Comparing Synthesizable HDL Design and Stream Programming

Jesse G Beu
Thomas M. Conte
Georgia Institute of Technology

Outline

- Origin
- Stream Programming
- Synthesizable HDL ‘programming’
- Design example
- Comparison
- Why should we care?

Origin

- C***omputing Beyond Von Neumann***
  - Micro 2007 Panel discussion
  - Concerns over future of programming
  - Human brain, Quantum, Biology, Microarchitect
  - No ‘hardware’ representative
- What about HDL as a programming model?
  - Hardware designers have been ‘programming’ in parallel for years

Stream Programming

- Data organization followed by computation
  - Gather-operate-scatter type methodology
  - DMA and Kernel Processing
- Extensive use of Synchronous Data Flow (SDF) graphs
- Readily available, visible parallelism
  - Compiler opportunity
  - Easier mapping to hardware resources

Synthesizable HDL ‘programming’

- ASICs are inherently parallel
- Synchronous steps via procedural blocks
  - Sensitivity lists
  - @posedge clock
  - Non-blocking assignment <=
- Block Diagram Organization
- RTL design – SDF analog

```
Always @(posedge clk) begin
   a <= b;
   b <= a;
end
```

Register Transfer Level

- Register to register signal flow
- Signal operator logic
  - Explicit combination expression
  - Standard constructs (if-then-else)
- Synthesizable RTL is strict
  - Enforces a design-before-coding style
  - Heavy block diagram and flowchart use

Design Example – CPU Core

- Datapaths
- Control Units
- Memory
- Clock Circuitry

Design Example – CPU Core

- Notice ALU's size
- Reflects Stream Programming
  - Primary effort in packing and unpacking
  - Small yet powerful kernel
  - Overall effective design

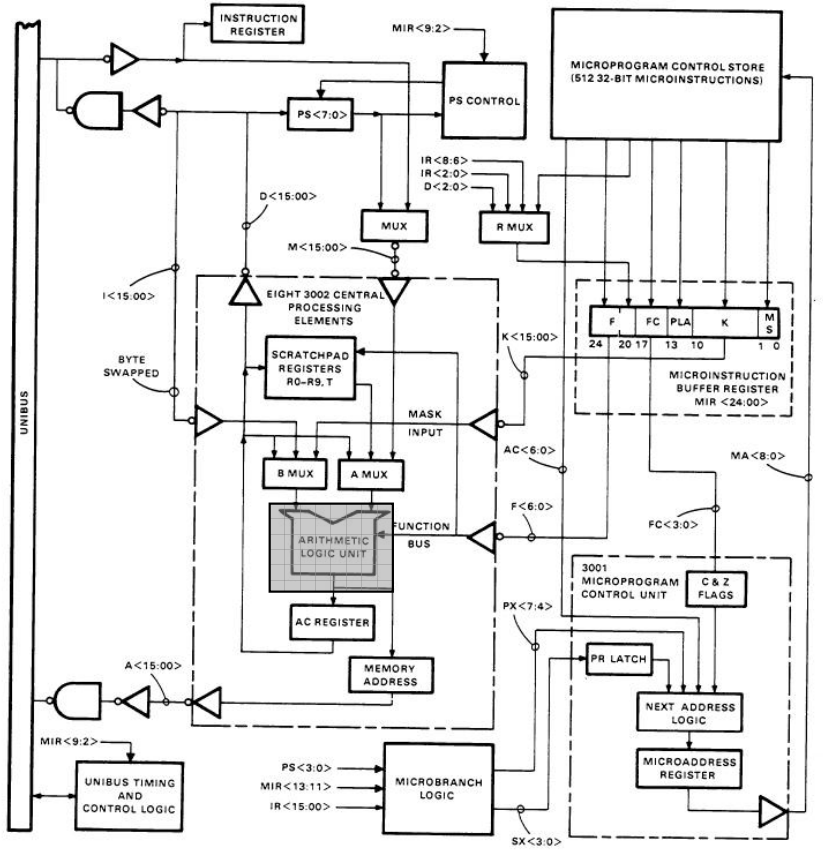

Image for CMU11 RTL borrowed from http://research.microsoft.com/~gbell/computer_engineering/00000472.htm

Streams and HDL Similarities

Stream Programming

- Decouple memory and computation
- Synchronous Data Flow
  - Discrete work units
  - Expresses parallel and pipeline potential
- Stream compiler exploit locality

Synthesizable HDL

- Decouple function units and communication
- Pipelined Block Diagram
  - Discrete function units
  - Expresses parallel and pipeline potential
- Hierarchical nature expresses locality

Contrasting Streams and HDL

Stream Programming

- Dynamic Memory
  - Performance Impact
- Algorithmically complex is acceptable
- Easily manipulated
  - Application evolution
  - Fast turn around
  - Tuning

Synthesizable HDL

- Fixed resources
  - Data structure implications
- World of ‘worst cases’
- Targeting high-volume, long term deployment
- Validation and Reliability consideration

Why should we care?

- Streams and HDL are similar, yet disjoint fields
- Potential for better streaming languages
  - Learn from HDLs widespread acceptance
  - Use as a model for teaching parallel programmers
- Potential for better HDLs
  - Possible application of stream techniques to design?
  - Atomicity anecdote: Rishiyur Nikhil and Bluespec

Thank you

Questions?