KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, KSA

WORMHOLE RTR FPGA WITH DISTRIBUTED CONFIGURATION DECOMPRESSION

EXPLORING NEW METHODS FOR FAST RUNTIME RECONFIGURATION

A JOINT PROJECT REPORT FOR:
CSE-670 DESIGN ISSUES OF PROGRAMMABLE ASICS

SUBMITTED TO:
DR. M. E. ELRABAA

BY:
(IN ALPHABETICAL ORDER)
MUSTAFA IMRAN ALI
(st230203)
ALI MUSTAFA ZAIDI
(st230301)

PROJECT INTRODUCTION

In our previous work [1], we explored various methods of achieving fast Runtime Reconfiguration. The following methods were identified as directly applicable to conventional FPGAs:

- Configuration Compression
- Configuration Caching
- Configuration Prefetching
- Partial Runtime Reconfiguration (along with its variants)
- Multi-context Reconfiguration

Finally, we mentioned a novel technique for Runtime Reconfiguration of Coarse-Grained, ALU based Reconfigurable Architectures: Wormhole Runtime Reconfiguration. This technique of reconfiguration has to date been only explored for coarse-grained architectures that have strictly defined dataflow between their basic units. For such architectures, Wormhole RTR promises significant benefits that become increasingly important as fabrication technologies improve and growth in chip densities outpaces the achievable configuration input rate using the more conventional methods.

The Primary Goal of this project was to explore the feasibility of adapting Wormhole RTR, or a similar concept for use with conventional FPGA architectures. An Additional Goal was to further improve run-time reconfiguration performance by exploring novel approaches to Configuration Compression using Variable length Codes (for example Huffman codes), as well as exploring the possibility and potential benefits of a distributed decompression scheme. Thus we have:

- Explore Wormhole RTR for Conventional FPGAs
- Compression Schemes using Variable Length Codes
- Potential and benefits of Distributed Decompression schemes

It is important to note at this point that this study is targeted towards Data-path oriented FPGA Architectures.

CONFIGURATION ISSUES WITH ULTRA-HIGH DENSITY FPGAS

FPGA densities are growing dramatically with process technology improvements. As a result, configuration loading using conventional methods takes increasingly larger amounts of time as newer and denser FPGAs arrive.

DEFINING NEW METRICS FOR RECONFIGURABILITY

As we observed in [1], FPGAs are increasingly used as compute engines that implement data-intensive portions of applications directly in hardware. However, a lack of an efficient method for dynamic reconfiguration of these large FPGAs will lead to such systems being severely underutilized. Support for fast reconfiguration will enable what may be termed as efficient ***temporal scheduling*** of the available resources.

Furthermore it is not guaranteed that the size of tasks implemented in hardware will grow as available hardware increases. This means that as the size of an FPGA grows, the more efficient utilization of its resources would be to share the Large FPGA between different tasks/configurations, i.e. different parts of the same fabric used to implement distinct and independent algorithms in hardware. What is needed is an effective ***spatial scheduling*** technique for optimally utilizing the available FPGA area at any given time.

Traditionally, FPGAs have been only used to accelerate a single application at a time. Multi-tasking on the FPGA was not considered feasible because of the limited amount of available resources. Thus, efficiently supporting either temporal scheduling or spatial scheduling was enough to ensure reasonably efficient utilization of the FPGA's resources by any given application. Consequently, making this distinction between temporal and spatial scheduling was not necessary until now.

As the amount of available computational resources on FPGAs grow beyond what is required by an average application, multi-tasking between several applications will become essential to fully utilizing the available hardware.

In this section, we identify the impact of growing FPGA density on two basic Run-time reconfigurable architectures: The Partial RTR model, which allows for effective spatial scheduling, and the Multi-Context FPGA model, whose forte is efficient temporal scheduling. We highlight potential issues with scalability, and determine how effective both these models are in terms of supporting efficient multi-tasking, with growing FPGA sizes.

SCALABILITY ISSUES WITH THE MULTI-CONTEXT APPROACH

The basic concept behind multi context FPGAs with respect to Run-time reconfiguration is that new configurations may be loaded into the idle contexts while a configuration is running on the active context. In this way, Multi-context devices attempt to hide configuration latency with overlapping configuration with computation.

Multi-context FPGAs have traditionally provided excellent temporal scheduling for individual applications. However, it is not possible to efficiently schedule the available spatial resources in each context. This is due to the fact that in the basic multi-context model, each context is programmed

serially. Spatial scheduling on a serially programmed FPGA can only be done at compile-time, thus significantly limiting its effectiveness.

As FPGA sizes grow, the following issues become visible:

- The amount of configuration information per context grows, because there are more resources available on a larger FPGA. Thus each context takes longer and longer to be configured as configuration is still being done serially. This makes it increasingly difficult to hide configuration latency by overlapping configuration with computation.
- Since Multi-context devices have traditionally not been effective in providing efficient spatial scheduling, the burden of multitasking applications falls squarely on its temporal scheduling prowess. The effectiveness of temporal scheduling here is limited by the number of available contexts.

As we can see, as FPGA densities grow, the temporal-scheduling ability of the Multi-tasking approach becomes increasingly less effective. This problem is exacerbated by the fact that lack of efficient spatial scheduling ability leads to only a very few active configurations being able to run simultaneously. Given the dramatic overheads involved with implementing the multi-context FPGA model, this approach is becoming increasingly inefficient as FPGA densities grow.

SCALABILITY ISSUES WITH THE PARTIAL RTR APPROACH

The basic concept for the Partial RTR model is that the configuration memory is addressable and can therefore be accessed like RAM. In such a scenario, it may be possible to configure one part of the FPGA, while an active configuration runs on another part. This feature has allowed individual applications to implement efficient run-time spatial scheduling of different configurations.

However, Partial RTR FPGAs generally have only a single configuration port, thus limiting the reconfiguration to be undertaken by one application at a time. In order to use a PRTR FPGA for multitasking, access to the configuration port must be arbitrated between applications, so that different applications may only configure the fabric in sequence. Due to this serialization constraint, Partial RTR FPGAs will find increasingly inefficient utilization of their available resources as these resources grow.

Furthermore, the centralized configuration resources (namely the row-and column decoders, select lines, and the multiple data busses) bring into question the issue of their scalability: as chip sizes increase, signals may require multiple clock cycles to traverse long wires. Also, the row and column decoders will have to span the length and width of the entire chip!

A stop-gap solution to the problem of serialization can be to increase the number of configuration ports, in a similar fashion as is done for multi-ported memories and register-files. However, it has been noted in literature that increasing the number of parallel access ports to a memory can result in a quadratic increase in the required area.

WHAT FUTURE FPGAS NEED

We can see from the above that for efficient utilization of ultra-high-density FPGAs, the following two items are essential:

- Support for efficient multitasking. This means that both temporal and spatial scheduling must be efficiently supported.
- This support for multi-tasking must be scalable.

In order to fully understand the last point about scalability, let us consider the following: Efficient temporal and spatial scheduling may be supported by a multi-context FPGA architecture in which each context is Partially-reconfigurable. Although this forms a logical extension of the two ideas, it can easily be seen that temporal scheduling is still limited by the number of contexts available, while spatial scheduling is still limited by the number of configuration ports per context. Increasing either would result in a dramatic increase in overheads. Thus this approach can not necessarily be considered scalable.

One approach that potentially satisfies both these criteria is the Wormhole RTR approach. The problem is that Wormhole RTR has to date only been explored for coarse-grained architectures with clearly defined dataflow. In this project, we aim to explore this approach in terms of its applicability towards conventional FPGA Architectures.

In order to further improve run-time reconfiguration speed, we have also explored the use of variable length compression schemes such as Huffman encoding etc, to represent frequently used logic-block configurations. This is coupled with a distributed configuration decompression scheme that is very well suited to the Wormhole RTR approach in that it reduces the length of the configuration streams.

ADAPTING WORMHOLE RTR FOR CONVENTIONAL FPGAS

Introduced in 1997 by [7], Wormhole RTR is a method of reconfiguration for coarse-grained reconfigurable architectures that have strictly defined dataflow between their basic programmable units. The WRTR approach attempts to address the issue of reconfiguration scalability by advocating fully distributed control resources, as well as shared use of I/O ports for configuration as well as data.

This has the primary advantage of allowing different parts of the same reconfigurable resource to be configured independently and simultaneously, allowing for efficient implementation of both spatial and temporal scheduling. Furthermore, distributed control provides scalability: multiple applications are capable of configuring and utilizing the same fabric simultaneously, thus allowing larger, denser FPGAs to be utilized more efficiently.

Additional advantages of having no centralized resources include:

- Fewer single point failures that can lead to total system failure: since all resources are fully distributed, it is unlikely that any single point of failure will prove to be catastrophic.
- Increased resilience: using WRTR, it may be possible to isolate and route around faults in the chip that are introduced during fabrication, potentially improving chip yields?
- Eliminates configuration bottleneck: no one resource may become the performance bottleneck, as all resources are distributed, including configuration ports.

ORIGINAL IDEA FOR WORMHOLE RTR

Before we can consider adapting Wormhole RTR to conventional FPGAs, it is necessary to understand the context in which WRTR was developed. This is essential to avoid ourselves being limited by unnecessary constraints that were valid for its original domain, but may not be applicable to our domain.

Wormhole RTR was originally intended as a method of rapidly creating and modifying ‘custom computational pathways’ in a coarse-grained fabric. The basic essence of the WRTR concept was: the implementation of computation through the use of independent, self steering streams that carried both configuration information as well as the data operands to be operated upon.

The main features of the proposed WRTR concept were:

- The configuration information is stored in the header of the stream, and determined the sequence of operations to be performed on the data, as well as the path of the stream through the configurable fabric.
- Runtime determination of the path of the stream is possible, allowing allocation of resources as they become available. This also implies that this model was used specifically for implementing operations suitably represented as long dataflow pipelines.

For detailed treatment of the original proposal for Wormhole RTR, please refer to [8]

ISSUES IN ADAPTING WRTR FOR CONVENTIONAL FPGAS

Conventional FPGAs implement algorithms in the form of hardware circuits, and the dataflow through the implemented circuit is completely unrelated to the manner in which configuration is carried out. Furthermore, circuits implemented on FPGAs have arbitrary structures, and can not be limited to be long, linear pipelines as was the case for the original WRTR implementation concept. In light of these considerations, we can make the following observations:

- As dataflow is completely decoupled from the configuration flow, WRTR in FPGAs may only be used for deploying configuration information.
- Configurations in FPGAs are not necessarily implemented as long, linear pipelines of arbitrary shape and length: each configuration has a rigidly defined structure, and therefore configuration streams cannot be allowed to determine their path at runtime.
- In the original concept, the I/O ports made no distinction between Configuration information and the data operands. This was due to the fact that both were included in the same stream. However, for conventional FPGAs, since WRTR may only be used for configuration information, a distinction must be made between configuration information and data at the I/O ports. The I/O ports must be multiplexed between the implemented circuits, and the Wormhole configuration busses.

Our aims in adapting Wormhole RTR for FPGAs may thus be listed as:

- To devise a fast, parallel, and scalable system for reconfiguration of FPGAs…
- …that incurs minimum area overhead, and …
- …imposes minimum amount of restrictions on the underlying FPGA Architecture.

We will be using the original WRTR concept primarily as a source of inspiration for achieving these goals: it is not necessary that the original idea be followed to the letter.

PROPOSED FPGA MODELS

Our aim was to devise a high-speed, distributed configuration model with all the benefits of the original WRTR concept, but with minimum overhead. To this end, we developed three different FPGA configuration models that may be treated as representative points in the design spectrum of WRTR based FPGAs. Consequently, each model offers a different set to tradeoffs.

In this section, we discuss each of these models, identifying the pros and cons of each, as well as defining their basic area models in order to compare relative area overheads. Determination of the actual utility and effectiveness these models, however, can only be done empirically.

CONSTRAINTS ON THE UNDERLYING FPGA ARCHITECTURE

All of the Architectures that are based on the WRTR concept are essentially partially reconfigurable – different parts of the system can be configured to implement different operations, and can be operating simultaneously. This means that our proposed Wormhole RTR models suffer from exactly the same issues of handling 'parasitic configurations' as any partially reconfigurable FPGA. The concept of parasitic configurations was discussed in our last report [1].

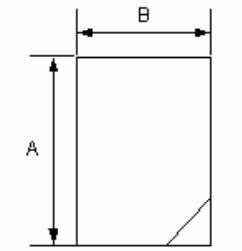
Modern Partially Reconfigurable FPGAs such as the Xilinx XC6200, and the Virtex series FPGAs employ special enhancements to their routing architectures in order to handle parasitic configurations. Although scant details are available about these enhancements, it is known that they guarantee that an FPGA will never be damaged by any configuration pattern. Furthermore, virtually no constraints are imposed on the structure of the FPGA itself. However, [2] states that these enhancements result in an area increase of approximately 25-30 percent, for any given FPGA.

THE BASIC AREA MODEL

In order to make an assessment of the required area overheads of our proposed FPGA models, we will compare our models with a generalized Partial RTR FPGA Model. For all FPGAs, we shall assume that the enhancements ensuring prevention of parasitic configurations have been employed.

Given below is the basic building block of our area model. This block is considered the basic unit of reconfigurability, i.e. each of these blocks may only be programmed as a whole. As a result, the size of this basic block determines the granularity of reconfiguration: a larger block means coarser granularity. Ideally, we would want the finest possible granularity, so that a finer degree of control may be achieved over the allocation of resources to different configurations.

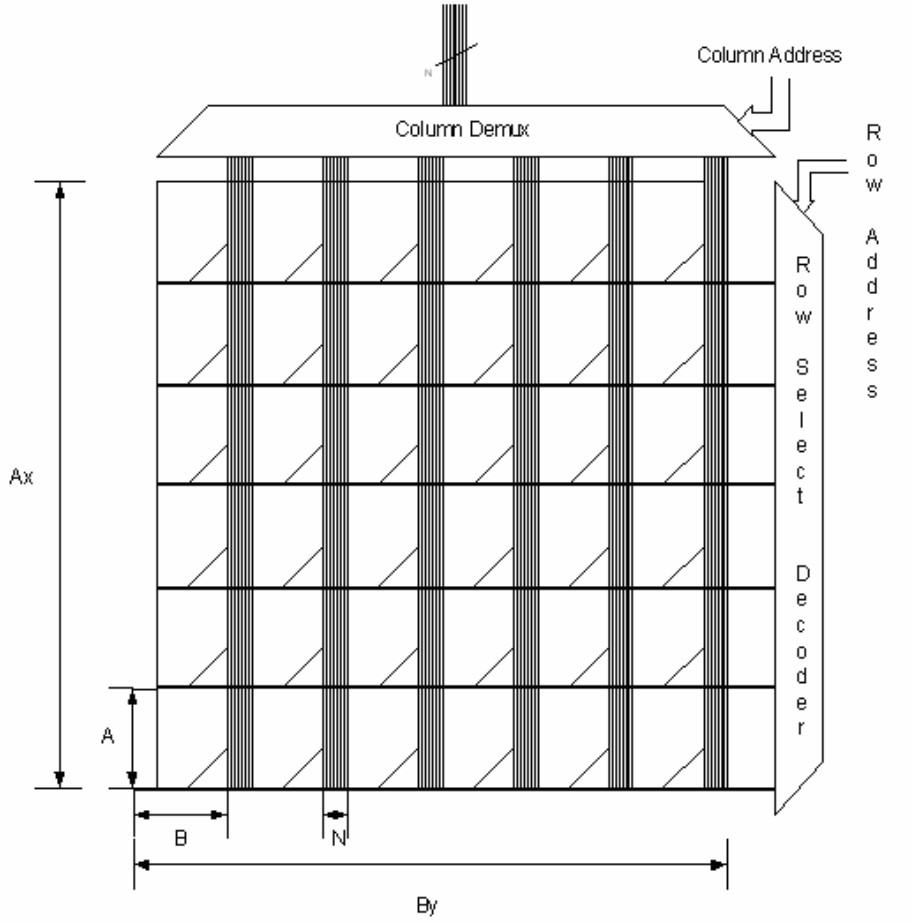
The amount of programmable resources within each block is determined by the parameters A and B, which represent the length and width of the basic configurable block in terms of basic gates or CLBs.



Each of the above blocks can essentially be seen as a small FPGA. In order to have minimum area within the block, we assume that the logic resources within are programmed by means of parallel scan-chains, so that the unit area taken up is only larger that that of a serially programmable FPGA because of the special enhancements to avoid parasitic configurations.

THE GENERALIZED PARTIAL RTR FPGA MODEL

Given below is given a generalized Partial RTR FPGA Model based on the above basic block. This model will serve as the basis for comparison to our proposed Wormhole RTR FPGA Models.



The above model is composed of an 'x * y' number of the basic building blocks, plus additional overheads required to support the Partial RTR paradigm. The total area can be given as:

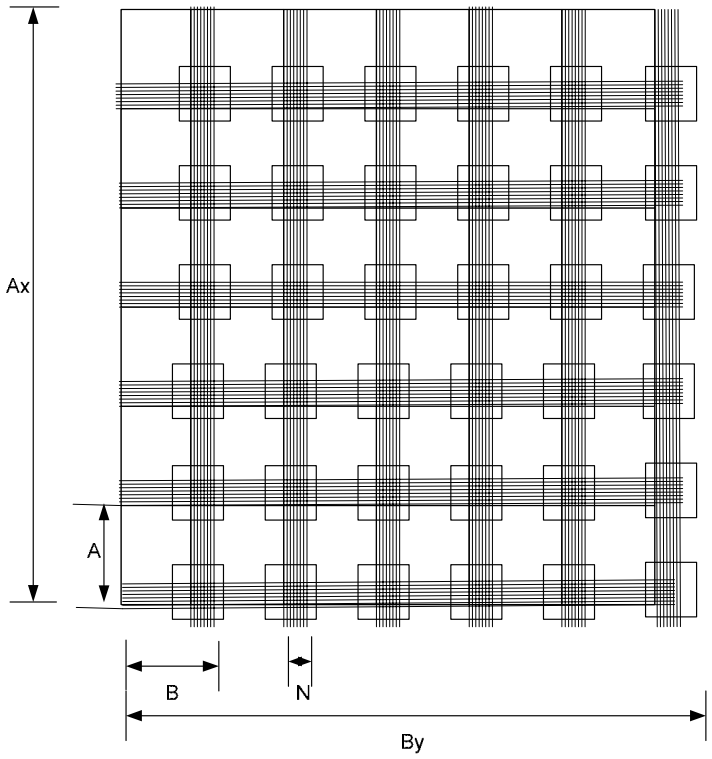
AREA = Area of Basic model + Overheads

Area of Basic Model = (A * B) * (x * y)
Overheads = Area of 'log2(x)-to-x' Row-select decoder + Area of 'log2(y)-to-y' n-bit Column De-multiplexer + Area of 1 N-bit bus * y

The configuration port is N bits wide, thus requiring y N bit busses, one for each column of Basic blocks. Also required are the Row select decoder, and the Column Select De-multiplexer.

PROPOSED MODEL 1: WRTR FPGA WITH INTERNAL ROUTING

Given below is the diagram for this model:



Description of this model:

- The FPGA I/O ports are multiplexed between the CLBs in each basic block, and the wormhole configuration bus adjacent to the block.
- Each of the basic configurable blocks is accompanied by a simple configuration stream wormhole router.
- Configuration Streams entering through any of the ports may be routed to configure any block in the FPGA by the wormhole routers.
- Overhead due to the routers and busses scales linearly with FPGA Size.

Expected Issues with this model:

- It is expected that each of the routers will incur significant overhead in terms of area, thereby affecting the optimal granularity of the FPGA – if the overhead per block is high, the block size may have to be increased to amortize the cost of this overhead.
- As the routers may have to deal with multiple independent configuration streams at the same time, the need arises for the inclusion of arbitration mechanisms. Furthermore, it is even possible that multiple configuration streams may create deadlock situations. All these issues will need to be addressed at various levels. In the next chapter, we propose simple mechanisms for arbitration and deadlock detection.

The area model for this design:

AREA = Area of Basic Model + Overheads

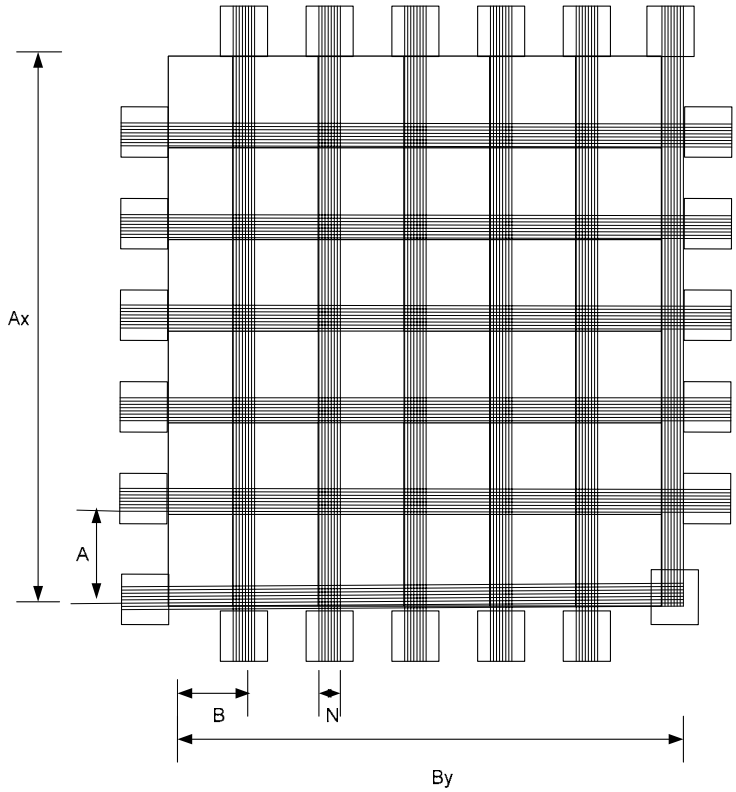
Overheads = Area of 1 n-bit bus * 2x + Area of 1 n-bit bus * 2y + Area of 1 4-D Router block * [x * y].

In the above equation, the areas occupied by the busses can easily be found for any specified fabrication technology. So the only unknown in the above equation is the area of the 4-D Router. In the next chapter, we develop a rudimentary model for the 4-D router, from which its area may be estimated.

PROPOSED MODEL 2: WRTR FPGA WITH PERIMETER ONLY ROUTING

When we designed the first model, it was felt at the time that the large overhead of associating wormhole routers with each basic block would limit the configuration granularity to a relatively coarse level. In an effort to reduce the overhead of the routers and improve the achievable granularity, we reasoned that perhaps there is no need to associate routers with every block. Instead, routers could be limited to the periphery of the FPGA, routing configuration streams into other rows or columns

Given below is the diagram for this model:



The benefits of this approach over the last are:

- Overhead scaling is improved, as FPGA size grows.
- Finer Granularity of Basic Blocks is achievable.

The potential disadvantages of this approach are:

- It may take a longer time to reach distant parts of the FPGA as sizes grow.
- Since all streams are routed at the perimeter, reconfiguration time may suffer due to increased arbitration delays at the routers. Also, there is a considerably increased risk of deadlock, as we now have a much smaller number of routers handling potentially the same amount of configuration streams.

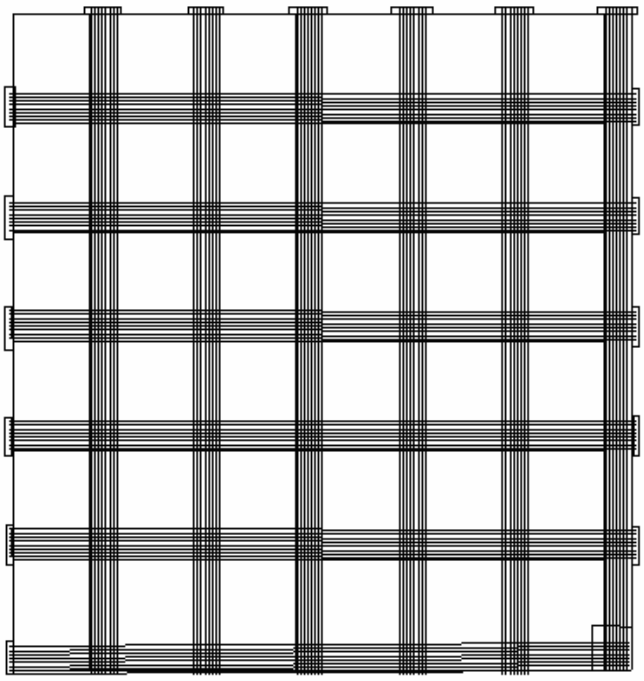
The area model for this design:

AREA = Area of Basic Model + Overheads

Overheads = Area of 1 n-bit bus * 2x + Area of 1 n-bit bus * 2y + Area of 1 3-D Router block * [2(x + y) – 1]

PROPOSED MODEL 3: PACKET BASED FPGA MODEL

In the previous model, we attempted to mitigate the router overhead by limiting the routers to the periphery of the system. However, we realized that this could cause a dramatic increase in routing delays. In the third model we devised, we considered that perhaps routers could be done away with altogether. Instead, we would impose a constraint on the host system that configuration data for each block to be configured be routed only to the port(s) associated with appropriate row or column of the FPGA that contains the block to be programmed. The diagram for this model is given below:



In order to utilize such a design, a centralized external resource is required that receives different configuration streams from various sources, extracts from the streams individual ‘packets’ that

contain configuration information for a single block, then groups these packets into new streams according to which row or column their target block resides in. The newly compiled stream, which only contains configuration information for blocks in the same row or column, is then routed directly into the correct row or column.

The benefits of this approach are:

- Zero area overhead due to routers
- No arbitration issues or deadlocks.
- Extremely simple architecture.

The obvious drawback of this approach is the external centralized control resource. As was seen in the last chapter, as FPGA densities increase centralized resources tend to become bottlenecks to configuration performance. As a result, we have suspended any further consideration of this model as it does not meet our scalability requirement, although it is still an interesting approach that can find potential for use in small embedded systems that require fast parallel reconfiguration.

The area model for this design:

AREA = Area of Basic Model + Overheads

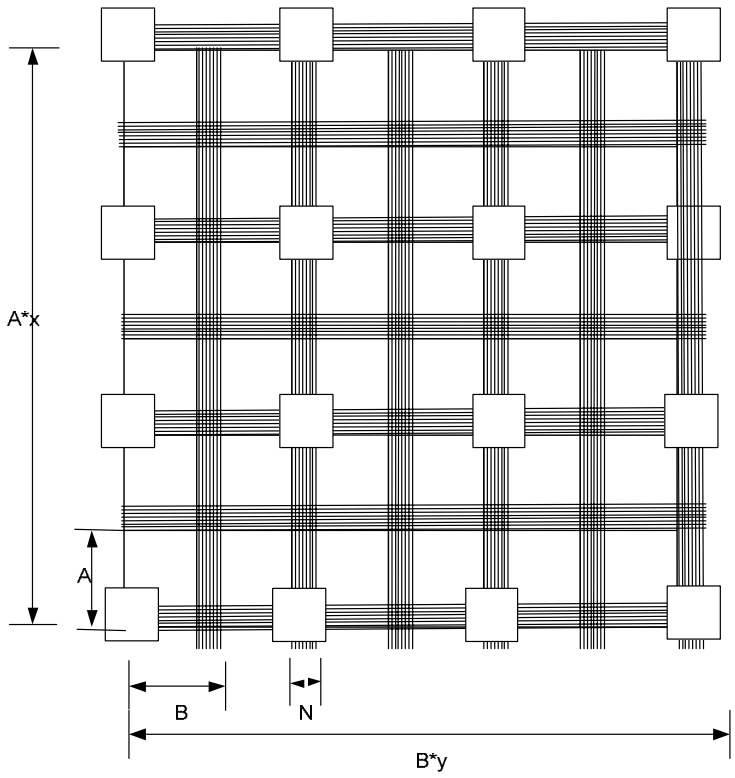
Overheads = Area of 1 n-bit bus * 2x + Area of 1 n-bit bus * 2y

Additional area overhead is incurred because of the external control resource that must be implemented. However, as does not necessarily reside on the same chip as the FPGA fabric, we have excluded it from consideration in the area model. This is acceptable because we are comparing the area overheads of various FPGA models, and are not as yet concerned with system level issues. The overhead of the external controller must however be included in any evaluation of the overall system.

ANALYSIS

The first two Wormhole RTR FPGA models that we have proposed represent extreme points of possibility from a spectrum of approaches that can be considered when adapting WRTR for FPGAs. Consequently they have rigidly defined pros and cons. It is also possible to explore compromises between these models as a means of improving features, while reducing overheads.

For example: instead of having routers associated with every basic block, they may be associated with every alternate block, or every *n*th block in both horizontal and vertical directions. As a result, the area between routers may be considered similar to the second model, while the overall design supports the flexibility of the first model. An example of such a scheme is given in the figure below.


A*x
A
B
N
B*y

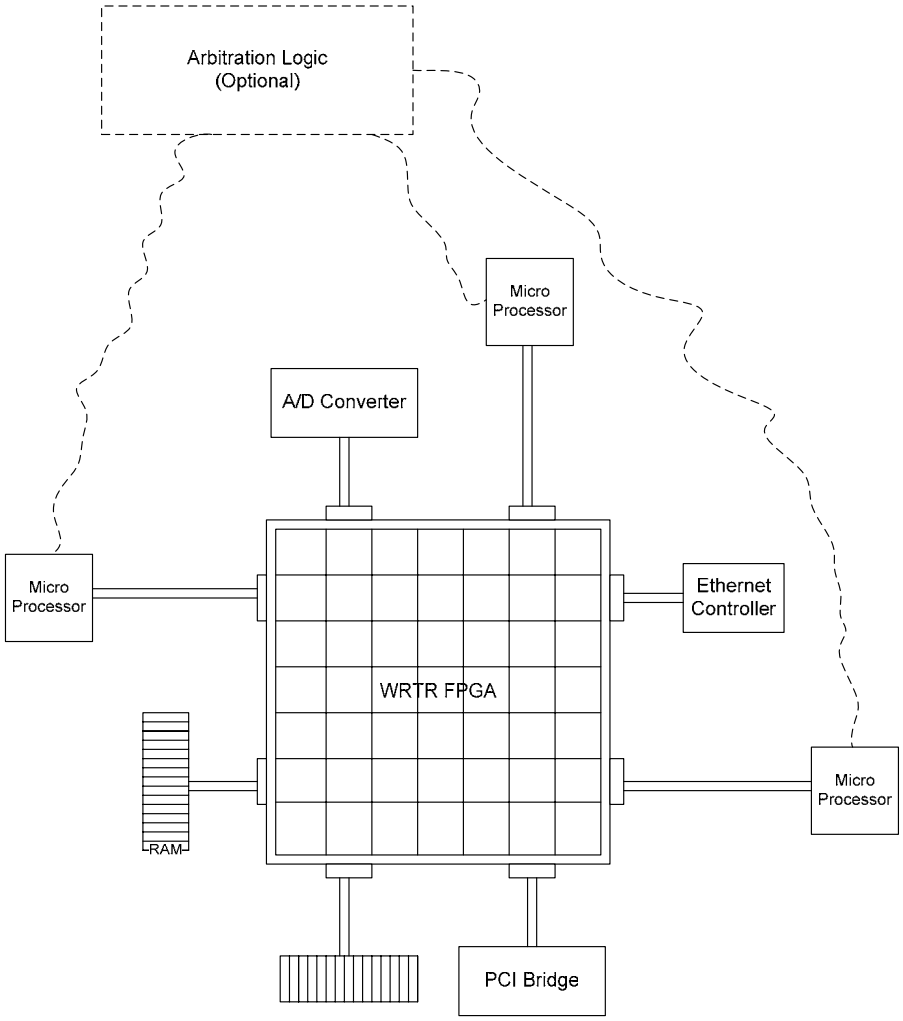
SYSTEM LEVEL ISSUES

In this chapter, we consider the finer details of this new reconfiguration paradigm. We take a closer look at how WRTR FPGAs may be incorporated into systems, and how we can handle basic arbitration tasks at the router, such as stalling a configuration stream and detecting deadlocks. We also look at a rudimentary router design, and propose a flexible structure for the configuration stream itself.

SYSTEM DESIGNS

Due to its support for scalable multi-tasking as well as temporal and spatial scheduling of tasks, a single WRTR FPGA can be efficiently utilized by multiple hosts and multiple applications. This shows an additional advantage of WRTR FPGAs: a single WRTR FPGA that has adequate capacity can be used to accelerate tasks for multiple host processors when otherwise each processor would have its own dedicated acceleration engine, potentially reducing chip count.

We tentatively propose two kinds of system designs for utilizing a WRTR FPGA, based on the diagram below:



The two types of systems are:

1. Multiple hosts with Arbitration Logic
2. Multiple Hosts without Arbitration Logic.

Before we consider each of these types of systems, let us first understand the sample system presented above. The diagram may be explained thus:

- Control of the numerous multiplexed IO/configuration ports is essentially divided between the various host processors.
- Some of the ports are dedicated as data only by the system implementation, for e.g. the ports connected to the A/D converter, or the RAM banks.
- Each processor is responsible for generating the configuration stream that it will use to program the configuration fabric. Processors insert the stream through the port that is dedicated to them. Each processor host is allowed access to the ***entire resources*** of the WRTR FPGA, provided that they are not being used by the other processors in the system.
- In this basic model we assume that each host processor runs a single application at a time.

MULTIPLE HOSTS WITHOUT ARBITRATION LOGIC

The basic idea here is that there is no direct communication or coordination between any of the host processors. Instead, each host assumes that the entire WRTR FPGA fabric is at its disposal, and each host launches configuration streams into the fabric with this assumption. The benefits of such an approach would be:

- No need for centralized arbitration circuitry.
- No artificial restrictions are imposed on the hosts, limiting the amount of fabric they can use at any time.

The obvious drawbacks of such an approach are:

- Increased contention for resources between configuration streams within the FPGA.
- Potential for having a running configuration of one host being overwritten by a configuration stream from another host. We will need to introduce a 'resource locking' mechanism that prevents such a scenario.
- Configuration latency is not guaranteed.

MULTIPLE HOSTS WITH ARBITRATION LOGIC

Based on the issues we faced above, we may choose to include arbitration logic that is loosely coupled with the entire system. In this scenario, each processor is allotted perhaps a rectangular portion of the FPGA fabric, based on its requirements at the time, as well as the overall system load (i.e. also considering the requirements of the other hosts). This allotment is decided by the arbitration logic that keeps track of the allotted areas and the requirements of all hosts. The simple benefits of such a system are:

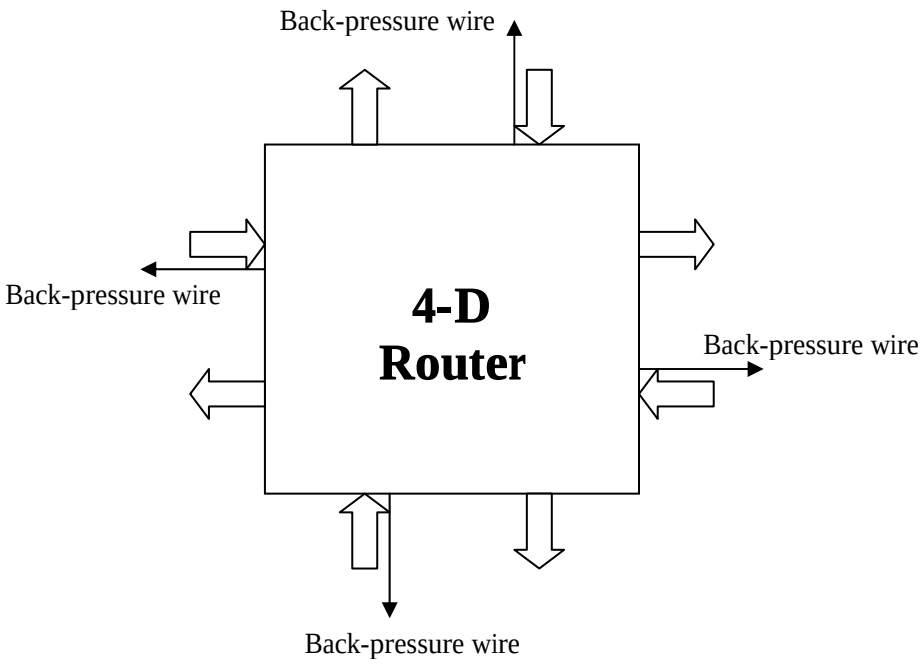
- Minimized contention for resources between configuration streams of different hosts.
- Minimized arbitration issues and reduced potential for deadlock situations.

One potential drawback is that this approach may lead to a less efficient utilization of resources. However, it needs to be determined empirically as to which approach is more practical.

HANDLING ARBITRATION ISSUES AND DEADLOCKS

ARBITRATION ISSUES

It is possible that a configuration stream arriving at a wormhole router within the FPGA needs to travel to a direction that is already receiving another configuration stream. In such a case, it becomes necessary to stall the newly arrived stream to prevent loss of configuration information. Furthermore, this stalling must propagate back through multiple routers all the way to the end of the stream. To this end, we introduce a 'back-pressure' mechanism associated with every input port of the router. This mechanism simply sets a signal traveling in the opposite direction to the incoming stream, informing the last source router that the stream must be stalled. The wire carrying this signal is shown below:



This mechanism is explored in greater detail in the next section.

DEADLOCK DETECTION

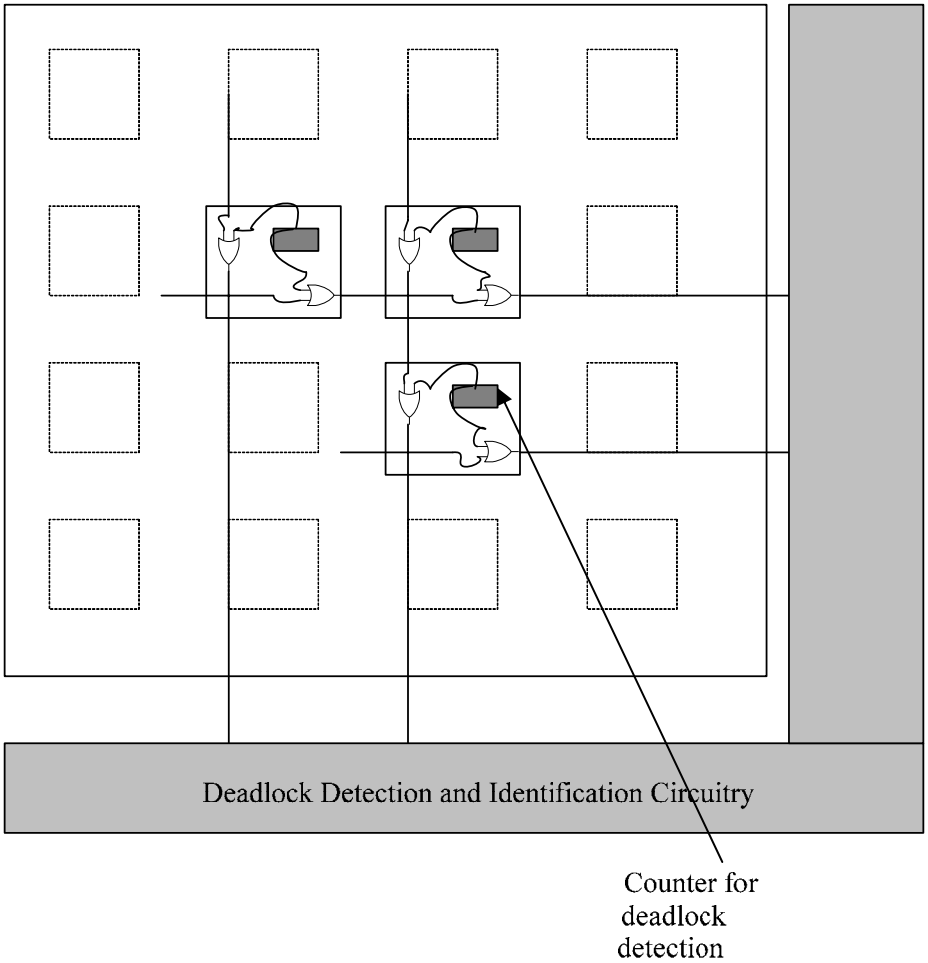
Another issue of concern is the potential for occurrence of deadlocks between two or more streams (deadlock is even possible for a single, poorly compiled stream that attempts to loop back on itself). Although there are two basic methods of handling deadlocks:

- Deadlock prevention
- Deadlock detection and recovery,

we have opted for the simpler second method. The basic idea is to have a counter of predetermined size associated with each input port of the router. The counter is activated as soon as a back-pressure signal is applied to that particular port, and is reset when the back-pressure signal is removed. The essence of the idea is that we do wish for any configuration stream to be stalled for any more than a fixed number of clock cycles. Thus as soon as that number is exceeded by the counter, a deadlock signal is activated.

THE DEADLOCK SIGNALING MECHANISM

The signaling mechanism is extremely simple, yet still allows the external environment to identify where the deadlock occurred in the Fabric. A single, un-pipelined wire traverses the entire length of every row and every column that contains a router. At each router, the deadlock wire is ORed with the local deadlock signal, and passed to the next router, until the end of the chip. Thus the deadlock signal from the two edges of the FPGA may be retrieved by a deadlock detection circuit that attempts to identify the general area of the deadlock, so that only the selected parts of the system may be reinitialized, instead of the whole system. It may even be possible to extrapolate as to which streams caused the deadlock, so that only those streams can be reinitialized. As an example, consider the diagram below. Only the three deadlocked routers are shown, and have set their deadlock signal high, allowing the external environment to identify the area where the deadlock has occurred.



CONFIGURATION STREAM STRUCTURE

At this point, we would like to briefly identify the structure of the configuration stream. Considering that there are several types of information packed into the stream (compressed configuration for datapath logic, uncompressed configuration for random logic, stream start and end markers, direction information etc.) it was decided that the stream structure be similar to the output of a compressed stream as is generated for example by the JPEG encoding process. The JPEG compressed stream is essentially a set of different types of data packed into a single stream, with each type of data being identified by the marker that precedes it.

In our design, since we are using a distributed decompression scheme, only the start and end markers, and the direction information are read by the router, while all other markers are to be decoded by the decompression logic of the destination block.

ROUTING METHODOLOGY

Two basic types of Routing methodologies can be considered:

- Active Routing,
- Passive Routing.

Active routing is much like modern network routing in that it makes use of routing tables, to determine the best possible path for data to travel towards its destination. The primary benefits of this approach are:

- In the event of a routing conflict configuration streams need not be stalled, instead they may be redirected
- Some deadlocks conditions may be resolved locally, instead of requiring full re-initialization of the FPGA.

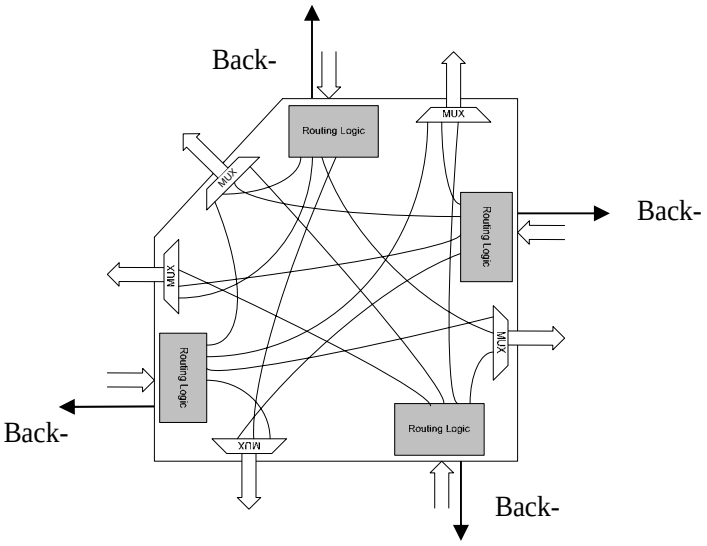
The primary disadvantage is the high complexity of this design, which would incur significant overhead on the FPGA, and would limit the block granularity to a coarser level.

On the other hand, in passive routing we assume that the directional information is embedded within the configuration stream, and the router mechanism simply server as a switch, directing the stream along its desired path.

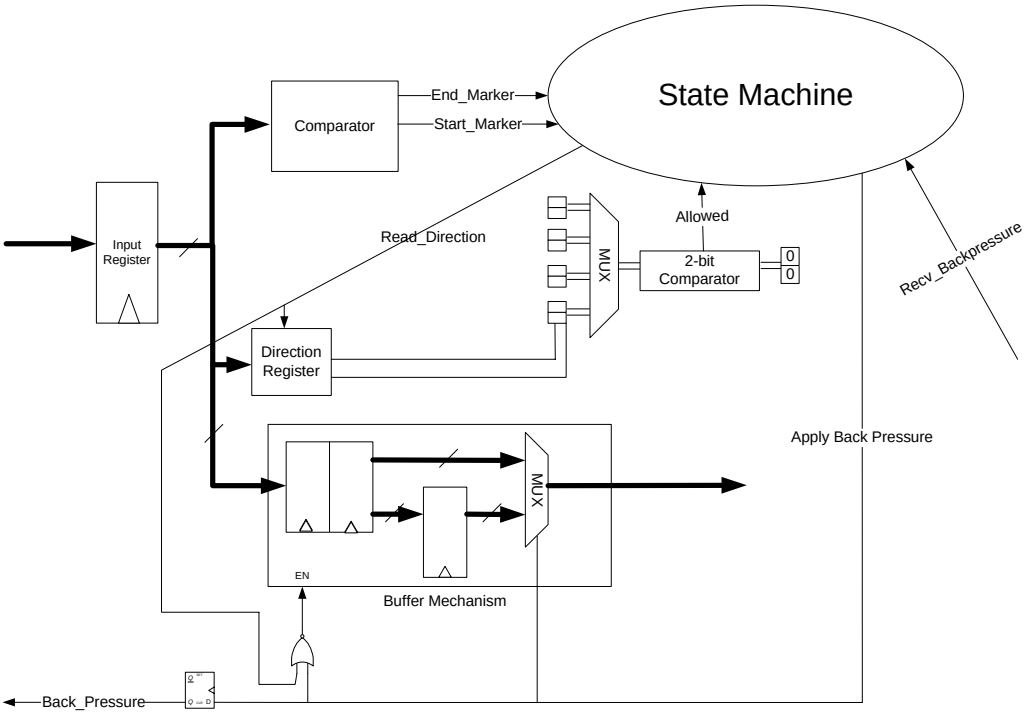
The primary benefit here is the simplicity and small size of the design. Furthermore, since the wormhole network is used exclusively for configuration, and not data, it seems overkill to include large active routing elements that are used only for a small percentage of the total operational time (assuming that the execution time for an average configuration is longer than the time taken to load the configuration.) Because of these reasons, we have selected the Passive router design for our WRTR FPGA models.

In order to complete our comparison of the relative overheads associated with our WRTR FPGA models, we need to approximate the area of the routing logic. To do this, we have designed a simple passive router. Given below are the details for this design.

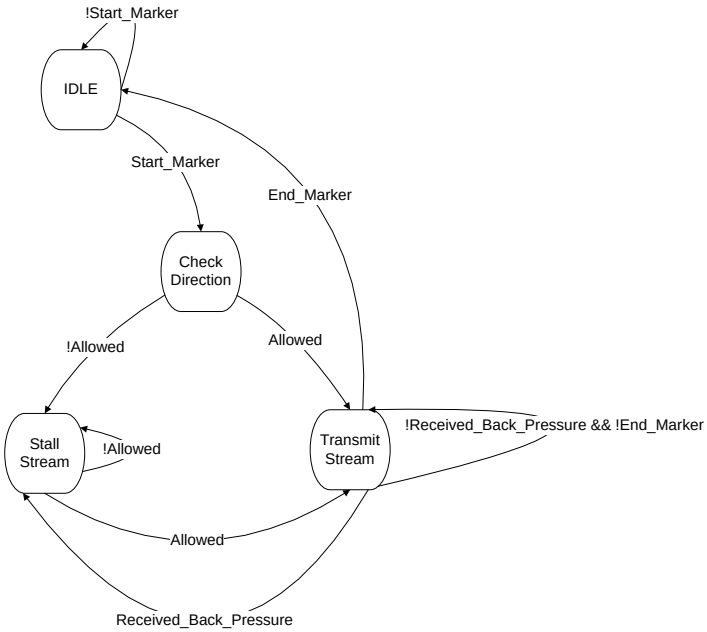
ROUTER STRUCTURE



ROUTER SUB-BLOCK ARCHITECTURE



STATE-MACHINE FOR THE ROUTER SUB-BLOCK



ESTIMATE FOR REQUIRED LOGIC

From the above structures, we can estimate the approximate amount of logic required by a single 4-D router, as well as a single 3-D router. This logic may then be normalized into gate-count etc, in order to estimate the area overhead of the various WRTR FPGA models proposed in the last chapter.

We have the following estimate for logic in a single router sub-block:

- 4 N-Bit registers,
- 1 N-bit comparator
- 1 2-bit comparator
- 1 2-bit 4-1 mux
- 5 flipflops (2 for state, 2 for direction register, 1 for back pressure)
- 1 N-bit 2-1 Mux
- some state-machine logic (maximum 4 * 4-LUTs worth, as there are two inputs to state machine and 2 states)

Now for a complete 4-D router block, we will have 4 of these sub-blocks and 5 additional n-bit 4-1 Multiplexers. From this estimate for logic content, we can easily derive equivalent gates, and compare with out basic PRTR FPGA model.

VARIABLE LENGTH CONFIGURATION MECHANISMS

Configuration data size is huge for large FPGAs [VirtexII and StratixII]. If context switching is to be employed for large applications, configuration bit streams have to be reduced in size to make fast context switching possible. The idea being explored in this work is that of variable length configurations for the logic blocks and associated routing. The aim of variable length configuration is to provide minimal number of bits for a (re)configuration that are based on the frequency of occurrence of that configuration. In datapath based applications, which this work targets, there are distinct pattern in configurations for datapath functionality implementations that can be identified and minimally coded.

CONFIGURATION COMPRESSION/DECOMPRESSION APPROACHES

Configuration overhead minimization can be achieved by removing the redundancy in configuration data, or in other words, compressing the contents configuration bit stream. This will minimize the information that is required to be conveyed during configuration or reconfiguration. It can be achieved by applying some sort of compression to the configuration data stream. The compression mechanism explored in the literature can be termed as the centralized approach [3, 4, 5] in which the configuration data is decompressed at the boundary of the FPGA. The new paradigm explored in this work is termed as the distributed approach that advocates decompressing data at the boundary of the logic blocks or a logic cluster. Each of these approaches is described in the next two sections.

CENTRALIZED CONFIGURATION DECOMPRESSION

The centralized approach has been studied in [6] for Xilinx XC6200 and Virtex FPGAs, aiding the partial run-time reconfiguration of these devices. Significant reductions in configuration data stream sizes have been reported. Lossless compression techniques such as run-length encoding and Lemple-Ziv based compression techniques are employed. The advantages of this approach are that it requires decompression hardware to be implemented only at the boundary of the device and thus minimizing the support hardware overhead. Commercial FPGA such as Atmel 6000 also employ support for configuration compression. The limitation of this approach is that the more efficient lossless compression schemes such as variable length coding are not easy to use because of the large number of symbol possibilities and no defined boundaries (symbols) in configuration data. In case of a heterogeneous device, which can have different types of blocks, it is still more difficult to quantify symbols efficiently in configuration data stream.

DECENTRALIZED APPROACH

The decentralized approach relies on the idea of frequently used configuration patterns. The idea is largely applicable to datapath based designs that have well defined logic block configurations for basic datapath elements which are repeated frequently in the data stream. As such, a Huffman like coding scheme can be used to compress the data stream since it now has well defined symbols in the form of quantifiable configurations of logic blocks. The task of configuration frequency analysis can be handled by the configuration generation software. The software can optimally generate a list of codes for a specified number of frequently used configurations. It should be noted that not every possible configuration will be encoded. This is especially true for random logic configurations and routing switches configuration, which will have to be transmitted in full uncompressed form. The reason for this being the large number possibilities for these configurations and their non-repeating and/or non-quantifiable nature. In order to handle uncompressed data among compressed bit

stream, a special code can be inserted before the uncompressed configuration data to identify that the data following the code is to be handled separately.

Distributed configuration also has advantages that are specific to wormhole RTR. If worms are decompressed at the boundary of the device, this will result in longer internal worm lengths and thus leading to greater worm blockages and having to deal with arbitration issues that can negate the benefits of worm hole RTR. The decentralized approach favors shorter worm lengths, thus allowing more of parallel worms to traverse with fewer blockages.

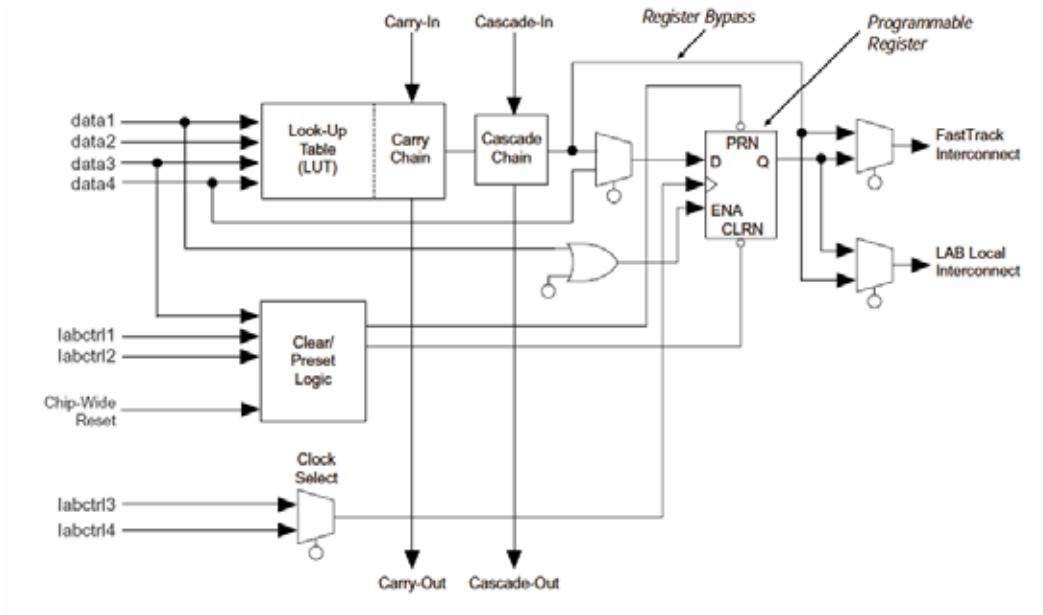
There are, however, significant issues to be taken care of in implementation of hardware support for distributed configuration. The hardware has to be replicated for each logic cluster and this will lead to a significant area overhead depending upon the structure of the hardware employed. This fact leads to the very important question of optimality determination, i.e. the area of decompression hardware should be amortized over how much of the logic block area in order to achieve significant reductions within a reasonable cost of hardware.

Though, the distributed decompression approach is decoupled from any specific logic block architecture, certain logic blocks that will favor more compression depending upon the applications. The next section discusses some logic block type selection issues affecting the distributed configuration efficiency.

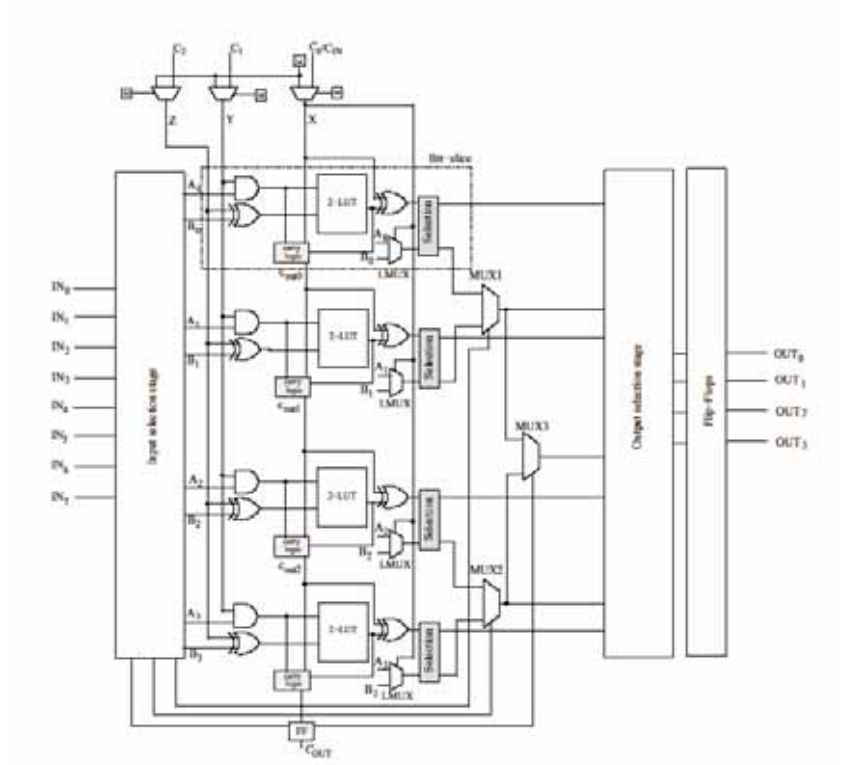
LOGIC BLOCK STRUCTURE ISSUES

Since the application domain being targeted for runtime reconfiguration is datapath oriented, the logic block should be datapath oriented with support for flexible implementation of random logic. If the logic block offers a high functionality with minimum number of configuration bits, it reduces the number of bits required for datapath implementations. Also, the logic block should have well defined configuration patterns for datapath implementation to aid in the quantification of frequently used configurations.

For the sake of the argument presented in this section, a typical low functionality logic block is shown here:



The chosen high functionality logic block is also shown:



The main proposition is that high functionality blocks allow dense datapath implementation using a single logic block compared to multiple blocks for low functionality blocks. It is also true that low functionality block requires less configuration bits/block and vice versa. It is assumed that random logic is implemented equally well in both types of logic blocks. In case of Huffman hardware implementation, the size of the table storing the frequently used configurations depends upon the size of configuration. Thus, this overhead will be lesser for low functionality blocks and vice versa.

The configuration time however will be longer for less functionality blocks or it can be said that datapath oriented designs will require fewer blocks to configure with high functionality blocks and a higher compression while a larger overhead for random logic implementations than low-functionality blocks.

This argument leads to the following conclusions:

- Proper logic block selection for a particular application affects decoder hardware size and the configuration compression ratio
- Since random logic is not compressed, using high functionality blocks for less datapath oriented applications will result in a high decoder overhead that is unutilized and a low compression thus leading to longer configuration streams.

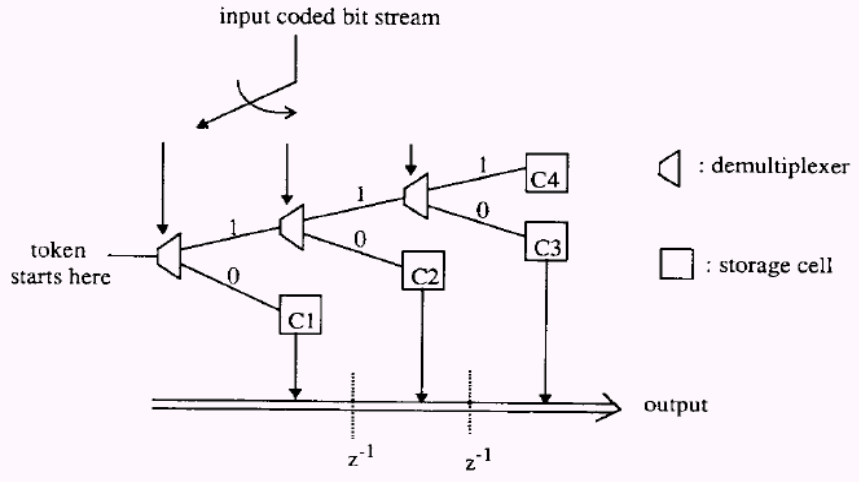
The next section discusses the hardware required for the Huffman decompression and thus an estimate for the area overhead for implementing variable length configurations.

HUFFMAN DECODER HARDWARE

Logically, Huffman decoding can be implemented in different ways for frequently used configurations.

- The configuration pattern itself can be Huffman coded and treated as a symbol, or
- A limited set of frequently used configurations can be stored in a look-up table, which is then addressed to generate commonly used configurations.
- A limited set of frequently used configurations can be stored in a look-up table and the table address bits can be Huffman coded.

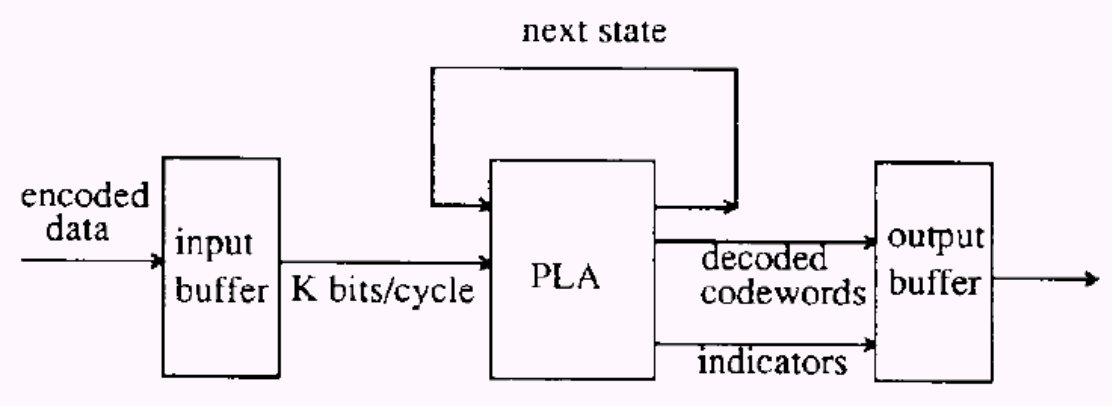
It is understandable from the issue of configuration frequency analysis presented earlier that an adaptive coder (changing codes) will be required to handle different applications efficiently. Which ever approach is used, the basic Huffman hardware is shown below:



This is a direct implementation of the tree based structure, is sequential in nature and has variable input rate and variable output rate. For this reason, sequential decoding is not suitable for WRTR

because a variable input processing rate means that the configuration streams will have to be stalled, which increases arbitration issues and negates some of the benefits of parallel configurations. If the configuration bus is N-bits wide, the hardware should be able to process N-bits at a time. This requires a constant input-rate architecture [VLSI Huffman Designs].

A constant input rate architecture described in [VLSI Huffman Designs] uses a PLA for the table look-ups. The architecture is shown below.



The architecture is low complexity and high speed. The area results for various input/output code sizes are given here.

	input #	output #	product #	minimized product #	throughput
constant input rate					bits/cycle
K=1	9	17	512	235	1
K=2	10	17	1024	406	2
K=3	11	26	2048	612	3
K=4	12	26	4096	901	4
K=5	13	34	8192	1239	5
K=6	14	34	16384	2048	6
K=7	15	43	32768	3130	7
K=8	16	43	65536	4665	8

The area is a function of the constant input rate and the inputs output sizes. The number of inputs and outputs depend upon the symbol sizes and the maximum code length. For instance, if the given number of frequently used configurations is fixed to 15 (+1 for random logic), it will require 4 bits for the table addresses. The Huffman codes for the table address will range from 1 to 8 bits approximately.

Since the architecture is constant-input rate and thus variable output rate, more than one output symbols can be produced in a given cycle. If the smallest code is 1 bit length, then with N=4, the

maximum number of parallel codes will be N per cycle. This parallel decoding has to be consumed by the lookup-table's address decoder and as such to take account of it, multiple configuration chains have to employed. Also, the table in this case has to be a multi-ported memory structure, which can output a maximum of N configurations at a time. A counter can be used to cycle between the chains and keep track of which chain has to be fed next.

For parallel decoding of N codes and M-bit decoded configurations, the decoder hardware will essentially consist of

- Huffman Decoder (discussed before)
- N M-bit decoders
- N port ROM table

The next section discusses the issues in determining the cost-benefit metrics for using the variable length configuration mechanism.

PRACTICAL ISSUES

As it can be inferred from the hardware structure described in previous sections, the hardware overhead for Huffman decoding is not trivial for the case of handling N-bits at a time. The benefit of adding such hardware on the computation speed-up or specifically, reconfiguration speed-up has to first be determined through benchmark implementations. The claims can only then be verified by working on empirical data. The second phase is then to determine the granularity of the logic cluster (the size of cluster served by a single decoder hardware that amortizes the area overhead.) Unless this is done, the suggestions presented cannot be treated as claims.

It may still be feasible to achieve a limited amount of compression however, by using only the look-up table addresses as codes for the most frequently used configurations. Although this scheme requires no Huffman decoding circuitry, it is possible that this may serve as a cost effective means of compressing the configuration.

CONCLUSION

In this project, we have studied the potential for adapting the Wormhole RTR concept for FPGAs, as well as explored for the first time the potential for utilizing highly efficient variable length encoding schemes, as well as Distributed Decompression at each basic block.

We have proposed several Wormhole RTR FPGA architectures, and defined their basic area models. We have also provided an in depth treatment of several system-level issues, as well as considering several options for implementing fast, cost effective distributed decompression schemes.

Based on our work so far, we are able to compare the relative area overheads of the WRTR FPGA models with an equivalent generalized PRTR FPGA model. However, as FPGA design is a highly empirical field, only a proper and thorough experimental analysis of the concepts herein will provide any conclusive result as to the potential of these concepts, or even their viability.

We conclude by describing a general project methodology that needs to be followed for the proper conclusion of this work. The checked items indicate work that has already been done.

- ✓ In depth study of issues.
- ✓ Definition of basic Architecture models.
- ✓ Definition of Area models (for estimation of relative overhead w.r.t other RTR schemes).
- ≈ Design of Reconfigurable systems around designed Architecture models.
- Identification of FPGA resource allocation methods
- For distributing FPGA area between multiple applications/hosts (at runtime, compile-time, or system design-time).
- Selection of Benchmarks for testing and simulation of various approaches.
- Experimentation with WRTR systems and corresponding PRTR system without distributed configuration (i.e. single host/application) for comparison of baseline performance.
- Evaluate resource utilization, and normalized reconfiguration overhead etc.
- Experimentation with all systems with distributed configurations (i.e. multiple hosts/applications). The PRTR systems' configuration port will be time multiplexed between the various applications.
- Evaluate resource utilization, and normalized reconfiguration overhead etc.

APPENDIX: BIBLIOGRAPHY

[1] Pleas refer to Literature survey report on Fast-context switching by Ali Mustafa.

[2] John Reid Hauser, "Augmenting a Microprocessor with Reconfigurable Hardware", PhD Dissertation, Department of Computer Science, University of California at Berkeley, 2000.

[3] Scott Hauck et al, "Configuration Compression for the Xilinx XC6200 FPGA", IEEE Transactions on CAD of ICs and Systems, Vol 18, No. 8, August 1999

[4] Scott Hauck, William Wilson, "Runlength Compression Techniques For Fpga Configurations", IEEE Symposium on FPGAs for Custom Computing Machines, 1999.

[5] D. Deshpande, A. K. Somani, A. Tyagi, "Configuration Caching Vs Data Caching for Striped FPGAs", ACM/SIGDA International Symposium on FPGAs, pp. 206-214, 1999.

[6] Zhiyuan Li, Katherine Compton, Scott Hauck, "Configuration Caching Techniques for FPGA", IEEE Symposium on FPGAs for Custom Computing Machines, 2000.

[7] Ray Bittner, Peter Athanas, "Wormhole Run-time Reconfiguration", Virginia Polytechnic Institute and State University, ACM 1997.

[8] Ray Bittner, "Wormhole Runtime Reconfiguration PhD Thesis", Virginia Polytechnic Institute and State University, 1997