LA-UR- 03 -0352

Approved for public release; distribution is unlimited.

Title: FABRIC-BASED SYSTEMS: MODEL, TOOLS, APPLICATIONS

Author(s): CHRISTOPHE C. WOLINSKI
MAYA B. GOKHALE
KEVIN P. MCCABE

Submitted to: CONFERENCE FCCM 2003
IEEE SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES

Fabric-Based Systems: Model, Tools, Applications

Christophe Wolinski
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.
IRISA, IFSIC France

Maya Gokhale
Kevin McCabe
Los Alamos National Laboratory
Los Alamos, NM, U.S.A.

Abstract

A Fabric Based System is a parameterized cellular architecture in which an array of computing cells communicates with an embedded processor through a global memory. This architecture is customizable to different classes of applications by funtional unit, interconnect, and memory parameters, and can be instantiated efficiently on platform FPGAs. In previous work [13], we have demonstrated the advantage of reconfigurable fabrics for image and signal processing applications. Recently, we have build a Fabric Generator, a Java-based toolset that greatly accelerates construction of the fabrics presented in [13]. A module-generation library is used to define, instantiate, and interconnect cells' datapaths. FG generates customized sequencers for individual cells or collections of cells. We describe the Fabric-Based System model, the FG toolset, and concrete realizations of fabric architectures generated by FG on the Altera Excalibur ARM that can deliver 4.5 GigaMACs/s (8/16 bit data, Multiply-Accumulate).

1 Introduction

A computational fabric is a form of cellular array composed of application-specific, interconnected compute cells. A Fabric-Based System (FBS) combines a computational fabric with a conventional control processor such that the processor views the fabric as a large intelligent memory [14, 13]. Fabric-Based Systems combine the compute power of a fabric with the flexibility, visibility, and control of traditional processing.

While computational fabrics have been proposed and implemented in the past, the notion of a Fabric-Based System introduces two key new concepts. First, the model incorporates a standard processor that views the fabric as a memory. Each cell in the fabric has a local data memory. The collection of local memories of all the cells form a global memory. By making this memory dual-ported, it can be accessed concurrently by a cell as well as the processor. Second, the computational fabric is highly parameterized, permitting application-specific customization for efficient mapping to reconfigurable logic.

We present implementations of FBS on a platform FPGA, the Altera Excalibur ARM. Since the computational fabric is mapped to reconfigurable logic, application-specific fabrics can be easily generated. Parameters to fabric generation include the number of nodes; the amount of local memory per node; the amount of global memory shared between microprocessor and fabric; the arithmetic unit functionality; dimensionality and form of the interconnect; and width of the interconnect.

In this paper, we describe the Fabric-Based System model; a toolset, the Fabric Generator, to create fabrics; and applications that demonstrate the performance of FBS.

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 [11], [10], [7], [8], [12], and [1].

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

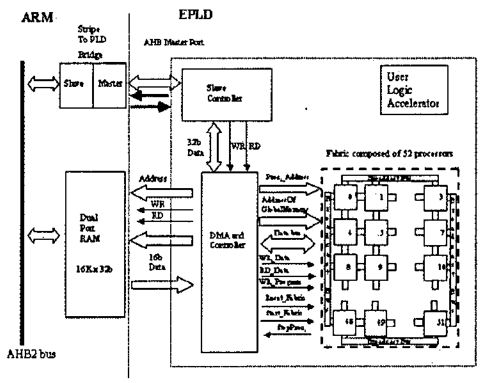

Figure 1. A Fabric-Based System

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[9], LDG[6], PamDC[3], 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.

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 Fabric-Based System

A Fabric-Based System consists of a standard processor closey coupled to a computational fabric through a memory interface. A reference implementation of an FBS on the Excalibur ARM is shown in Figure 1 in which a 52-cell fabric communicates with the ARM processor via a DMA controller.

The computational fabric contains simple, inter-connected 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 have the same datapath – a fabric may contain groups of homogeneous cells, as illustrated in Figure 3, 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. A controller can run many different programs. 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 standard to the processor. The processor loads the fabric with program and data, synchronizes top-level control flow and phase-ordering, and reads back results.

4 Fabric Generator

In order to facilitate the generation of fabrics in a FBS, we have created a Java-based fabric generator library.

The FG library contains classes to

- define a module

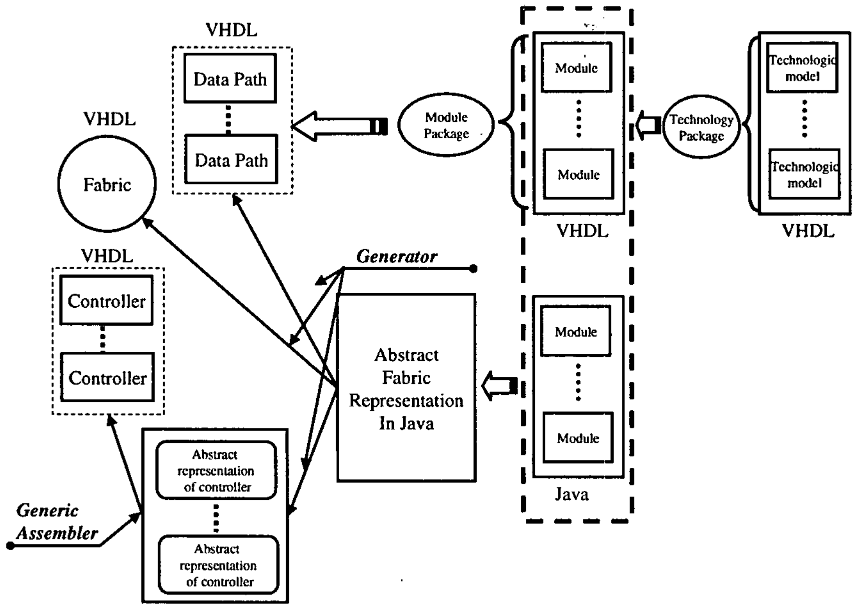

Figure 2. Fabric Generator Organization

- 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 (see Figure 6, the section marked "Generated by Fabric Generator").

The designer fills 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 to obtain the fabric's bit stream. The FG system is shown in Figure 2.

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. A collection of instantiated modules defines a datapath. The next level of hierarchy is the cell. 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). At the next level of hierarch, a fabric is simply a collection of interconnected 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 written an assembly language program to control the 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.

Example fabrics are shown in Figure 3. 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 FG toolset developed to help build such fabrics is described in more detail in the following sections.

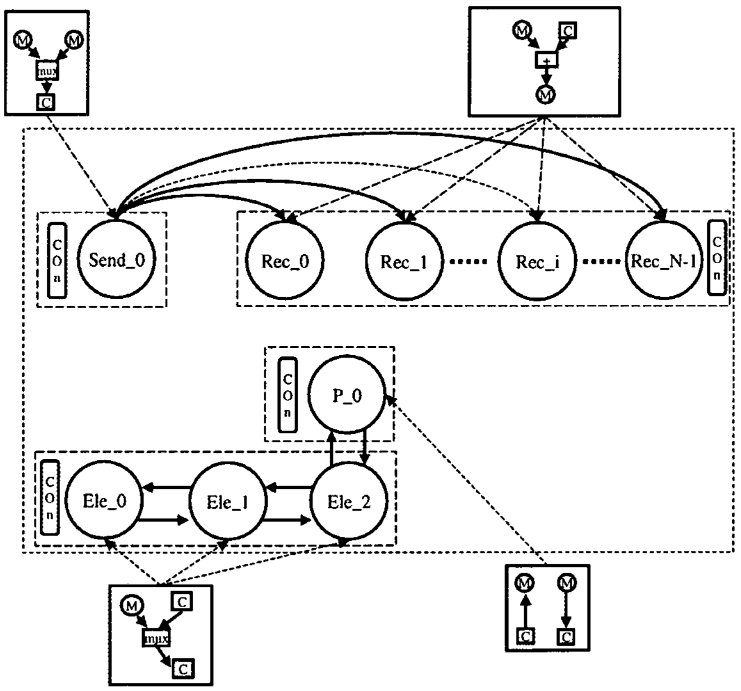

Figure 3. Example Fabrics

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

- 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. These basic control signals offer a standard method to control the datapath, and are particularly attractive for automatic generation of fabrics from high level synthesis. 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.

```
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 4. Creating a Datapath

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 4 shows how to create a datapath for the Receive cell of Figure 3. 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 5 shows the library calls required to instantiate the two types of cells from Figure 3.

```
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 5. 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 inter-connected cells, FG generates a state machine to sequence and control the cells. There are many different options in sequencer generation. A single sequencer can control an array of cells. A single sequencer can control an individual cell.

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 6. 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 assembly language 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

```
#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 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 6. The Receive Cell Program

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 addressed in the meories 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 Applications

In this section we describe two applications mapped to computing fabrics in an FBS. The applications, generated using the FG toolset, are characteristic of data intensive processing of multi- and hyperspectral remote sensing imagery.

5.1 Unsupervised Clustering

The unsupervised clustering algorithm is a popular data mining technique, also called k-means clustering. The purpose of the algorithm is to classify a data set into a number of classes. Each element of the data set is a vector, such as the red, green,

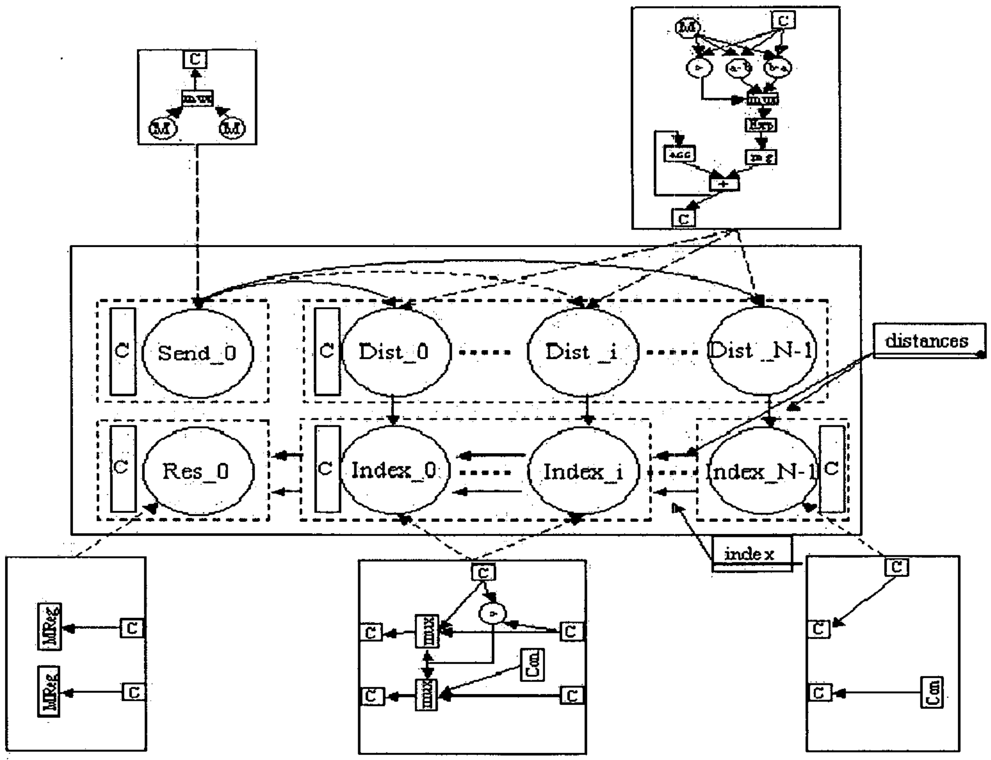

Figure 7. KMeans Fabric

and blue components of a pixel. This iterative algorithm has the following steps:

1. Randomly assign each data element A[i] to one of k classes
2. Compute the centers of the classes
3. Loop over the data set A
   (a) Let C = class of A[i]
   (b) Determine the class number K which has the minimum distance to C
   (c) if C is not equal to K, move pixel C to Class K
4. Recompute the centers of the classes K and C

A reconfigurable hardware implementation of this algorithm has been reported in [4] and direct mapping to the Altera Excalibur NIOS in [5]. The key compute intensive kernel of this algorithm is distance calculation between a class center and a data element. We use the distance metric suggested by [4], $|A[i] - C[i]|$ to approximate the more traditional Euclidean distance metric.

This design has four cell types, Send, Distance Calculation, Index Calculation, and Receive. There are multiple distance calculation cells (Dist_1 ... Dist_N-1), and index calculation cells (Index_0 ... Index_N-1), where N is the number of classes. We parallelize the computation across classes, with one distance/index pair per class. The distance calculation cells compute the distance between the pixel and the class centers. The index calculation cells calculate the index of the class having the minimum distance to the pixel. Cell Res_0 uses two registers to store the minimum distance and associated class index.

Cell Send_0 sends the pixels from one of two memories. Since the entire image does not fit into the small on-chip memories, we divide an image plane smaller blocks. The ARM processor transmits each block to the Fabric while the Fabric computes with the previously transmitted block. This concurrent computation and communication means that there is no

Clock pins	1
Fast i/o pins	0
Global signals	3
I/O pins	72
Logic elements	23289
ESBs	152 / 160 (95 %)
Macrocells	0
Flipflops	12181
FastRow interconnects	0 / 120 (0 %)
Maximum fan-out	3578
Total fan-out	99790
Average fan-out	4.1
ESB pterm bits used	0 / 327680 (0 %)
ESB non-CAM memory bits used	311296 / 327680 (65 %)
ESB CAM bits used	0 / 327680 (0 %)
Total ESB bits used	311296 / 327680 (95 %)
Total RAM block bits	311296 / 327680 (95 %)

Figure 8. Physical APEX resources used by KMeans Fabric

added cost for working with a segmented image beyond the time to transfer the first block. The double buffering hides the communication latency to acquire the pixels.

The Dist_i cells receive the pixel and compute its distance to their stored class centers, forwarding the distance to the associated Index_i cells. As in the homogeneous fabric implementation, the index calculation cells compare the distance to the minimum distance and sends the minimum distance along with the index associated with the minimum distance to their neighbor. However, in this fabric, dynamic reconfiguration of the communication pattern through an instruction is not required. Instead, all possible paths have been encoded into the cell, and the output is selected and forwarded by the datapath rather than through a controller instruction. The index calculation takes N cycles. The cell Res_0 receives the distance and index.

In this implementation, there are five unique sequencers. The cells Send_0, Res_0, and Index_N-1 each have a dedicated sequencer. All the Dist cells share one sequencer, as do the Index 0 – N-2 cells. Resource utilization on the Excalibur ARM is shown in Figure 7, with a clock rate of 33 MHz. This fabric can compute 150 classes, and operates at 4.5Gops/s, where the operation is defined as before (8-bit abs/accumulate).

5.2 Spectral Matched Filter

The matched filter is well-suited to automatic detection of signals within multi- or hyper-spectral data image cubes containing tens to hundreds of spectral channels. The purpose of a matched filter is to match an image pixel's spectral signature against a pre-determined "background" signature. When analyzing multi- or hyper-spectral imagery with complex background clutter for small/weak targets, filtering a target image through a matched filter suppresses background spectra and thus increases response to non-background features.

Let R represent an image cube, b = the spectral signature of an expected target, and $K = RR^T$ be the covariance matrix of R, which describes the statistics of the background clutter. Then $q = b$ is the matched filter[1], and $q^T R$ is map of the desired target within the image (see Figure 9[2]). A bank of matched filters may be created to filter out a variety of "clutter" effects.

For our purposes, we assume that the bank of (non-adaptive) matched filters exists, and present algorithms to filter an image cube relative to the matched filter bank.

The matched filter fabric uses two types of cells, a Send cell and a Match cell (Figure 10). The Send cell broadcasts pixels to all the Match cells. Each Match cell computes the inner product and stores results to memory. Resource utilization on the Excalibur ARM is shown in Figure 11 for 140 Matched Filters, with a rate of 4.5GMACs/s. The Excalibur runs at 33 MHz.

[1]alternatively $q = K^{-1}b$ is the adaptive matched filter.

Spectral matched filter

- R = image cube
- b = spectral signature of plume
- $K = RR^T$ = covariance matrix; describes statistics of background clutter

- $q = b$ = ordinary matched filter
- $q = K^{-1}b$ = adaptive matched filter

- q^TR = plume map

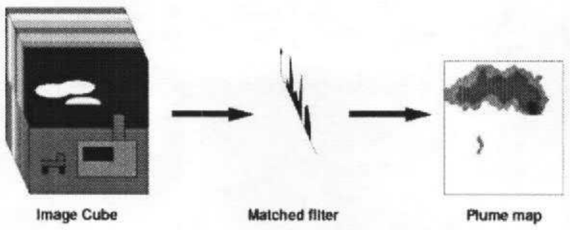

Figure 9. Matched Filter Concept

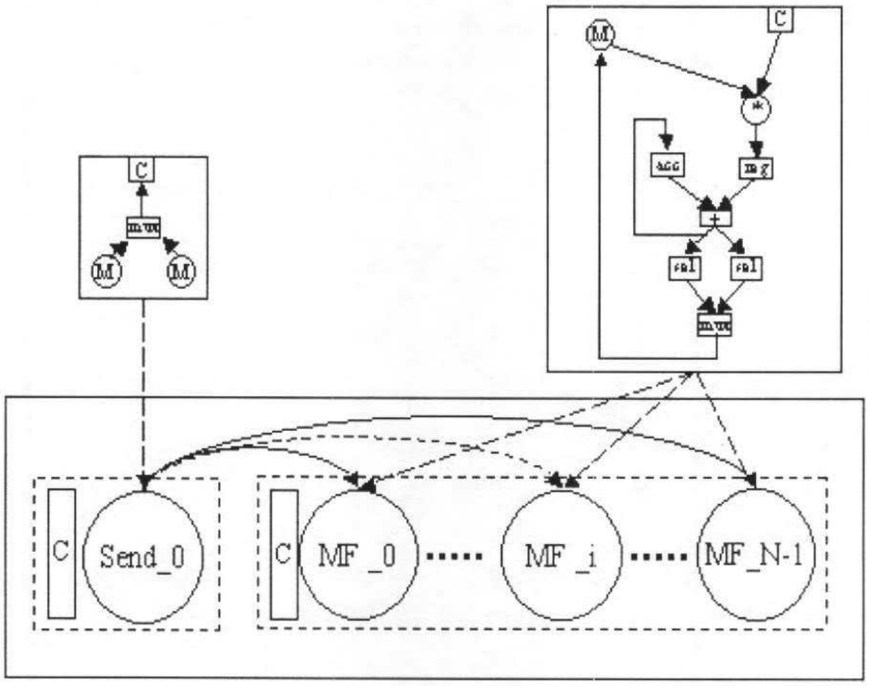

Figure 10. Matched Filter Fabric

Logic elements	28178
ESBs	142 / 160 (88 %)
Macrocells	0
Flipflops	5753
FastRow interconnects	0 / 120 (0 %)
Maximum fan-out	2240
Total fan-out	119829
Average fan-out	4.1
ESB pterm bits used	0 / 327680 (0 %)
ESB non-CAM memory bits used	290816 / 327680 (88 %)
ESB CAM bits used	0 / 327680 (0 %)
Total ESB bits used	290816 / 327680 (88 %)
Total RAM block bits	290816 / 327680 (88 %)

Figure 11. Physical APEX resources used by Matched Filter Fabric

6 Conclusions

We have described a computing fabric consisting of a parameterized cellular array with a memory interface to a conventional processor. The parameterized nature of the architecture permits generation of the fabric for specific classes of applications. A novel aspect of the FBS is the use of global memory. This memory gives the host procesor random access to all variables and instructions on the fabric's cells. The memory can be initialized in the same way as a standard memory in a computer. Programs and data can be dynamically loaded during processing on the fabric because the global memory is dual ported, reducing overhead for preparing the data and programs. We have described the Fabric Generator, a tool set to aid construction of fabrics for platform FPGAs. 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 two data and compute intensive applications.

References

[1] 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.

[2] J. Bloch. rcc.lanl.gov/imaging/index.php. 1999.

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

[4] M. Estlick, M. Leeser, J. Szymanski, and J. Theiler. Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware. *ACM FPGA 2001*, 2001.

[5] M. Gokhale, J. Frigo, K. McCabe, J. Theiler, and C. Wolinski. Experience with a hybrid processor: K-means clustering. *Journal of Supercomputing*, 2002.

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

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

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

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

[10] T. Miyamori. A quantitative analysis of reconfigurable coprocessors for multimedia applications. *IEEE Symposium on FPGAs for Custom Computing Machines*, Apr. 1998.

[11] J. Vuillemin, P. Bertin, et al. Programmable active memories: Reconfigurable systems come of age. *IEEE Transactions on VLSI Systems*, 4(1):56–69, Mar. 1996.

[12] E. Waingold, M. Taylor, et al. Baring it all to software:raw machines. *IEEE Computer*, pages 86–93, sep 1997.
[13] C. Wolinski, M. Gokhale, and K. McCabe. A new polymorphous computing fabric. *IEEE Micro*, Sept. 2002.
[14] C. Wolinski, M. Gokhale, and K. McCabe. A reconfigurable computing fabric. *ERSA 2002*, June 2002.