Bluespec at MIT

Arvind (arvind@mit.edu)
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

The First Bluespec Workshop, MIT,Cambridge, MA
August 13, 2007

Real power saving implies specialized hardware

- H.264 implementations in software vs hardware
  - the power/energy savings could be 100 to 1000 fold

but our mind set is that hardware design is

- Difficult, risky
  - Increased time-to-ma
- Inflexible, brittle, er
  - How to deal with changing standards, errors

New design flows and tools can change this mind set

SoC Trajectory:

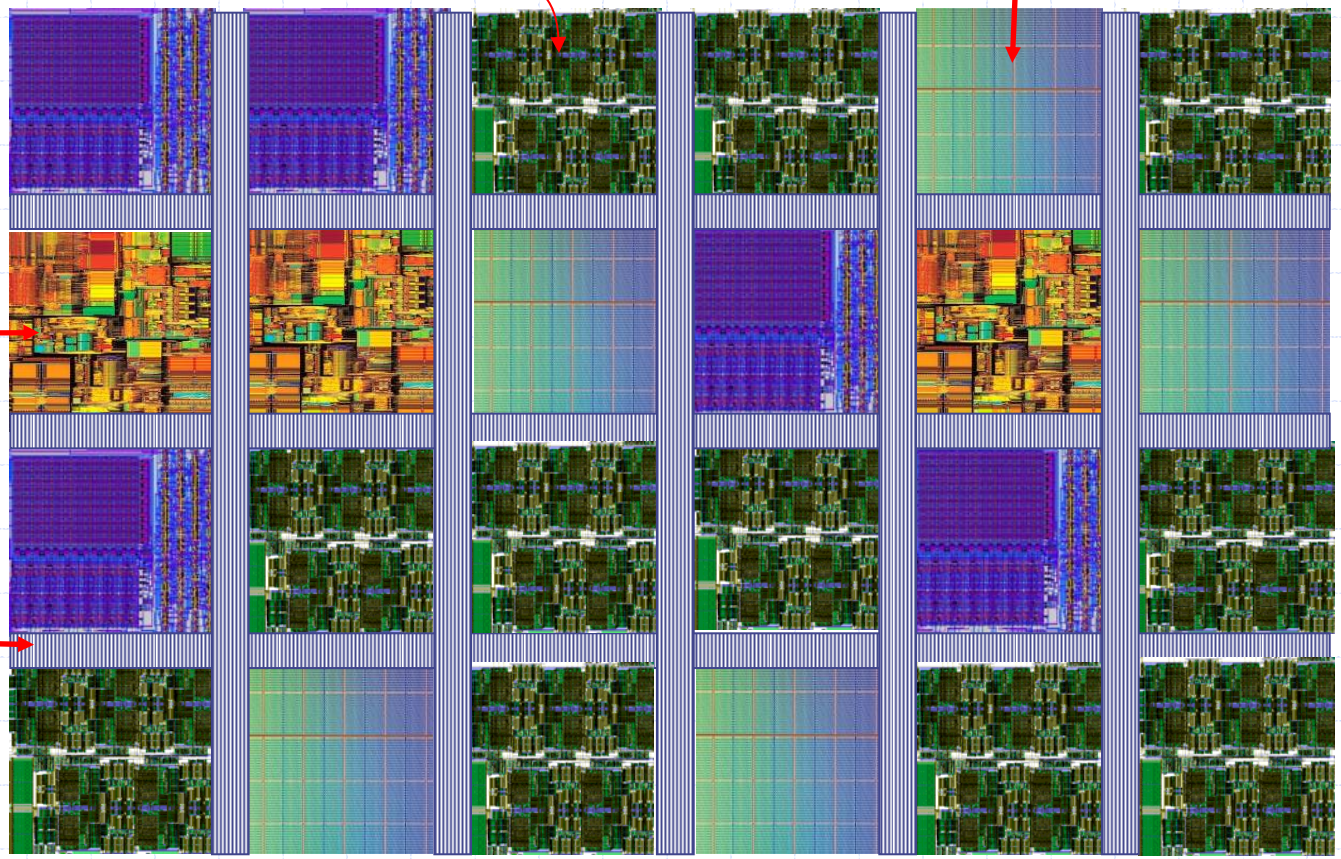
more application specific blocks

On-chip memory banks

Application-specific processing units

General-purpose processors

Structured on-chip networks



Can we rapidly produce high-quality chips and surrounding systems and software?

Making hardware design easier

- Extreme IP reuse
  - Multiple instantiations of a block for different performance and application requirements
  - Packaging of IP so that the blocks can be assembled easily to build a large system (black box model)
  - Whole system simulation to enable concurrent hardware-software development

Bluespec addresses all these issues

Recent Applications

- Multiradio OFDM: From WiFi to WiMax
  - 802.11a and 802.16 from the same source
- H.264 Decoder
  - Baseline profile, 720p X ~75 frames
  - FPGA implementation working

Other examples: Processors, Cache Coherence Protocols, IP Lookup, ...

Negotiations are underway with sponsors to publish all designs done at MIT under the MIT open source license

Importance of Publishing Bluespec Designs

- Enables whole community to undertake much more ambitious projects
  - We already see the effects in 6.375 projects
- Enables derivative designs, specializations and variety at a fraction of the development cost

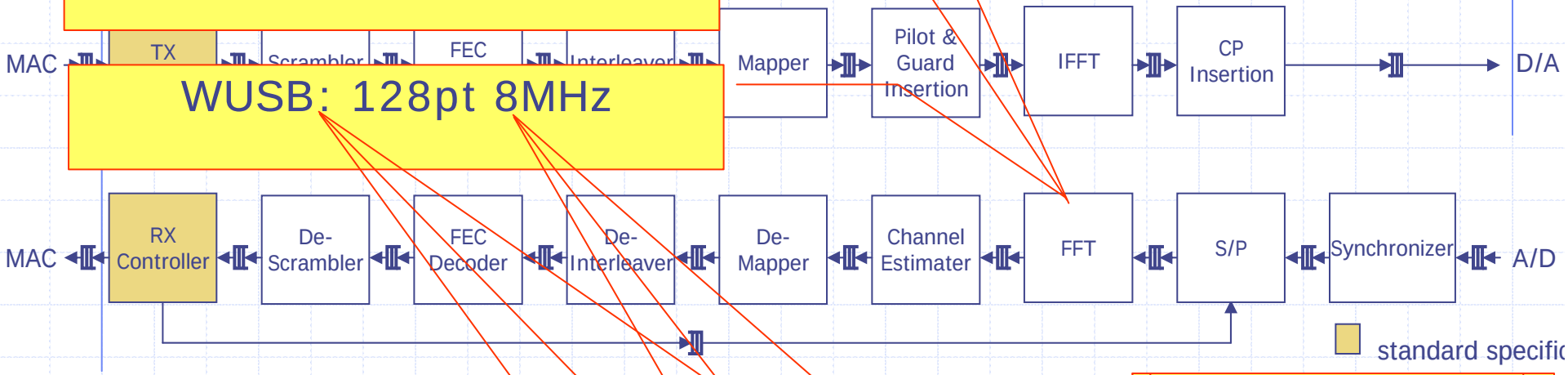
Parameterized modules

Example OFDM based protocols

WiFi: 64pt @ 0.25MHz

WiMAX: 256pt @ 0.03MHz

WUSB: 128pt 8MHz



Convolutional

Reed-Solomon

Turbo

- Reusable algorithm with different parameter settings
- [illegible]
- Different algorithms

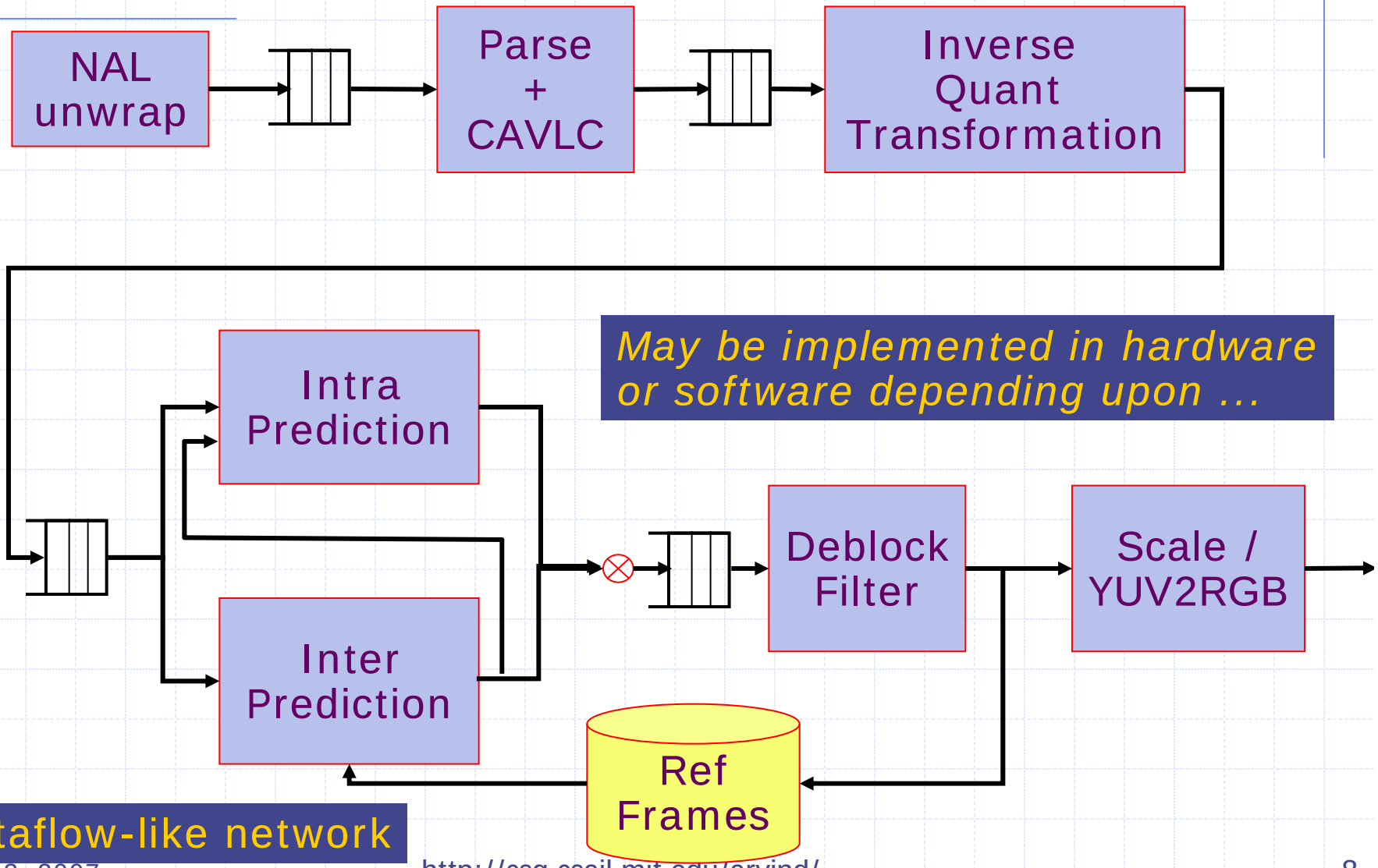
85% reusable code between WiFi and WiMAX

From WiFi to WiMAX in 4 wee

(Alfred) Man Chuek Ng, ...

H.264 Decoder

K Elliott Fleming, Chun-Chieh Lin, ...



May be implemented in hardware or software depending upon ...

A dataflow-like network

H.264 Learnings

- Productivity: Base profile
  - Effort: Less than one-man year
  - 8K lines of Bluespec (contrast 20k to 80K lines of C)
  - First draft decoded 720p @ ~32fps, (Available C codes do not meet this performance)
- Architectural Exploration: Many improvements made over a period of sevral months to increase performance and reduce area
  - Process several samples / cycle
  - Adjust FIFO depths
  - Pipeline modules: Interpolator, Deblocking filter
  - After improvements decodes 720p @ ~95fps (180nm)

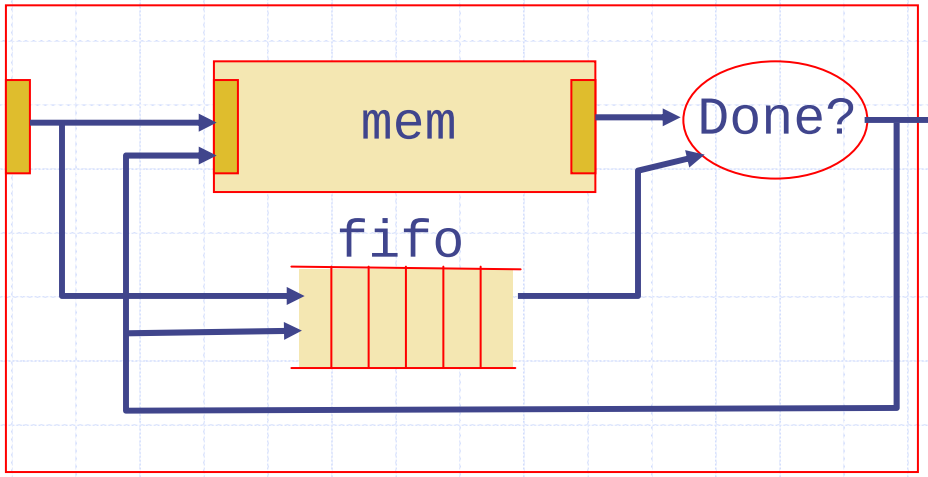
Modular refinement is both feasible and essential

Current research

- Make the path to hardware design easier
  - FPGA emulation infrastructure
  - Set up an infrastructure to study power related optimizations
  - Hardware-software interaction: test benches, device drivers, transaction-level modeling
  - Continue to explore new examples: PowerPC
- Semantic extensions and associated compiling schemes
  - The sequential connective: Control over scheduling, Multi-cycle atomic actions
  - Recursive method calls
- Exploratory: Compiling Bluespec for multicores

The need for sequential connective

An Example: Table Lookup



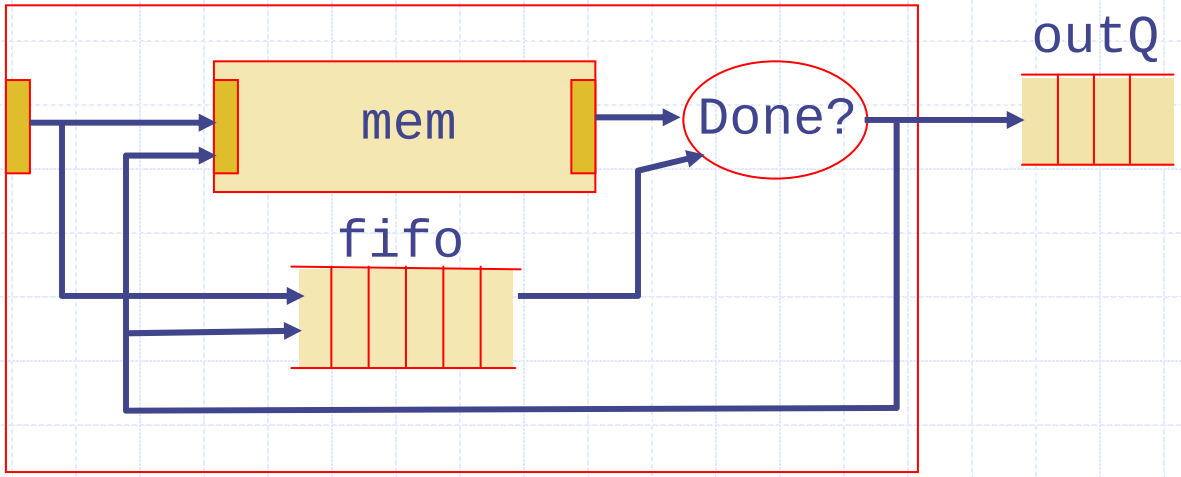
When can rule recirculate fire?

Made up syntax

```
module lpm
rule "recirculate"
   (x = mem.res() in (y = fifo.first() in
   (if done?(x) then fifo.deq() | mem.deq() | outQ.enq(x)
                else  (mem.deq() | mem.req(addr(x)))
                    | (fifo.deq() | fifo.enq(y)))

action method  enter(x) = mem.req(addr(x)) | fifo.enq(x)
```

Table lookup using the sequential connective




```
module lpm
rule "recirculate"
  (x = mem.peek() in (y = fifo.first() in
  (if done?(x) then fifo.deq() | mem.deq() | outQ.enq(x)
                else  (mem.deq(); mem.req(addr(x)))
                    | (fifo.deq(); fifo.enq(y)))

action method  enter(x) = mem.req(addr(x)) | fifo.enq(x)
```

Made up syntax

Notice