe, instantiate, and interconnect hardware modules. JHDL and PamDC, by using the overloading features of Java and C++, also provide simulation capability, which our system does not yet include.

HPCA-9/SSRS Anaheim, California, February 8-12, 2003

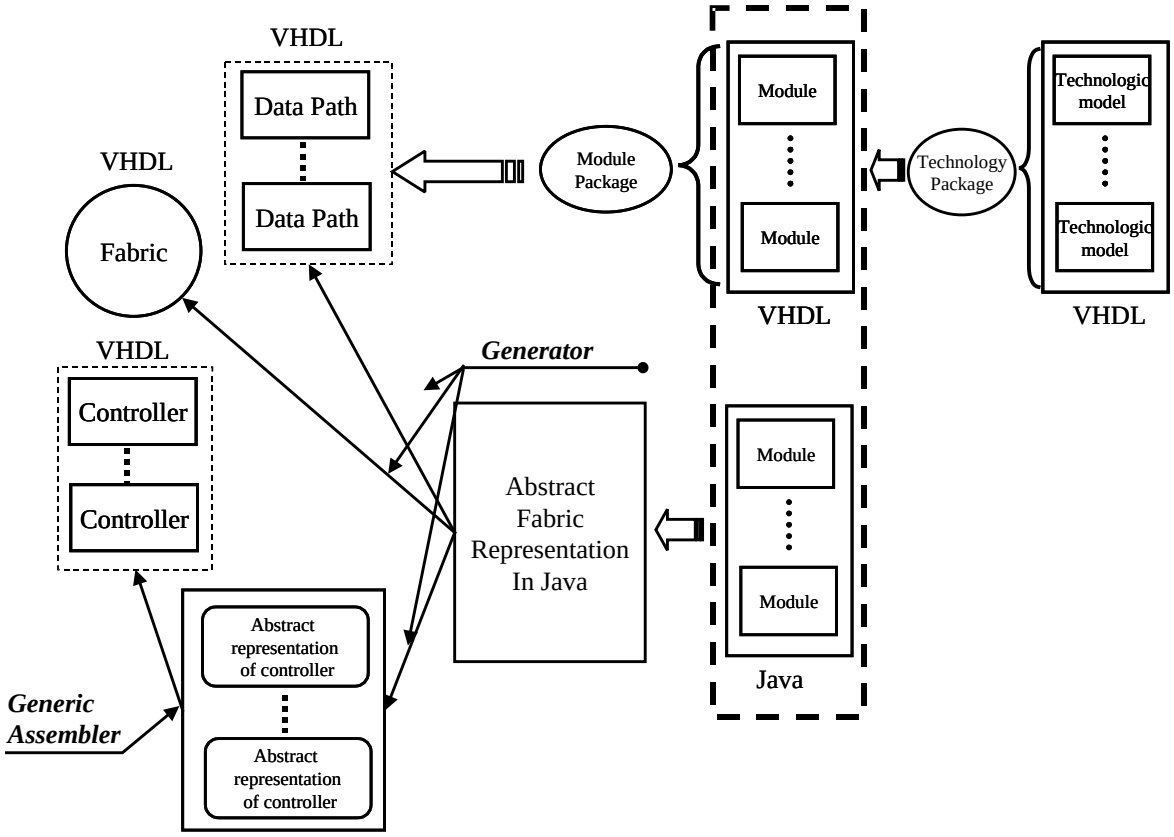

Figure 1: Fabric Generator Organization

In contrast to these tools, FG uses a built-in programming model of the Fabric-Based System, and automatically generates control signals associated with a cell's datapath. When given a microcode program for a specific datapath, FG generates a cell sequencer to control one or more cells. It also generates a complete fabric with interconnected cells and associated controllers. Unlike JHDL, FG generates Register-Transfer-Level VHDL. The present implementation is targeted to the Altera Excalibur part with Apex PLD.

3. POLYMORPHOUS COMPUTING FABRIC

At a high level, our Polymorphous Fabric is simply a fabric of simple, inter-connected computational datapath cells, each with an optional local memory. The collection of local memories forms a (dual-ported) global memory that can be loaded and examined from the attached processor. The cells composing the fabric need not all be the same – a fabric may contain groups of homogeneous cells, as illustrated in Figure 2, in which different cell types are distinguished by different names ("Send," "Rec," "P,", and "Ele"). Each datapath cell may have its own controller, or alternatively, a group of identical cells may share a controller. Many different sorts of communications patterns may be realized within a single fabric.

The Fabric can be considered as a processing memory and presents a standard memory to the embedded processor.

4. FABRIC GENERATOR

The FG library contains classes to

- define a module
- create a datapath of interconnected modules
- instantiate cells consisting of datapaths with associated sequencers
- create a fabric of interconnected cells

As cells are instantiated, information is automatically collected about the control signals associated with each datapath, and a file listing each signal is generated as exemplified in Figure 5, with empty program space. The designer enters into this file the microcode sequence for a datapath and then runs an assembler, which generates the state machine to control the datapath. The combination of datapath cells and sequencers is then synthesized by standard logic synthesis tools and FPGA-specific Place and Route tools to obtain the fabric's bit stream. The FG system is shown in Figure 1.

The "Abstract Fabric Representation in Java" is a Java program written by the fabric designer. The fabric program can call methods provided by the library to define and instantiate modules (see Java modules in the dotted box). In addition, modules may be defined in VHDL using technology-specific components, if desired, and instantiated

in the fabric program. Modules can be instantiated to build a datapath. Next, cells may be defined. A cell contains a datapath and specifies a sequencer (controller). Cells may share a sequencer (SIMD mode) or each cell may have a unique sequencer (MIMD mode). A fabric contains a collection of cells and sequencers. As shown in the Figure, the fabric program calls a method to generate a sequencer description file (labelled "Abstract representation of controller"). After the fabric designer has microcoded a low level control sequence for a datapath, an assembler generates a state machine ("Controller" in the Figure). The Controllers, Fabric, and Datapaths in the Figure are then synthesized through the standard CAD tool chain. The result is a component with standard memory interface which can be connected to any processor. In our case the Fabric was connected to ARM processor inside of Altera Excalibur ARM SoPC.

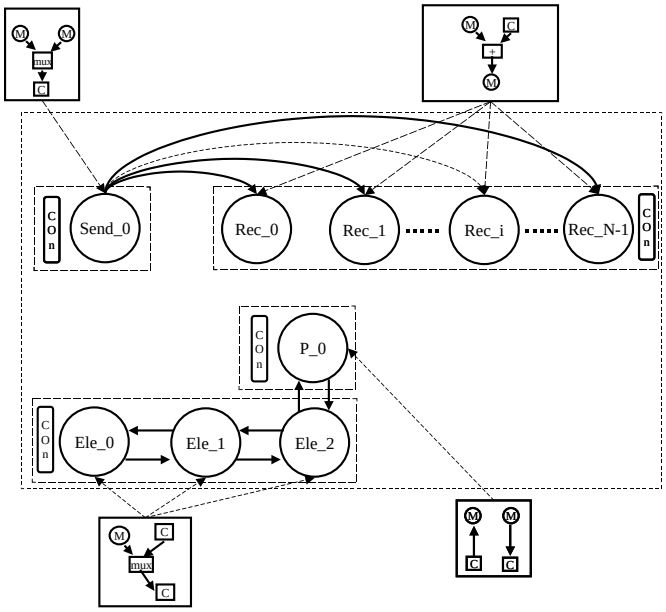

Figure 2: Example Fabrics

Example fabrics are shown in Figure 2. The upper fabric consists of a Send cell and multiple Receive cells. The Send cell broadcasts data to all the Receive cells. The datapath for the send cell contains two memories, a multiplexer and a communication channel. The Send cell has its unique controller. The Receive cells add the channel input with input from memory, storing the result in memory. All the Receive cells share a common controller.

The lower fabric is a linear bi-directional array. P_0 reads data from a memory and sends it on the output communication channel. It receives data on the input channel and stores to memory. The three Ele cells each select from either memory or the input communication channel and forward on the output channel.

The tools developed to help build such fabrics are described in the following sections.

4.1 Defining a Module

The module is the lowest level construct in the design hierarchy. The library presently contains about twenty parameterized modules. Most modules are parameterized by data width, and may have additional parameters. Arithmetic fixed point modules may be signed or unsigned. Existing modules include

```
Cell c=new Cell("cell_0",Technologie,"SIGNED");

c.addModule(new Channel("ch",8,1));
c.addModule(new Memory("m0",256,8,0,0,1));
c.addModule(new Operation("op0","+"));
c.addModule(new Memory("m1",256,8,0,0,1));
c.makeConnectionFromModule1ToModule2("ch","op0");
c.makeConnectionFromModule1ToModule2("m0","op0");
c.makeConnectionFromModule1ToModule2("op0","m1");
```

Figure 3: Creating a Datapath

- communications channel, may be registered
- multiplexers, bus selectors, bus expanders, bus merge modules
- memory modules, including dual port memory with a variety of access options such as random, sequential, circular and indexed,
- several varieties of registers, including registers that are visible from the embedded processor
- fixed and floating point adders and multipliers, parameterized by mantissa and exponent size
- fixed point multiply-accumulate
- condition codes and registers to send state information from datapath to controller

New modules can be written in Java, building on a ModuleBase class, or may be created in VHDL and then instantiated through the ModuleBase methods.

Each module has a set of data busses and a set of *hidden* signals. The data busses are determined by the nature of the module. For example an adder has two input busses and an output bus, while the multiplexer module has an unlimited number of input busses and one output bus.

The hidden signals include the clock, reset, and enable signals as well as condition, and interface signals. The condition signals are automatically connected to the sequencer, so that computation within a datapath can cause state changes. The interface signals appear at the high level of the fabric and thus can be seen from the processor or other external device.

Moreover the hidden signals provide the interface to high level synthesis tools. The information on the module's enable and reset signals is used during the automatic controller synthesis process.

4.2 The Datapath

The "cell" class is used to create datapaths by selecting modules for inclusion in the datapath and interconnecting the datapath modules. Figure 3 shows how to create a datapath for the Receive cell of Figure 2. The cell instantiates Channel, Memory, and Operation modules, and then connects the modules.

4.3 Fabric Specification

A fabric is specified by instantiating cells. Figure 4 shows the library calls required to instantiate the two types of cells from Figure 2.

```
FabricMachine f = new FabricMachine("EF",0,2);

fabric.addCell("Send",0,cell1,0); int i;
for (i=0;i<140;i++) {
   fabric.addCell("Rec",i,cell,0);}
for (i=0;i<140;i++) {
   fabric.ConnectCell1ToCell2("DataChannel",
                              "Send",0,"ch",
                              "Rec",i,"ch");}
```

Figure 4: Creating a Fabric

This fabric contains one Send cell and 140 Receive cells. The Send output channel is connected to the input channel of each Receive cell.

Of note is the final parameter to the addCell method. This parameter selects a controller for the datapath. All like cells with the same parameter value are assigned a common controller. In this example, the Send cell has its own controller 0 and all the receive cells share a common controller 0. If the parameter were changed to "i", the loop variable, each Receive cell would have its own controller.

4.4 Sequencer Generation

In addition to generating the VHDL associated with interconnected cells, FG generates a state machine to sequence and control the cells. The state machine is built from a microcode program written by the fabric designer. A template listing the datapath's control signals is generated when a fabric is instantiated, as shown in Figure 5. This Figure shows the Receive program. The hidden control signals associated with modules in the datapath are listed in the generated template, and the fabric designer appends the actual program that references these signals. In this example, there are ten signals that may be set by the sequencer. Two are 8-bit buses, m0_Operand and m1_Operand and the rest are one-bit control signals. The Condition section is empty, as a condition module was not used in a datapath. Each "Instr" may set any of the signals. If a signal is referenced, it is set to one (or a specific value if it is a bus); otherwise the signal defaults to zero.

The Receive program has two sections, labelled "process" and "start." When the processor issues a Reset signal, control is transferred to the "start" label, where the StartProgram token appears. The "wait_start" directive means to wait for the start signal from the processor, and when received, branch to "process." At the same time, the m0 and m1 ResetCounter signals are asserted. Once a start signal is sent from the host, the instruction sequence at "process" is executed. The first instruction is a single cycle noop. The next instruction reads from the DataChannel and concurrently accesses memory m0 in read mode by asserting the m0_MEnableAcces signal. It also accesses memory m1 in write mode by asserting m1_MReadWrite and m1_MEnableAcces. Consecutive addresses in the memories are accessed. This instruction executes for 256 cycles. Then the datapath must wait for another start signal from the processor to repeat the cycle.

5. EXAMPLE: MATCHED FILTER

The matched filter is a well-known technique for detecting a signal in an image in the presence of known forms of "clutter." By filtering with respect to pre-determined background signatures, weak signals may be recovered that might otherwise have been lost in the background. For this application, we assume that matched filter coefficients have been created, and present a fabric design for a bank of matched filters. This design was created with the Fabric Generator tool set. The matched filter fabric uses the communication pattern introduced above in the upper example of Figure 2 in which a Send cell broadcasts a data stream to multiple Receive cells. A double buffering technique is used by the Send cell. The host writes data to one memory while the Send cell broadcasts from the other memory. Thus the communication time is overlapped with computation. Each Receive cell computes an inner product, using coefficients stored in one memory and data from the input channel. The result is stored in the other memory. The processor initializes the coefficient memories of the Receive cells and reads back the result memories after the data has been processed. The matched filter fabric is shown graphically in Figure 6. The Send and Receive microcode programs are shown in Figure 7.

The Send program has two sections, labelled mem0 and mem1. When the processor issues a Reset signal, control is transferred to the second instruction of mem1, where the StartProgram token appears. While waiting for the start signal, the m0 and m1 ResetCounter signals are asserted. Once a start signal is sent from the host, the instruction at mem0 is executed. This instruction accesses memory m0 asserting the m0_MEnableAcces signal and puts a result onto DataChannel. This instruction executes for 256 cycles. Then the datapath must wait for another start signal from the processor, and when it is received, branches to mem1. When start is received, m1 is read, the multiplexer setting is switched so that mem1 data is output to the channel. The Receive program is a bit more complicated due to a 16-bit accumulation stored into an 8-bit memory. After the initial StartProgram sequence, the accumulator is initialized. Then the DataChannel and memory 0 are read. In the datapath, they are the input operands to the multiply. Concurrently the output of the multiply is loaded into a 16-bit register. The next instruction continues the same access pattern, and in addition enables the accumulator, which stores the result of the add into the accumulator (refer to the acc unit in Figure 6). This instruction is repeated for 255 cycles. After a one cycle delay, m0 is written with the lower half of the adder output. In the next cycle, m0 is written with the upper half of the adder output, and control returns to the start label to wait for a start signal from the processor. It is possible to fit 140 Receive units onto the APEX 20KE PLD portion of the Excalibur ARM device. With a clock frequency of 33MHz, the design achieves 4.5GOps/s where each Op is a multiply accumulate with 8-bit operands and 16-bit multiplier and adder. The speedup compared to ARM running at 200 MHz, is 994X. Compared to a TMS320c6201 running at 200MHz, the speedup is 48, and compared to a Pentium Xeon running at 1.7GHz it is 48.

6. CONCLUSIONS

We have described the Fabric Generator, a tool set to aid

```
#Inserted by Fabric Generator

#  StartProgram                  -- an entry point into program
#  EndLoop      label  iteration -- String   int,  if iteration=0 permanent Loop
#  wait_start   label            -- String   Waiting for Start Signal if start=1 jmp to label
#  wait_cycles  number_cycles    -- int
#  jmp          label            -- String
#  putChannel   name   Number    -- String   int
#  getChannel   name   Number    -- String   int

Channels
   DataChannel IN;
END Signals
   m0_Operand        8
   m0_MEnableAcces   1
   m0_MReadWrite     1
   m0_MResetCounter  1
   m0_MLoadCounter   1
   m1_Operand        8
   m1_MEnableAcces   1
   m1_MReadWrite     1
   m1_MResetCounter  1
   m1_MLoadCounter   1
END Conditions
END
#  End Inserted by Fabric Generator
Program
process : Instr ;
          Instr getChannel DataChannel, m0_MEnableAcces, m1_MReadWrite, m1_MEnableAcces,wait_cycles  256; # N time
start   : Instr StartProgram, wait_start process, m0_MResetCounter, m1_MResetCounter;
END
```

Figure 5: The Receive Cell Program

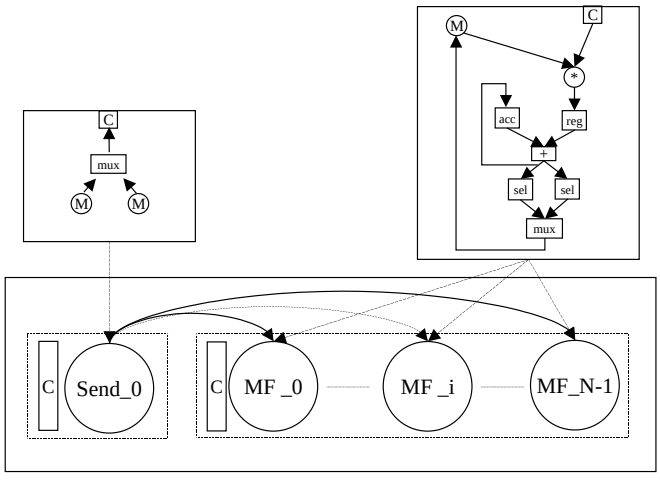

Figure 6: Matched Filter Fabric

construction of fabrics for Systems on a Programmable Chip. FG supports a programming model of an array of cells with memory-based communication with an embedded processor. FG consists of a Java library through which modules, datapaths, and fabrics may be constructed, and an assembler that generates a state machine sequencer from a sequence of datapath-specific microinstructions. We have used FG to build applications for the Altera Excalibur ARM and have achieved 4.5GOps/sec on a Matched Filter.

7. REFERENCES

[1] C. Wolinski, M. Gokhale, and K. McCabe, "A new polymorphous computing fabric," *IEEE Micro*, Sept. 2002.

[2] ——, "A reconfigurable computing fabric," *ERSA 2002*, June 2002.

[3] Altera, "Altera Corporation," *www.altera.com*, 2002.

[4] Xilinx, "Xilinx, Inc." *www.xilinx.com*, 2002.

[5] Quicksilver, "Quicksilver Technology," *www.qstech.com*, 2002.

[6] Chameleon, "Chameleon Systems," *www.chameleonsystems.com*, 2002.

[7] M. Gokhale, J. Frigo, K. Mccabe, D. Lavenier, and J. Theiler, "Early experience with a hybrid processor: K-means clustering," *ERSA 2001*, June 2001.

[8] J. Vuillemin, P. Bertin, *et al.*, "Programmable active memories: Reconfigurable systems come of age," *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 56–69, Mar. 1996.

[9] T. Miyamori, "A quantitative analysis of reconfigurable coprocessors for multimedia

```
#Inserted by Fabric Generator

#Send Program
Channels
   DataChannel OUT ch_EnableRegister;
END Signals
   m0_Operand        8
   m0_MEnableAcces   1
   m0_MReadWrite     1
   m0_MResetCounter  1
   m0_MLoadCounter   1
   m1_Operand        8
   m1_MEnableAcces   1
   m1_MReadWrite     1
   m1_MResetCounter  1
   m1_MLoadCounter   1
   mx0_0             1
END Conditions
END
Program
mem0  : Instr m0_MEnableAcces, putChannel   DataChannel   0, wait_cycles 256;      # N time
        Instr wait_start mem1, m0_MResetCounter , m1_MResetCounter;
mem1  : Instr m1_MEnableAcces, mx0_0, putChannel   DataChannel 1, wait_cycles 256; # N time
        Instr StartProgram, wait_start mem0, m0_MResetCounter , m1_MResetCounter;
END

#Receive Program Channels
   DataChannel IN;
END Signals
   m0_Operand         8
   m0_MEnableAcces    1
   m0_MReadWrite      1
   m0_MResetCounter   1
   m0_MLoadCounter    1
   reg0_RegisterLoad  1
   acc0_RegisterLoad  1
   acc0_RegisterInit  1
   mx0_0              1
END Conditions
END
Program
start   : Instr StartProgram, wait_start process, m0_MResetCounter; process : Instr acc0_RegisterInit;
          Instr getChannel DataChannel 0, m0_MEnableAcces, reg0_RegisterLoad;
          Instr getChannel DataChannel 2, m0_MEnableAcces, reg0_RegisterLoad,
                acc0_RegisterLoad, wait_cycles 255; # N-1
          Instr ;
          Instr m0_MReadWrite, m0_MEnableAcces;
          Instr mx0_0, m0_MReadWrite, m0_MEnableAcces, jmp start;
END
```

Figure 7: The Matched Filter Programs

applications," *IEEE Symposium on FPGAs for Custom Computing Machines*, Apr. 1998.

[10] C. E. D. C. Green and P. Franklin, "RaPiD – reconfigurable pipelined datapath," in *Field-Programmable Logic: Smart Applications, New Paradigms, and Compilers. 6th International Workshop on Field-Programmable Logic and Applications*, R. W. Hartenstein and M. Glesner, Eds. Darmstadt, Germany: Springer-Verlag, Sept. 1996, pp. 126–135.

[11] J. R. Hauser and J. Wawrzynek, "GARP: A MIPS processor with a reconfigurable coprocessor," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, J. Arnold and K. L. Pocek, Eds., Napa, CA, Apr. 1997.

[12] E. Waingold, M. Taylor, *et al.*, "Baring it all to software:raw machines," *IEEE Computer*, pp. 86–93, sep 1997.

[13] V. Baumgarte, F. May, *et al.*, "Pact xpp - a self-reconfigurable data processing architecture," *International Conference on Engineering of Reconfigurable Systems and Algorithms*, June 2001.

[14] B. Hutchins *et al.*, "Jhdl-an hdl for reconfigurable systems," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 1998.

[15] M. Gokhale, A. Kopser, S. Lucas, and R. Minnich, "The logic description generator," *InternationalConference on Application Specific Array Processors*, Sept. 1990.

[16] Compaq, "Pamdc," *research.compaq.com/SRC/pamette/PamDC.pdf*, 1999.