CPRE/COMS 583x Adaptive Computing Systems	Fall Semester 1998
Lecture 25: November 17, 1998	
Instructor: Arun K. Somani & Akhilesh Tyagi	Scribe: Wei Chen

1 Introduction

In this section we studied three different architectures for reconfigurable computing: the Garp project at UC Berkeley, PipeRench architecture at CMU and Colt project from Virginia Tech.

2 Garp Architecture

2.1 Introduction

Garp architecture is targeting to build a machine that tightly couples reconfigurable hardware with a convential microprocessor. Garp's reconfigurable array is composed of entities called blocks. One block on each row is knowned as a control block. The rest of the blocks in the array are logic blocks. The basic "quantum" of data within the array is 2 bits. Logic blocks operate on values as 2-bit units, and all wires are arranged in pairs to transmit 2-bit quantities. Memory buses are used for moving information into and out of the array. The loading and excution of configuration is under the control of the main processor. To do that, several instructions have to be added into the MIPS-II instruction set.

The control blocks at the left end of each row serve as liasion between the array and the outside world. Control blocks can interrupt the main processor and can initiate data memory accesses to and from the array.

2.2 Array logic blocks

Each logic block in the array can implement a function of up to four 2-bit inputs. Operations on data wider than 2 bits can be formed by adjoining blocks as a row. The principal logic block functions are illustrated in [1] Figure 5. The four modes in that figure implement:

- an arbitrary 4-input bitwise logical function;
- a variable shift of up to 15 bits;
- a 4-way select(multiplexor) function; and
- a 3-input add/subtract/comparison operation.

The decision to make everything 2 bits wide is to reduce the time required to load configuration and the space taken up on the die to store them. Compared to typical FPGAs, Garp expends more hardware on accerlerating operations like adds and varaiable shifts.

2.3 Array wires

Vertical and horizontal wires exist within the array for moving data between logic blocks. All array wires are grouped into pairs to carry 2-bit quantites. Each pair of wires can be driven by only a single logic block but can be read simultaneously by all the logic blocks spanned by the wires. The driver of every wire is fixed by a configuration and cannot be changed without loading a new configuration.

2.4 Array timing

Delays in Garp are defined in terms of the sequences that can be fit within each array clock cycle. Only three sequences are permitted.

- short wire, simple function, short wire, simple function;
- long wire, any function not using the carry chain;
- short wire, any function.

Any other sequence must be assumed to require multiple clock cycles. The short wires includes all the shorter horizontal wires, plus vertical wires less than a certain length. A simple function could be a direct table lookup, etc..

2.5 Processor control of array execution

The main processor has some specific instructions for controling the array. A clock counter controls array execution. While the clock counter is zero, the array is stopped. By setting the clock counter positive, the main processor can make the array execute for a certain number of clock cycles.

2.6 Configuration

Each block in the array requires exactly 64 configuration bits to specify the sources of the inputs, the function of the block, and any wires driven with outputs. Garp allows partial array configurations. The smallest configuration is one row and every configuration must fill exactly some number of contiguous rows. Cache is used to to accelerate the configuration work.

2.7 Array access to memory

Memory accesses can be initiated in the array without direct processor intervention. The memory access can be divided into two phases: memory access request and data transfer. Array memory access

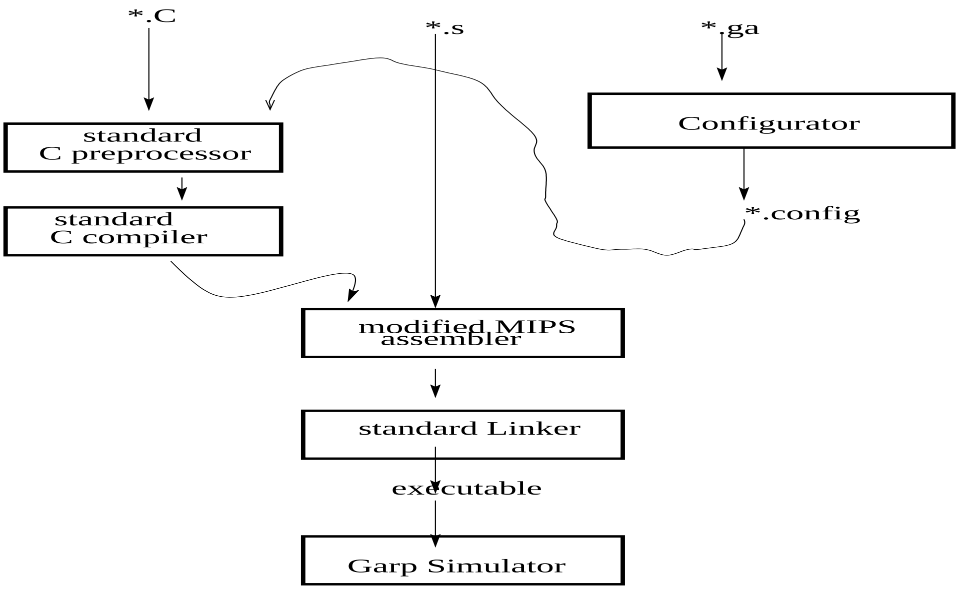

Figure 1: The Garp programming environment.

is controlled by the control blocks at the edge of the array. An address bus runs vertically through the rows. Control signals requesting memory accesses can be generated in the array logic blocks and forwarded by the control blocks to the memory system. The array sees the same memory hierarchy as the main processor. Garp's array contains only a modicum of writable storage– 4 bits per logic block. The reason is to limit the time needed for context switches of running processes.

2.8 Software Environment

The softare path of Garp is summarized in Figure 1.The configurator accepts a human-readable description of a configuration and converts it to the binary representation used by the processor. A configuration is defined as a collection of rows, with each row containing within it logic blocks in specific columns. One such example is as following:

row .a:
{ *– Drive Z registers onto V wires*
4-19: A(Zreg), function(A), Vout(Z);
– Drive D register onto H wires below
4-19: D(Dreg), Hout(D);
}

A complete hardware implementation of Garp does not yet exist, so Garp programs must be executed on a simulator, which is also a part of the whole software environment.

3 PipeRench Architecture

3.1 Introduction

Hardware virtulization is a technique allowing low cost of developing and upgrading applications for reconfigurable computing. The PipeRench project at CMU is targeting to provide a pipeline configuration device by allowing an application to transparently exceed the logical capacity of the physical FPGA at runtime. A pipeline-reconfigurable architecture must have two features:

- Support the configuration of a computataionally significant pipeline stage every cycle, while concurrently executing all other pipeline stages in the FPGA
- Support different pipeline stages to be placed in different absolute locations in the physical device at different times.

PipeRench's stripe configuration is designed to satisfy the above features.

3.2 Stripe Configuration

Striped configuration combines many of the advantages of currently available on-line configuration and multiple-context configuration. The most important idea behind striped configuration is that the atomic unit of reconfiguration of the FPGA has been chosen so that it matches a sensible level of reconfiguration within the application. In an ideal scenario, each of the application's stages would fit into one of the FPGA's stripe. The second basic idea of striped configuration is that the configuration within one stripe can move to a neighboring stripe within the FPGA. This feature is particularly useful for pipelined applications.

3.3 Implementing Pipelined Applications using Run-Time Reconfiguration

Consider applications very deeply pipelined. Assume the application has S identically-sized pipeline stages. Further assume that there are K bytes of data flowing between each stage every cycle. In summary, we have following computing model:

S Total pipeline stages for the application

K data flowing rate(bytes per cycle) between stages

N Number of FPGAs to implement the whole application

T Excution time a pipeline stage takes

The throughput of static, N-FPGA implementation of the application is K/T bytes per second. The theoretical throughput of the 1-FPGA implementation is $K/(NT)$ due to the reduction in computing hardware.

3.4 Component-level Reconfiguration

Using component-level reconfiguratuion, the N different FPGA configuration from the N-FPGA design are used to configure a single FPGA. The configuartion controller loads one configuration, and allows the FPGA to perform operation on X pieces of data. It takes S/N cycles to get the first result, and X-1 cycles to get the remaining results. Therefore, the time required zto complete these computations, in seconds, is

$$T(X-1+S/N)$$

After this computation is complete, the system controller reconfigures the FPGA with the next configuration. If it takes C cycles to reconfigure the FPGA, then the throughput of this implementation can be described as:

$$\text{Throughput} = \frac{KX}{N{*}T{*}(X+\frac{S}{N}-1+C)} = \frac{K}{T{*}(N+\frac{S-N}{X}+\frac{NC}{X})}$$

The throughput falls short of the ideal based on two factor. The first factor is the penalty suffered for having to repeatedly fill and empty the pipeline between reconfigurations. This factor is reflected in the $(S-N)/X$ term. The second factor is the penalty for reconfiguration, which is reflected in the NC/X term. Both od these terms can be reduced by increasing X. However, two problems arise with arbitrarily increasing X. One is that the latency of the implementation increases. The other problem is that it is necessary to have enough memory to store all the data output from one block during reconfiguration so that it can be used as the input to the next block of the pipeline. The required amout of memory is:

$$\text{Memory} = \text{KX}$$

Any type of FPGA can be reconfigured using component-level reconfiguration. It is the only mode possible for standard FPGAs.

3.5 Incremental Pipeline Reconfiguration

Incremental pipeline reconfiguration is a restricted form of local RTR, in which the pipeline is seperated into S components, each corresponding to one pipeline stage. The FPGA can hold S/N of these pipeline stages, and the reconfiguration happens in an incremental manner. During each stage of computation, we add one additional stage to the configuration, and remove a stage if necessary to keep the amount of configuration within the capacity of the FPGA. Figure 2 illustrate this procedure.

Assume that the time to execute a pipeline stage is T, and the number of execution cycles required to reconfigure this entire FPGA is C. Thus the time required to substitute a new pipeline stage into the configuration is, ideally, $TC/(N/S)$. The total reconfiguration time for one sweep through the application will be TCN. Execution of the entire pipeline will take S cycles for the first element of data, and $(S/N)-1$ cycles to get the remaining data out of the configuration. Therefore, the best-case latency and throughput of this implementation are:

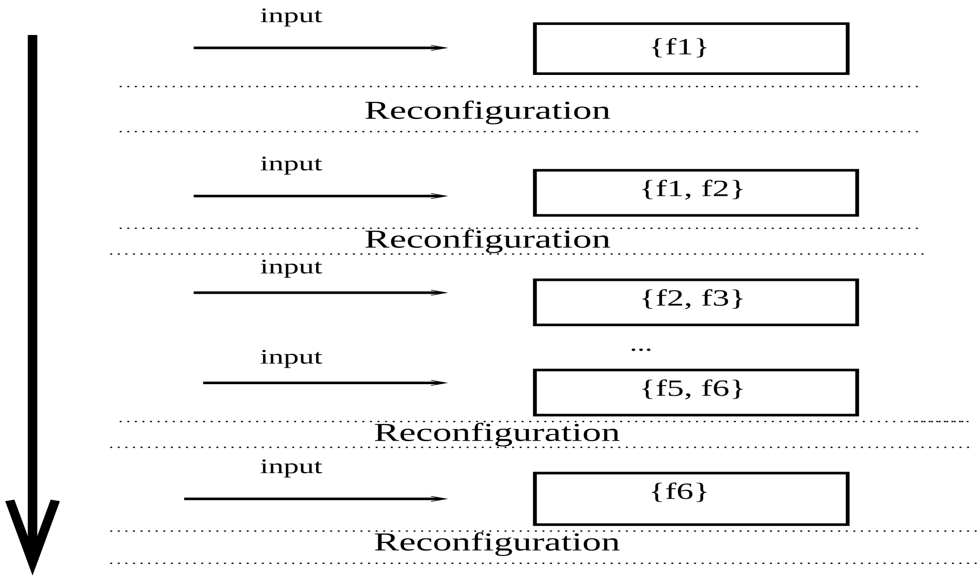

Figure 2: Pipeline Stage Reconfiguration

$$Latency = T(S + \frac{S}{N} - 1 + CN)$$
$$Throughput = \frac{K * \frac{S}{N}}{T * (S + \frac{S}{N} - 1 + CN)} = \frac{K}{T * (N + 1 - \frac{N}{S} + \frac{N^2}{S} * C)}$$

The advantages using incremental pipeline reconfiguration include that the throughput and latency of the incrementally reconfigured FPGA approach their respective theoretical maximums as C approach zero; and that all intermediate results remain stored in the appropriate pipeline stage, no need for supplemental storage.

3.6 Striped configuration control and design

One significant advantage of striped configuration is that the increased size of the atomic unit of reconfiguration eases problems of configuration control and algorithm design for RTR. The current control mechanism for a striped FPGA coprocessor is illstrated in Figure 3.

3.7 FPGA fabric

Three significant constrains must be dealed with when designing a striped FPGA.

- Width of the striped must match the width of the memory that stores the configuration information.
- A stripe should be able to implement a non-trival, pipelineable, portion of the application using a limited budget of configuration bits.

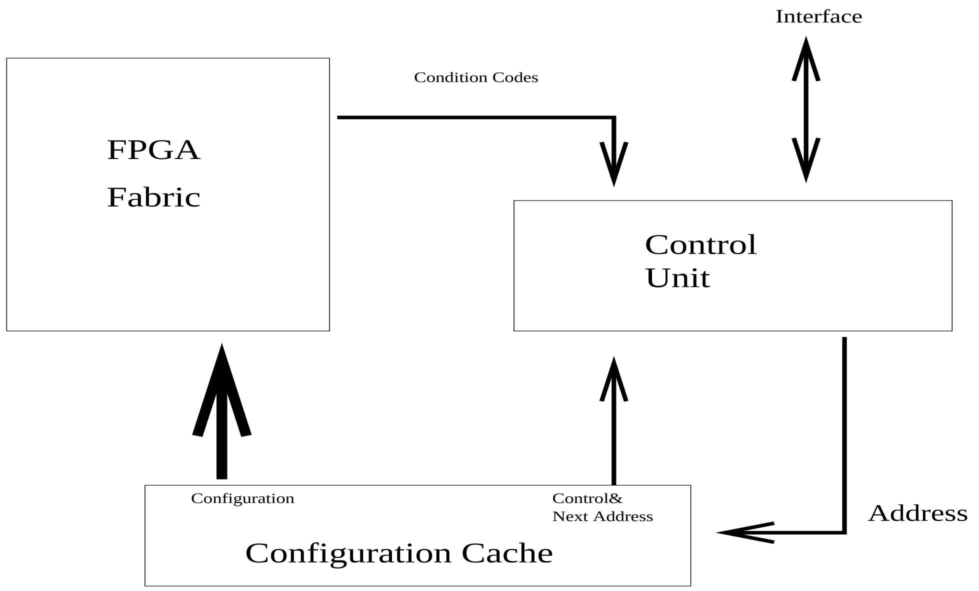

Figure 3: Stripe Control

- The interconnect between stripes must be very regular.

4 Colt CCM (configurable computing machines) architecture

4.1 Introduction

Wormhole run-time reconfiguration is presented as a distributed control methodology that is applicable not only to the problem of device-level CCM reconfiguration, but to system-wide concurrent computing as a whole.

4.2 Wormhole RTR

Wormhole RTR provides a framework for implementing large-scale rapid run-time reconfigurable CCM platforms. It is intended as a method of rapidly creating and modifying custom computational pathways using a distributed control scheme. Similar but not limited to the data movement process in data-flow machines, the onset or completion of computational streams are used to instigate the process of reconfiguration. Wormhole RTR is based on the stream concept. A stream is a concatentation of a programming header and operand data. The programming header is used to configure a computational pathwayd through the system as well as to configure the operations to be performed by the various computational elements along the path.

4.3 Distributed Control

Since the streams independently guide themselves through the system using the information contained in the stream header, the configuration process is inherently distributed. Multiple independent streams can wind their way through the Colt chip simultaneously. It is possible that part of the system can be used to process data operands while any set of neughboring units can be accepting configuration data from the header of one or more other streams, thus allowing overlap of the configuration/execute cycle and of process multitasking within the same system.

4.4 Stream Design Principles

To implement RTR, the control mechanism of the crossbar connecting Colt units must be a distributed system capable of simultaneously switching any set of inputs to a set of outputs, as well as supporting broadcast modes. To accomplish this, each of the crossbar outputs is assigned a unique address. Each switching point from input to output in the crossbar is designed with a small state machine.

References

[1] Garp: A MIPS Processor with a Reconfigurable Coprocessor

[2] Incremental Reconfiguration for Pipelined Applications

[3] Managing Pipeline-Reconfigurable FPGAs

[4] Wormhole Run-time Reconfigration