CS 583X Adaptive Computing Systems		Fall Semester 1998
	Lectures 23 & 24: November 10^{th} and 12^{th}	
Instructors: Arun Somani, Akhilesh Tyagi		*Scribe*: Anshuman S. Nadkarni

1 Introduction

We have already looked into the working of the Reconfigurable Multiple Bus (RMB) Architecture: the motivation behind it, its technical aspects like implementation issues etc. In these two lectures, we do a comparison of the Reconfigurable Multiple Bus Architecture with the existing networks, after which we study Striped FPGAs. We introduce the concepts behind Striped FPGAs and derive two caching schemes, Configuration and Data Caching.

2 Performance and Advantages of RMB

In this section, we study the performance Advantages that the RMB has over Previous networks. Several networks have been proposed and used in microprocessor design. We use the hypercube, Fat tree, and the mesh to analyze the effectiveness of the RMB. We use the number of links, Cross points and Area for the RMB as performance metrics. The number of communication links required by a particular model indicates the wire length, which in turn determines the delay that may occur when a signal propagates through a wire. The area and number of cross points indicate how much area a particular implementation requires.

We assume that there are N communication processors. A 'k' permutation allows any arbitrary k messages to pass through the RMB concurrently. In case of the Hypercube, we assume a cube of 'n' dimensions.

2.1 The RMB

To support a k-permutation, the RMB requires k buses. Hence the number of links is N * k. Also each output has 3 cross points and there are Nk output ports in all the INCs together. Hence the total cross points are = 3Nk. Hence the RMB can be laid out in an area of order Nk.

2.2 The Hypercube

The hypercube has NlogN links where N = 2^n, n being the number of dimensions. There are several variations of the hypercube, among them the GFC and the EHC. The degree of each node in an EHC is n+1, as there are 2 links between nodes in any one direction. Hence the no of links are NlogN + N. If the value of k is approximately logN, this is comparable to the RMB. As for a GFC, the total links required is less than (N/k) log (N/k), which is much less than that required for RMB. But the no of cross points are N(logN + 1)^2 and the area required for this is of the order of N^2. Hence this makes it unattractive for VLSI implementations.

2.3 The FAT tree

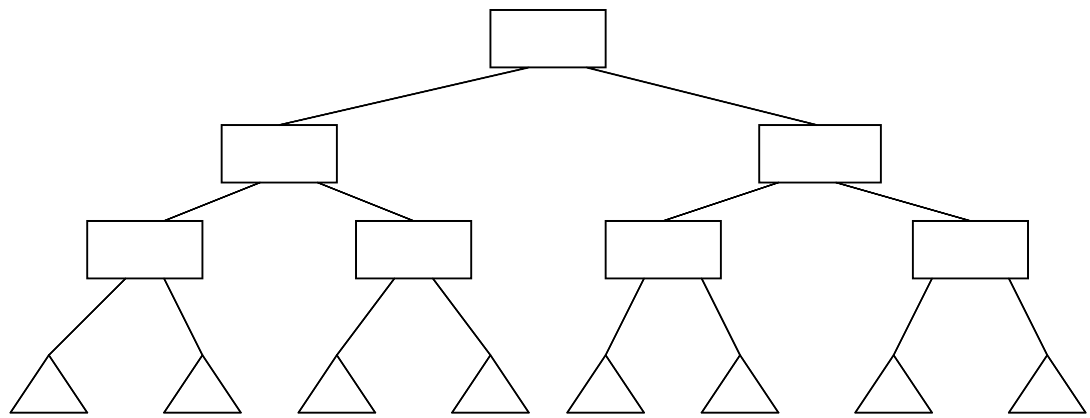

Fig 1
Fat tree

The FAT tree has a structure as shown in the figure (to support a k permutation among N processors) The system shown has N nodes, divided into N/k leaf nodes. Each leaf node has k PEs to connect to and can send upto k messages to the rest of the network. The total number of links in the structure can be found to be Nlogk + N –2k. So for k=N, the number of links is of the same order as the Hypercube supporting full permutation. The RMB has relatively more links.

The link lengths in a FAT tree depend on the layout. It could be of the order of sqrt(N) or sqrt(NlogN). The total wire, in any case turns out to be more than the wire length in the RMB.

The number of cross points in the Fat tree is seen to be of the order of Nk, where the constant is more than 6. The number of cross points in the RMB is favorable in comparison.

As for the Area, it can be shown to be of the order of Nk, with a constant of at least 12. Thus the Area for RMB is less than this.

2.4 The Mesh Network

The mesh architecture has 2N links, each node having a 4x4 crossbar. So the total number of cross points is 4x4N. The mesh can be laid in are of order N. But to embed a k-permutation, the total area of the mesh becomes O(Nk). The routing to establish an arbitrary permutation is not well understood. An RMB on the other hand offers very simple routing.

3 Striped Architectures

The striped FPGA provides hardware visualization by supporting fast run-time reconfiguration. Before studying Striped Architectures, we first take a look at the precursors to this.

3.1 FPGA Configuration Mechanisms

The FPGA configuration Mechanisms are as follows:

3.1.1 Standard FPGA Configuration

Popular FPGA's like the Xilinx 4000 series and Altera Flex family come under this category. They have exclusive configuration modes. There is no mechanism for allowing simultaneous operation and configuration or even partial modification of a configuration that is already loaded. The configuration time is large compared to the operating cycle time of a design. It is used for fast prototyping.

3.1.2 Online Configuration

The Xilinx 6200 family come under this category. They can be configured 'online', that is, the configuration storage of the FPGA is accessible and writable during execution. Reconfiguration can take place at run-time and partial reconfiguration is possible. When used to implement pipelined applications they require significant data storage and routing resources.
They have short duration for programming / reconfiguring.

3.1.3 Multiple-context FPGA's

This is similar to the standard FPGA except that instead of having one configuration stored in the FPGA, there are n configurations stored in the FPGA. A global selection mechanism selects determines which of the n configurations are to be used in the current cycle. This increses the reconfiguration speed. But this approach has its limitations, as it requires a lot of memory for storage of the configurations. Hence the size of the FPGA is large. Also the reconfiguration beyond n contexts must take place on slow, narrow configuration busses.

3.1.4 Striped Configuration

This combines many of the advantages of on-line configuration and multiple-context configuration, and suited for pipelined applications. A pipelined application can readily be decomposed into pipelined stages. The striped FPGA is separated into a set of 'stripes' such that ideally, each of the applications stages would fit in each stripe.

A striped FPGA can be configured in one cycle time, through an on-chip configuration cache. The unused configuration information can be stored in a large on-chip RAM. Modification of the cache can tae place concurrently with execution.

Another characteristic of striped FPGA is that the configuration info can move from within one stripe to other within the FPGA.

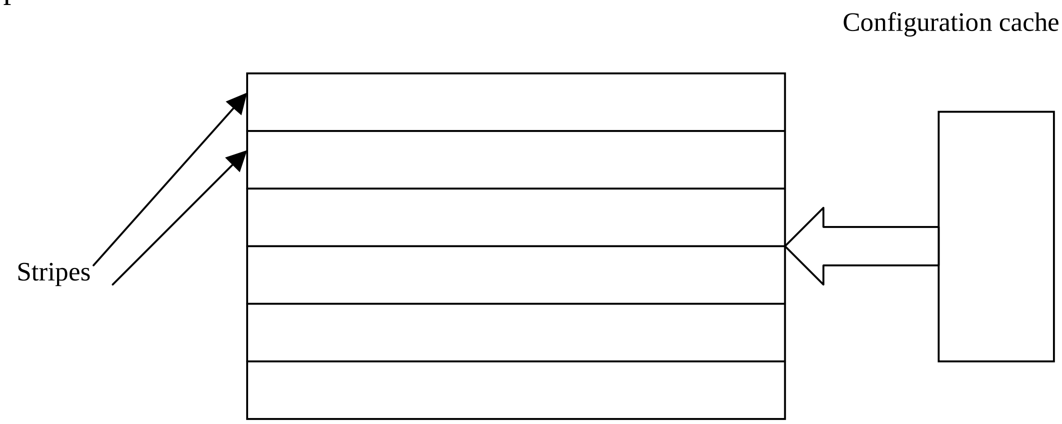

Fig 2
Stripe Configured FPGA

In multi-context, for each FPGA, we have memory blocks. When we distribute this memory, we are fragmenting. It is better to have contiguous memory rather than fragmented.

3.2 Motivation for Striped FPGAs

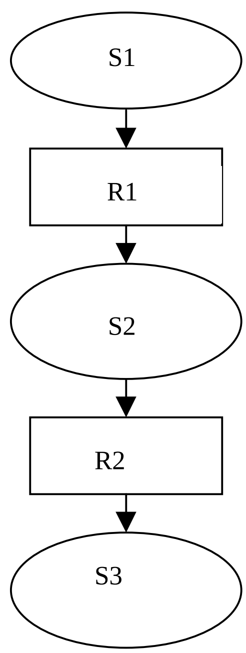

Fig 3

Consider a deeply pipelined application consisting of several stages, S1, S2 .. Sn. If we wish to map this application into hardware, we could use FPGAs to realize the same. But we may have to devote a large number of FPGAs for this purpose, as the number of stages, s, is large. If all the stages are mapped into one, then we may have a solution as indicated in the following figure:

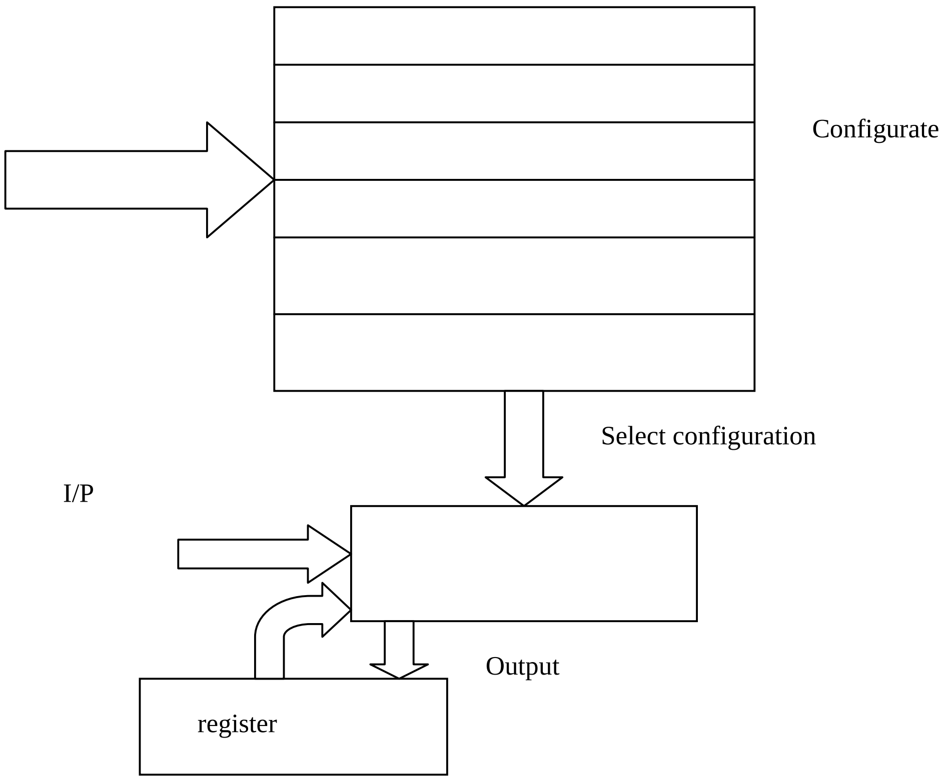

Fig 4

But this design has the following drawbacks due to which it cannot be realized in hardware:

1) if 1 data element is to be processed at a time, then the intermediate result indicated by register could be of any size. Most of the registers will be larger than the input. Depending on the application the sizes will be different.

2) A bigger problem would be that pipelining will not be possible, as we cannot process data elements simultaneously.

3.3 Solution: Configuration Caching

When we don’t have sufficient FPGA's to do the job, we reconfigure the stages. For instance if we have an FPGA that can accommodate only 3 stages at a time, but we have 6 (s=6) stages, then we can reconfigure s1 to behaves as s4, s2 as s5, and s3 as s6.

Hence,

s1-> s4
s2-> s5
s3-> s6

Therefore, its not fully pipelined, we stop at 3.
Assumption:
Stages S1 through S6 act on data elements D1 through D6. Only 3 hardware stages are available.
HS1, HS2, HS3 are the hardware stages.
Rec(Si) Indicates that the stage Si is being reconfigured.
Si(Di) Indicates that data element Di is being acted upon by Stage Si.
We also assume that the amount of time for reconfiguration = the stage time.

	HS1	HS2	HS3
1	Rec(S1)	-	-
2	S1(D1)	Rec(S2)	-
3	S1(D2)	S2(D1)	Rec(S3)
4	Rec(S4)	S2(D2)	S3(D1)
5	S4(D1)	Rec(S5)	S3(D2)
	…	…	…

Clock cycles

Fig 5

Given that we have n stages, we use (n-1) stage steps and 1 stage for reconfiguration
So n-1 data elements can be processed at a time.

But there is always 1 bubble in the pipeline.

Let
x ->data elements have
n-> no of stages need.
s-> no of stages have

1st data element will go fine. The 2nd and 3rd onwards (n-1) data elements will be followed by a bubble.

s+x-1 steps will be required in the traditional pipeline. Therefore to compute the x data elements, the total time required (ignore fillup and cleaning up time)
n branch Instructions, branch penalty = 2 cycles.
processos cycles to compute the listy with no other breaks:

$s+x-1+n*2$

	HS1	HS2	HS3
6	S4(D2)	S5(D1)	Rec(S6)
7	-	S5(D1)	S6(D2)
8	-	-	S6(D2)

Fig 6

s.x -> space time units available.
We also need time units to fillup, clean, reconfigure.

Therefore, time:
s.x + fillup time + emptying time + reconfiguration step.

Basically there is one bubble continuously in the pipeline.

3.4 Data Caching

This method may be used if we want to avoid wasting the cycle for reconfiguration. In this, we reconfigure only after all the elements have been through the stage. But for this approach, we will need to store the results that come out of the third hardware stage. Which again brings us to the problem of unknown storage size. We may have to allocate large amount of buffer space.

	HS1	HS2	HS3
1	Rec(S1)	-	-
2	S1(D1)	Rec(S2)	-
3	S1(D2)	S2(D1)	Rec(S3)
4	S1(D3)	S2(D2)	S3(D1)
5	S1(D4)	S2(D2)	S3(D2)
	…	…	…

Fig 6

Hence at cycle no N+x we will get the last data. Then we start reconfiguring again.

Due to these disadvantages we try to use a hybrid method that includes the advantages of both the schemes.

References:

1. Incremental Reconfiguration for Pipelined Applications, Herman Schmit, CMU.
2. Configuration Caching v/s Data Caching, Deepali Deshpande, Arun Somani, Akhilesh Tyagi, ISU.