PipeRench
February 4, 1998

- Lab 2
- PipeRench Motivation
- Class of RTR Architectures
- PipeRench
- Assembler
- Dataflow Intermediate Compiler

Lab2

- Implement a kernel on PipeRench
- Kernels
  - Floating point add/subtract
  - Floating point multiply
  - Systolic Priority Queue
  - IDEA encryption
  - Nqueens
  - Huffman Encoding
  - Integer Square Root
  - DNA Matching
  - 64x64 Multiply on as 10PE wide stripe
  - Matrix-Vector Multiply

Reconfigurable Computing Today

- No forward-compatibility
  - FPGA design effort is large
  - Performance speedup eroded by Moore's Law
  - Performance only with redesign
- Unforgiving constraints
  - "Need-one-more-cell" failures
  - Compilers need forgiving media
  - Users want applications to run on any platform
- Oriented toward bit-operations and gates

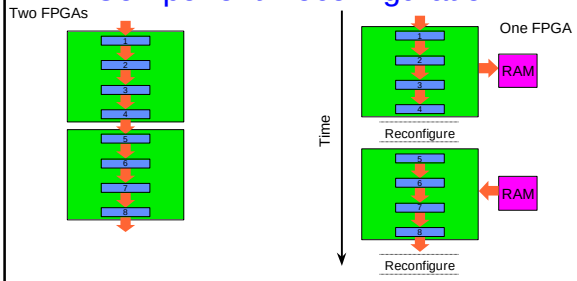
PipeRench Goals

- Virtualization of hardware designs
  - Hardware: ∞
  - Design for maximum parallelism
  - Time-multiplex on real hardware
  - Provide:
    - Forward compatibility
    - Forgiving constraints
    - Datapaths
- Target: Pipelined Applications
- Incremental Reconfiguration

Virtualization

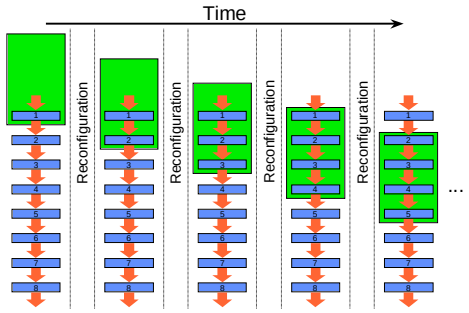
- What do we do if there's not enough hardware for the entire application?
- Break into pieces and time-multiplex
- Like a virtual memory:
  - Keep active portions in FPGA
  - Move inactive pieces out

Component Reconfiguration



- FPGA size determines unit of reconfiguration
- Need memory for intermediate results
- Reconfiguration time impacts performance

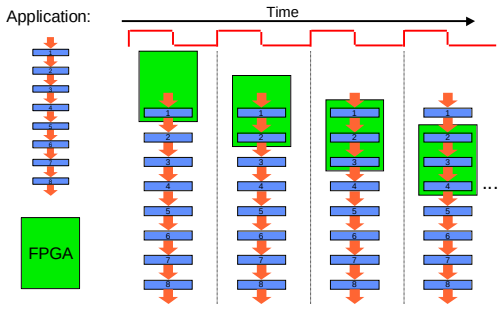
Pipeline Stage Reconfiguration



Pipeline Stage Reconfiguration

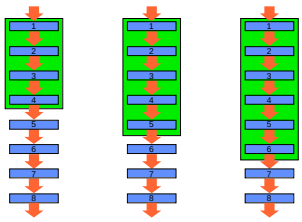
- Unit of reconfiguration “application-oriented”
- No storage needed for intermediates
- Can reconfigure while executing:
  Reconfiguration time of zero!
- More fabric ⇒ higher throughput.
  (until physical size = virtual size)

Pipelined Reconfiguration



Hardware Virtualization

- More Hardware = More Throughput



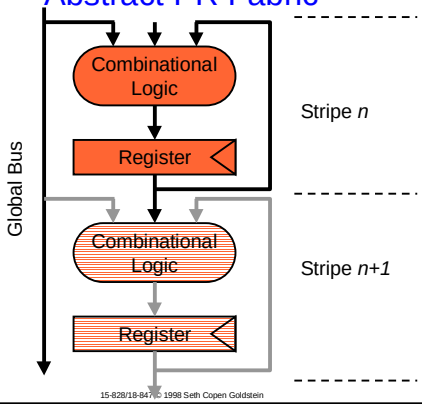
Benefits

- Forward Compatability
  - Design effort pays off for longer
  - Can target the 10x performance wins
    not just 100x and 1000x
- Graceful degradation
- Supports high-level Programing
- Hides Configuration Time
  Can target smaller apps with smaller data sets

Challenges

- Ultra-high speed reconfiguration
- Run-time scheduling of virtual hardware
  - Configuration Control
  - Memory Access Control

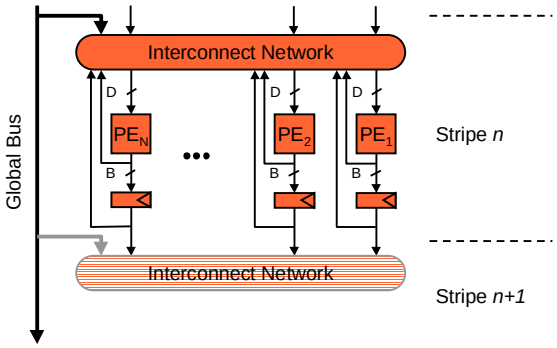
Abstract PR Fabric



Making It More Concrete

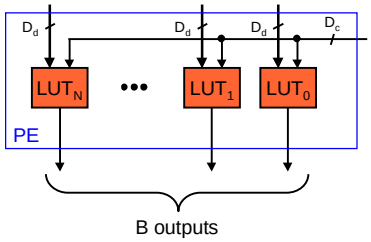
- Choosing method of implementing combinational logic: LUTs.
- One big LUT is too large, ∴ N PEs
- Must now have interconnect.
- Target is "word" oriented applications and want to keep size of config bits small
  ∴ Each PE is B D-LUTs

PE Based PR Fabric

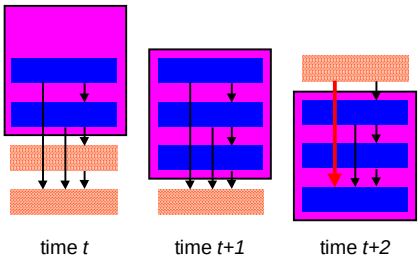


The PE

- A single PE must be able to act in concert
  ∴ $D=D_d+D_c$ inputs per LUT
- Each PE has BD_d+D_c inputs and B outputs

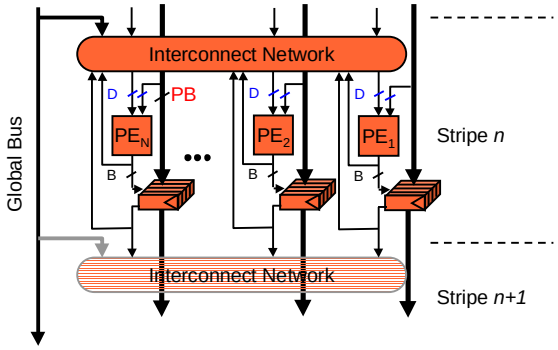


Virtualizing the Wires

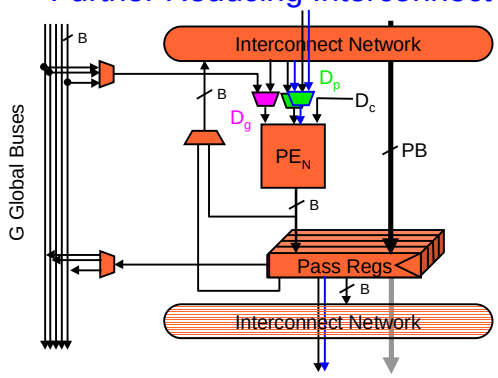


Why not just use PEs?

Adding Pass Registers



Further Reducing Interconnect



What is left to specify?

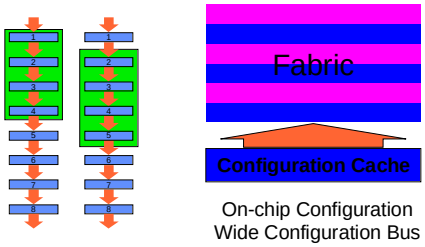
- Interconnect
- How control lines are routed & implemented
- Internal's of a PE (aside from LUTs)
- # of pass registers to Interconnect

PipeRench

- PipeRench is an instance of the pipelined reconfigurable class
- N 32 Number of PEs per stripe
- B 4 Width of a PE
- G 4 Number of global buses
- P 4 Number of pass registers
- D_g 1 Global bus inputs to PE
- D_p 1 Pass register inputs to PE
- D_c 1 Control inputs to PE

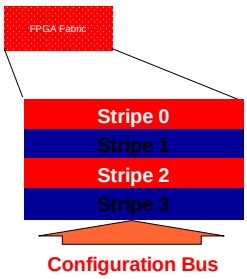
Pipeline Reconfiguration

- Objective: Configure 1 stage per cycle



On-chip Configuration
Wide Configuration Bus

Fabric Architecture

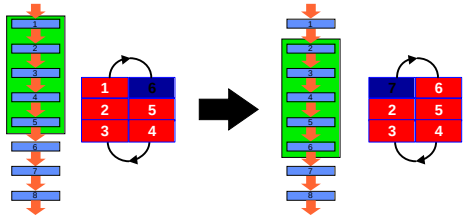


The Stripe:

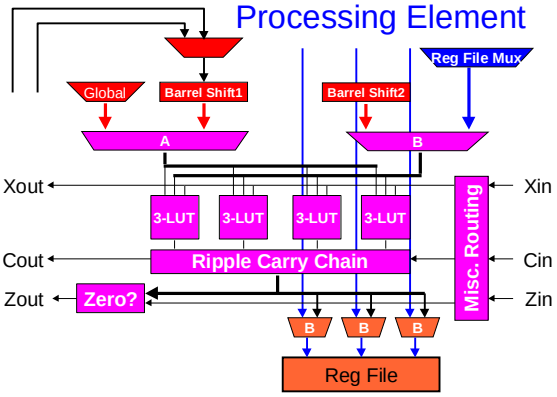
- Configured in one cycle
- Concurrent configuration & execution
- Local, registered interconnect between adjacent stripes
- Architected to implement one pipeline stage

Stripe Configuration

- Configuation doesn't really move!
  - Ring: local interconnect, stationary configuration

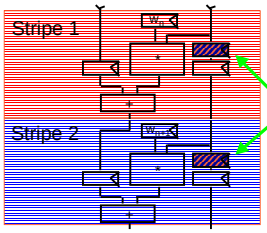


Processing Element



Save and Restore

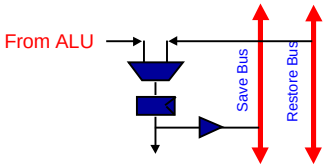
- Virtualizing “uneven” pipelines



Registers Store State
Value must be:
Saved when removed
Restore when replaced

Save and Restore

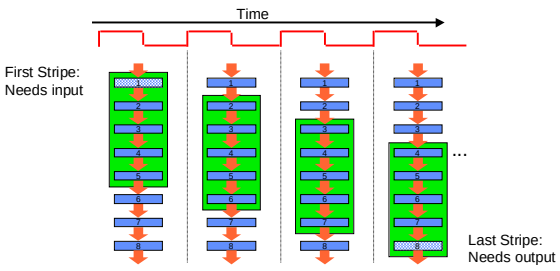
- When stripe is over-written:
  - Store its content
  - Restore it later
  - Necessary if stripes contain “state”
- Global busses to save and restore



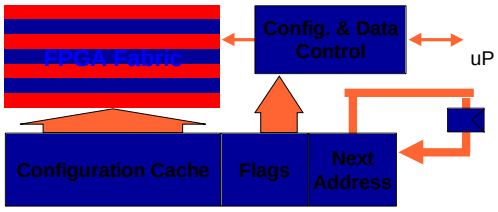
Configuration Controller

- Primary Function:
  - Map virtual pipeline to given hardware
    - Run-time
  - Configuration mapping:
    - Is time-multiplexing necessary?
    - From where? (in the memory)
    - To where? (in the fabric)

Memory Access Control

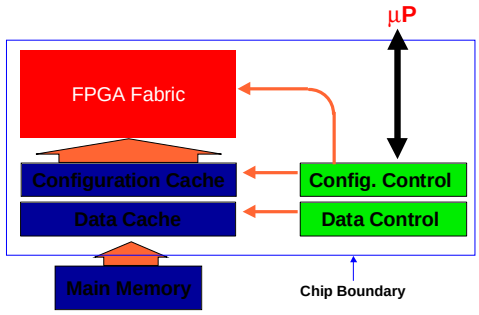


Implementation



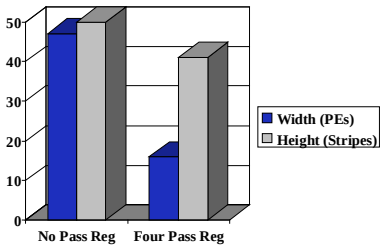
Flags:
Write bus (4)
Read bus (4)
Store/Restore
End

Architectural Summary



Pass Registers: DCT

- Reduction of PEs by 3x:
  - Translated to more parallelism



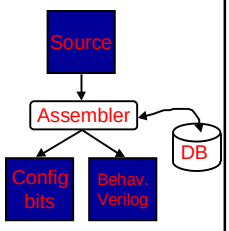
Assembly Language

- Implicit Pipelining
- Interconnect oriented
- Re-targettable to:
  - Variations of our architecture
  - Other architectures using pipeline reconfig.
- Generates
  - Behavioral Verilog for CVH simulator
  - Bit-streams for CVH

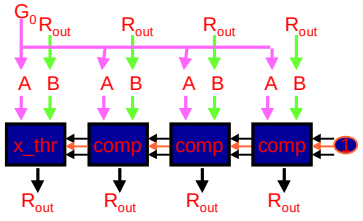
Assembly Language Features

- Parameterized
  - Width of stripe
  - Number of registers
  - Type of interconnect
- Promotes Abstraction
  - Can name PEs
  - Can reuse stripes
  - Can define operations
- Simple Interconnect Specification

A step above circuit design.



Simple Example




```
function comp low;
    0,3
end function;

stripe comparitor;
    pe.{2..0} = comp;
    pe.3 = x_through;

    {3..1}.zin = {2..0}.zout;
    0.zin = 1;
    A = Global.0;
    B = prev.Rout;
end stripe;
```


Using Symbolic Names

```
def in {2..0};
def result 3;

stripe comparitor;
       lut.in = comp;
       lut.result = x_through;

       result+in:~0.zin = result+in:~m.zout;
       in:0.zin = 1;
       A = Global.0;
       B = Global.1;
end stripe;
```