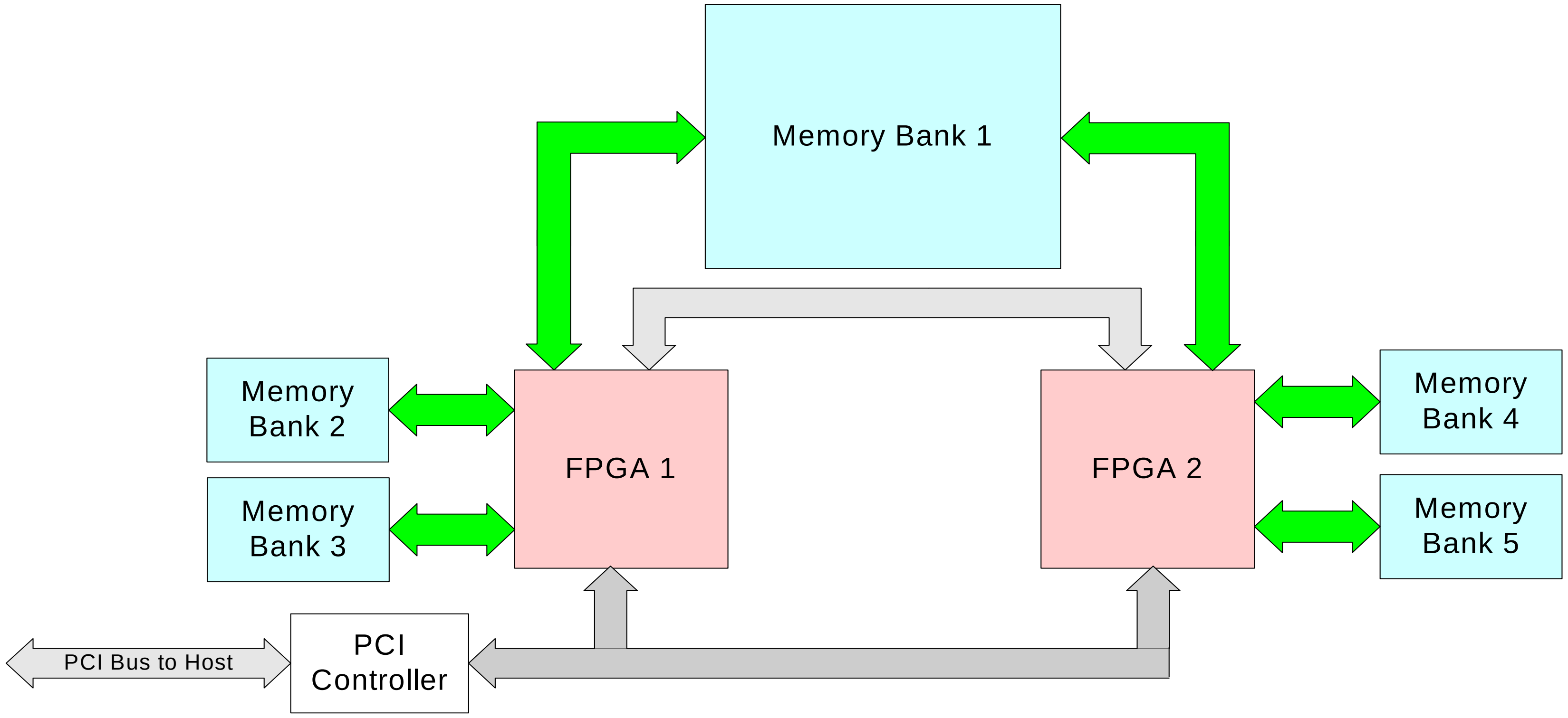
Leeser’s Lessons *or* How to Get Performance Out of Your FPGA Designs and How NOT To

Prof. Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University
Boston MA
mel@coe.neu.edu

Northeastern UNIVERSITY

MAPLD 2008

FPGA(Hardware) Platform



Partitioning

- What tasks go on the FPGA and what tasks remain in software on the processor?
- Profile your application.
  - Know which are the compute intensive kernels
- How NOT to get speedup:
  - Focus on the algorithm first. Worry about having the kernels to implement your design on an FPGA.
  - Ignore data transfer, etc.
  - Ignore particulars of the target platform.
    - Memory sizes
    - Bandwidth

Know Your Potential Speedup

- Assume the portion that will go on the FPGA will take zero time
  - and all associated costs (data transfer, etc.) are zero
- This gives you the maximum possible speedup you can achieve

Know the Characteristics of Your Hardware Platform

- Ignore the algorithm
- Focus on data movement
- Know the target platform and its performance characteristics
  - What is the data transfer speed and bandwidth between the processor and the FPGA?
    - In modern FPGAs, this can vary widely
      - PCIe, PCI-X, Hypertransport, Frontside Bus
  - How much memory is addressable from the FPGA?
    - On-chip memory is limited

It’s the Data Movement, Stupid

- Pay attention to data movement at all levels:
  - Between processor and RAM local to the FPGA
  - Between RAM and FPGA chip
- View memory as a hierarchy
  - Addressable by the PC
  - Addressable by the FPGA: On board
  - On the FPGA: On chip

Assess Costs *Not* in Initial Code

- Assume the kernel that runs on the FPGA takes *zero* time to process the incoming data
- Analyze the amount of time needed to transfer data to and from the FPGA board for your design
- The time to transfer data should not exceed the processing time of the section of code you are accelerating
- If you can overlap data transfer and processing, or stream data to the FPGA, you can tolerate more time to transfer data

FPGA Performance Limits

- Goal: Use all your resources all the time
- Get the maximum amount of data into and out of the FPGA in every clock cycle
- The size of problem you can handle may be determined by the amount of data you can fit in on-board RAM
- The rate that you can bring data into the FPGA determines the clock rate. Do as much work as you can in one clock cycle.

Speed of FPGA Design

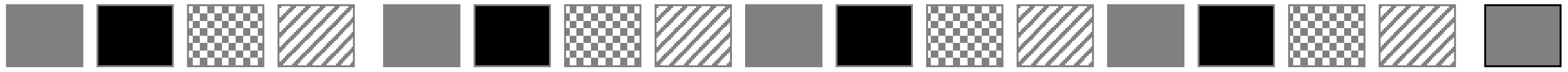
- Should you have different clock domains?
  - One for the memory interface
  - One for the processing logic
- Yes, if it makes sense for the hardware platform you are on, but:
  - Crossing clock domains is hard
  - Usually put FIFOs in between
    - Are you really getting more work done?
- More work at a slower clock rate results in lower power dissipation
  - Faster isn't always better

Pipelining and Parallelism

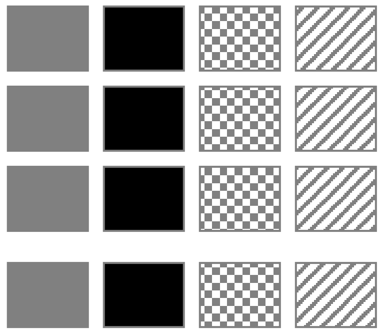
- The main ways you achieve speedup in an FPGA design
- Parallelism:
  - Replicating the same operation multiple times
  - Operate on different data at the same time
  - Uses area
- Pipelining
  - Overlap the sub-operations of different operations
  - Requires no extra hardware
    - except for control

Sequential vs. Parallel Hardware

- Sequential



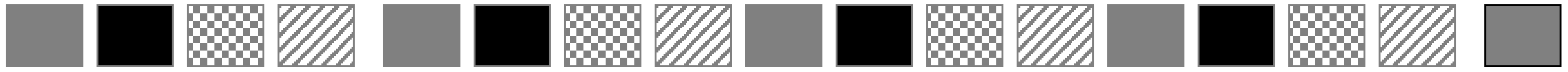
- Parallel



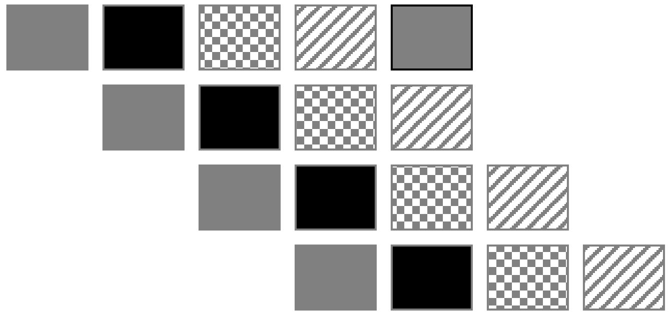
Parallelism requires more hardware

Sequential vs. Pipelined Execution

- Sequential



- Pipelined

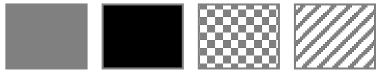


Pattern: task; horizontal axis: time

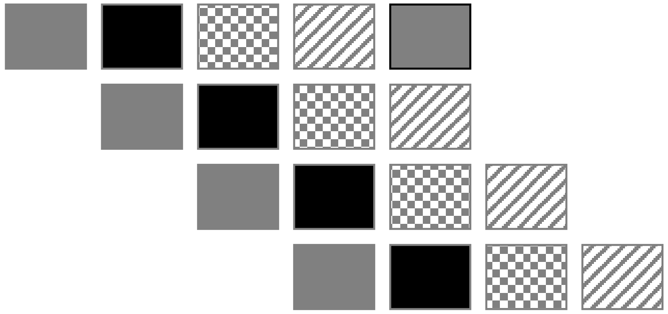
Showing schedule, not hardware components

Pipelined Hardware

- Sequential and Pipelined have same hardware



- Pipelined schedule, not hardware units



Pipeline Your Design

- You should be able to:
  - – Read inputs every clock cycle
  - – Produce results every clock cycle
- Definitions:
  - – Latency: The number of stages in the pipeline
    - Time it takes to generate one result
  - – Throughput: The time from one result to the next result
- For most applications
  - – The amount of latency does not matter
  - – The throughput matters

Use All Available Bandwidth

- Provide as many pipelines in parallel as your bandwidth to the FPGA will support
  - Combine pipelining and parallelism
- Avoid stalls in your pipeline
  - All data goes through exact same steps
- Many applications are bandwidth limited
- Try to fetch data from memory only once
  - on-chip blockRAMs are a “managed” cache
  - on-board memory is the main memory

Optimize Arithmetic

- Optimize the format of your arithmetic operands to your data/problem.
- You are not constrained to stick to 8, 16 and 32 bit integers, or standard floating point formats.
- You can choose integers, fixed point or floating point, and any number of bits in any format.
- This flexibility means a lot of work in terms of choosing the correct implementation.

Example: Custom Floating Point

- Inputs and Outputs are in single precision IEEE Floating Point
- Design consists of a large number of additions
- Do NOT normalize after each addition:
  - This requires an additional adder
- Instead, add a bit to the mantissa for each sum
- Normalize at the end

Avoid Special Cases

- Want to keep all your hardware running all the time
- Initialization happens only once
  - Do it in software
  - Figure out a way to handle the initial case the same as all other cases:
    - Pad with zeros, ...
- Do not devote hardware to special cases
- Avoid stalls, bubbles in pipelines

Avoid Data Copying

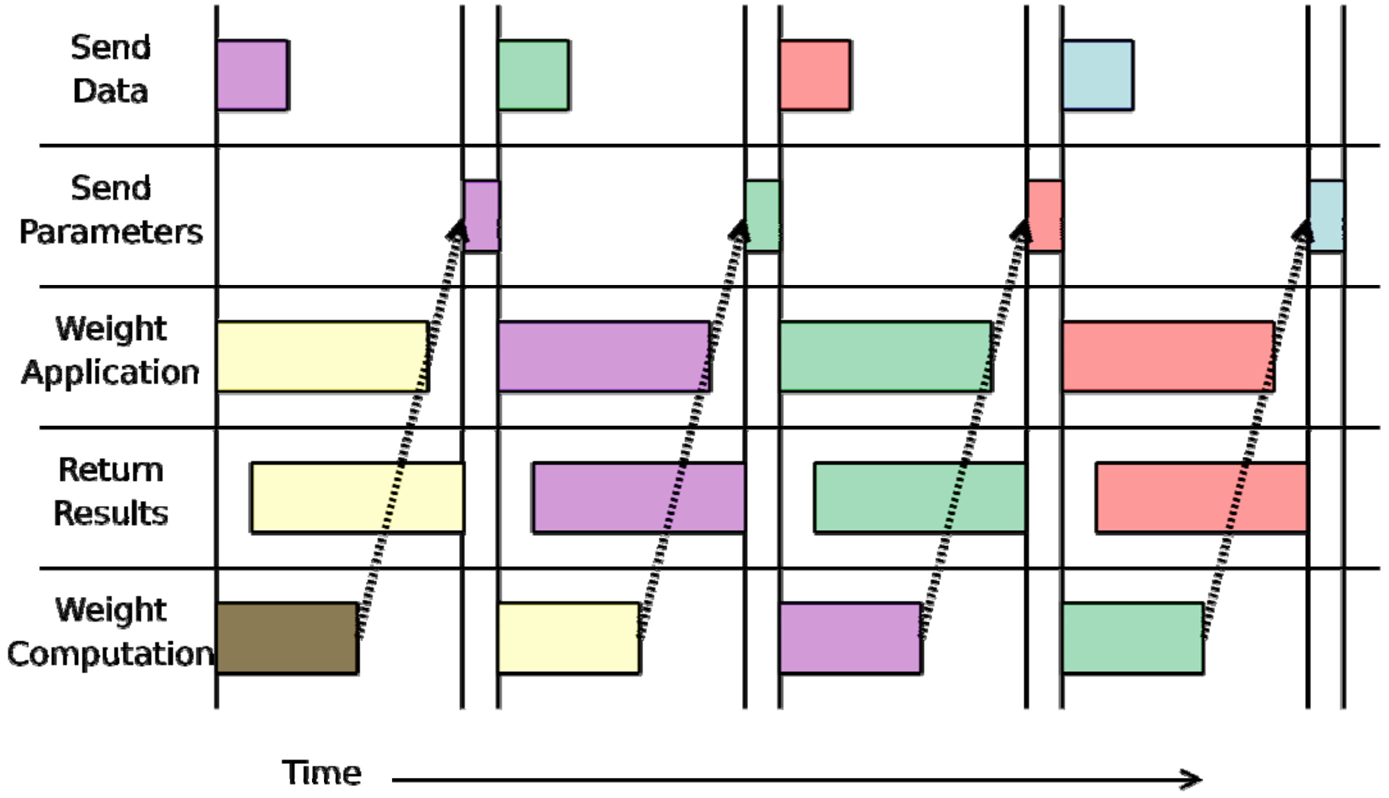
- If your algorithm iterates through the data several times:
  - Use one memory bank for input data and another for output data
  - At the end of each iteration, swap the roles of the input and output memories
- Avoid data copying at every level of the memory hierarchy.
- Overlap computation and data transfer as much as possible at every level of the hierarchy.

Now Implement the Kernel

- Use parameterized libraries where available
  - Xilinx Coregen
  - Altera IP
  - Commercially available libraries
- Avoid FIFOs
- Compilers are fine, once you have designed the outline of your solution
  - Memory Hierarchy
  - Clock Cycle
  - Pipelining

An Efficient FPGA Design

Described in: **Vforce: Aiding the Productivity and Portability in Reconfigurable Supercomputer Applications via Runtime Hardware Binding available from: http://www.ll.mit.edu/HPEC/agendas/proc07/agenda.html**



To Learn More

- Computer. Published by the IEEE, Special Issue on "**High-Performance Reconfigurable Computing"** March, 2007
  - **Achieving High Performance with FPGA-Based Computing** by M. Herbordt, T. VanCourt, et al., pp. 50-57
- Reconfigurable Computing Lab at Northeastern: http://www.ece.neu.edu/groups/rcl/
  - Click on "Publications"
- Thank you! Miriam Leeser
  - mel@coe.neu.edu