Bluespec: A New Way of Describing SoC Behavior

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

ISSoC, Tampere, Finland
November 14, 2006
The biggest SoC drivers

- Explosive growth in markets for
  - cell phones
  - game boxes
  - sensors and actuators

Functionality and applications are constrained primarily by:
- cost
- power/energy constrains
Will there be a greater or smaller variety of chips in a decade from now?

- **Cell phones, PDAs, sensors, ...**
  ⇒ much greater variety

- **Cost of development, business risks, ...**
  ⇒ smaller variety

*New design flows and tools can directly affect the outcome*
Current Cellphone Architecture

Today’s chip becomes a block in tomorrow’s chip

IP reuse is essential

Hardware/software migration

Complex, high-performance but must not dissipate more than 3 watts
An under appreciated fact

- If a functionality (e.g. H.264) is moved from a programmable device to a specialized hardware block, the power/energy savings are 100 to 1000 fold

Power savings $\Rightarrow$ more specialized hardware

but our mind set

- Software is forgiving
- Hardware design is difficult, inflexible, brittle, error prone, ...
SoC Trajectory: 
*multicores, heterogeneous, regular, ...*

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

- Design costs (hardware & software) dominate
- Within these costs verification and validation costs dominate
- IP reuse is essential to prevent design-team sizes from exploding

\[ \text{design cost} = \text{number of engineers} \times \text{time to design} \]
Common quotes

- “Design is not a problem; design is easy”
- “Verification is a problem”
- “Timing closure is a problem”
- “Physical design is a problem”

Mind set

Almost complete reliance on post-design verification for quality
Through the early 1980s:

The U.S. auto industry

- Sought quality solely through post-build inspection
- Planned for defects and rework

and U.S. quality was...
... less than world class

- Adding quality inspectors ("verification engineers") and giving them better tools, was not the solution
- The Japanese auto industry showed the way
  - "Zero defect" manufacturing
New mind set:

Design affects everything!

- A good design methodology
  - Can keep up with changing specs
  - Permits architectural exploration
  - Facilitates verification and debugging
  - Eases changes for timing closure
  - Eases changes for physical design
  - Promotes reuse

⇒ It is essential to

*Design for Correctness*
New Design flows & tools

*Synthesis* as opposed to *Decomposition*

- A method of designing and connecting modules such that the functionality and performance are predictable
  - Must facilitate natural descriptions of concurrent systems
- A method of refining individual modules into hardware or software for SoCs
- New techniques for producing SoCs so that the development costs are proportional to the changes
New ways of expressing behavior to reduce design complexity

 peny decentralize complexity: Rule-based specifications (Guarded Atomic Actions)
- Lets you think one rule at a time

Formalize composition: Modules with guarded interfaces
- Automatically manage and ensure the correctness of connectivity, i.e., correct-by-construction methodology

Bluespec

⇒ Smaller, simpler, clearer, more correct code
⇒ not just simulation, synthesis as well
Reusing IP Blocks

Example: Commercially available FIFO IP block

An error occurs if a push is attempted while the FIFO is full.

Thus, there is no conflict in a simultaneous push and pop operation when the FIFO is full. A simultaneous push and pop operation is allowed if the FIFO is empty, since there is no pop data to prefetch. However, the RAM read data must be captured on the next rising edge of clk. Thus, the RAM read data must be captured on the clk following the assertion of pop_req_n.

These constraints are spread over many pages of the documentation...
Bluespec promotes composition through guarded interfaces

```
theModuleA
  theFifo.enq(value1);
  theFifo.deq();
  value2 = theFifo.first();

theModuleB
  theFifo.enq(value3);
  theFifo.deq();
  value4 = theFifo.first();

theFifo
  Enqueue arbitration control
  Dequeue arbitration control

Self-documenting interfaces; Automatic generation of logic to eliminate conflicts in use.
```
Bluespec

- What is it?
- Programming with Rules
  - Example GCD
- Architectural Exploration
  - Example 802.11a transmitter
- Example: H.264

Bluespec is available in two versions:
BSV – Bluespec in System Verilog
ESEPro – Bluespec in SystemC
Bluespec: State and Rules organized into *modules*

All *state* (e.g., Registers, FIFOs, RAMs, ...) is explicit. *Behavior* is expressed in terms of atomic actions on the state:

Rule: condition $\rightarrow$ action

Rules can manipulate state in other modules only *via* their interfaces.
Programming with rules: A simple example

Euclid’s algorithm for computing the Greatest Common Divisor (GCD):

<table>
<thead>
<tr>
<th>15</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

answer: 3
module mkGCD (I_GCD);

Reg#(int) x <- mkRegU;
Reg#(int) y <- mkReg(0);

rule swap ((x > y) && (y != 0));
  x <= y;  y <= x;
endrule

rule subtract ((x <= y) && (y != 0));
  y <= y - x;
endrule

method Action start(int a, int b) if (y==0);
  x <= a;  y <= b;
endmethod

method int result() if (y==0);
  return x;
endmethod
endmodule

 Assumes x /= 0 and y16= 0
GCD Hardware Module

interface I_GCD;
  method Action start (int a, int b);
  method int result();
endinterface

The module can easily be made polymorphic

Many different implementations can provide the same interface:

module mkGCD (I_GCD)

In a GCD call \( t \) could be
Int#(32), UInt#(16), Int#(13), ...

implicit conditions

\[ y == 0 \]

\[ y == 0 \]
Architectural Exploration:

Area-Power tradeoff in 802.11a transmitter design
802.11a Transmitter Overview

- Controller
- Scrambler
- Encoder
- Interleaver
- Mapper
- IFFT
- Cyclic Extend

**IFFT Transforms** 64 (frequency domain) complex numbers into 64 (time domain) complex numbers

One OFDM symbol (64 Complex Numbers) accounts for 85% area

Must produce one OFDM symbol every 4 µsec

Uncoded bits:

- 24 Uncoded bits
## Preliminary results

[MEMOCODE 2006] Dave, Gerding, Pellauer, Arvind

<table>
<thead>
<tr>
<th>Design Block</th>
<th>Lines of Code (BSV)</th>
<th>Relative Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>49</td>
<td>0%</td>
</tr>
<tr>
<td>Scrambler</td>
<td>40</td>
<td>0%</td>
</tr>
<tr>
<td>Conv. Encoder</td>
<td>113</td>
<td>0%</td>
</tr>
<tr>
<td>Interleaver</td>
<td>76</td>
<td>1%</td>
</tr>
<tr>
<td>Mapper</td>
<td>112</td>
<td>11%</td>
</tr>
<tr>
<td>IFFT</td>
<td>95</td>
<td>85%</td>
</tr>
<tr>
<td>Cyc. Extender</td>
<td>23</td>
<td>3%</td>
</tr>
</tbody>
</table>

Complex arithmetic libraries constitute another 200 lines of code.
Combinational IFFT

All numbers are complex and represented as two sixteen bit quantities. Fixed-point arithmetic is used to reduce area, power, ...
Circular pipeline: Reusing the Pipeline Stage

16 Bfly4s can be shared but not the three permutations. Hence the need for muxes.
Superfolded circular pipeline:
Just one Bfly-4 node!

- 64, 4-way Muxes
- 4, 16-way Muxes
- Index Counter 0 to 15
- 4, 16-way DeMuxes
- Stage Counter 0 to 2
- Permute_1
- Permute_2
- Permute_3
- Bfly4

Input:
in0, in1, in2, in3, in4, ..., in63

Output:
out0, out1, out2, out3, out4, ..., out63
Which design consumes the least energy to transmit a symbol?

- Can we quickly code up all the alternatives?
  - single source with parameters?

Not practical in traditional hardware description languages like Verilog/VHDL
Expressing these designs in Bluespec was easy

- All these designs were done in less than one day!
  - Designers were experts in Bluespec
- Area and power estimates?

<table>
<thead>
<tr>
<th>Combinational</th>
<th>Pipelined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Folded (16 Bfly-4s)</td>
</tr>
<tr>
<td></td>
<td>Super-Folded (8 Bfly-4s)</td>
</tr>
<tr>
<td></td>
<td>Super-Folded (4 Bfly-4s)</td>
</tr>
<tr>
<td></td>
<td>Super-Folded (2 Bfly-4s)</td>
</tr>
<tr>
<td></td>
<td>Super-Folded (1 Bfly-4)</td>
</tr>
</tbody>
</table>

The same source code
802.11a Transmitter Synthesis results (Only the IFFT block is changing)

<table>
<thead>
<tr>
<th>IFFT Design</th>
<th>Area (mm²)</th>
<th>Symbol Latency (CLKs)</th>
<th>Throughput Latency (CLKs/sym)</th>
<th>Min. Freq Required</th>
<th>Average Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelined</td>
<td>5.25</td>
<td>12</td>
<td>04</td>
<td>1.0 MHz</td>
<td>4.92</td>
</tr>
<tr>
<td>Combinational</td>
<td>4.91</td>
<td>10</td>
<td>04</td>
<td>1.0 MHz</td>
<td>3.99</td>
</tr>
<tr>
<td>Folded (16 Bfly-4s)</td>
<td>3.97</td>
<td>12</td>
<td>04</td>
<td>1.0 MHz</td>
<td>7.27</td>
</tr>
<tr>
<td>Super-Folded (8 Bfly-4s)</td>
<td>3.69</td>
<td>15</td>
<td>06</td>
<td>1.5 MHz</td>
<td>10.9</td>
</tr>
<tr>
<td>SF(4 Bfly-4s)</td>
<td>2.45</td>
<td>21</td>
<td>12</td>
<td>3.0 MHz</td>
<td>14.4</td>
</tr>
<tr>
<td>SF(2 Bfly-4s)</td>
<td>1.84</td>
<td>33</td>
<td>24</td>
<td>6.0 MHz</td>
<td>21.1</td>
</tr>
<tr>
<td>SF (1 Bfly4)</td>
<td>1.52</td>
<td>57</td>
<td>48</td>
<td>12 MHz</td>
<td>34.6</td>
</tr>
</tbody>
</table>

TSMC .18 micron; numbers reported are before place and route. (DesignCompiler), Power numbers are from Sequence PowerTheater.
Note ...

- Bluespec language and tools have no knowledge of FFT or complex arithmetic ...

- Bluespec promotes concise implementation descriptions and synthesis, enabling *rapid architectural exploration*
Video Codec: H.264
Example: H.264 Decoder

A dataflow-like network

May be implemented in hardware or software depending upon ...
Available codes (not multithreaded)

- Reference code
  - 80K lines, awful coding style, slow
- ffmpeg code for Linux
  - 200K lines, mixed with other codecs
- Codes don’t reflect the dataflow structure
  - Pointers to data structures are passed around and modified. Difficult to figure out which block is modifying which parts
  - No model of concurrency. Even the streaming aspect gets obscured by the code

The code can be written in a style which will serve both hardware and software communities.
H.264 Decoder in Bluespec

*Work in Progress - Chun-Chieh Lin et al*

- Baseline profile, no inter-prediction
- Decodes 720p @ 55fps
- 7.3K lines of Bluespec

Concise

- Has been synthesized into hardware (RTL)
- Any module can be implemented in software
- Each module can be separately refined
- Hopefully a much easier source code for multicores

1. Each module embodies its own resources
2. The behaviors are composable in a concurrent setting
Summary

- Market forces are demanding a much greater variety of SoCs
- The design cost for SoCs has to be brought down dramatically by facilitating IP reuse
- High-level synthesis tools are essential for architectural exploration and IP development
- Bluespec facilitates the new design flow

Thanks