Raising the level of abstraction above RTL
Prof. Stephen A. Edwards
Students: Cristian Soviani, Jia Zeng (2007?)
Mike Kishinevsky, Intel
Results (Papers)

- Shannon decomposition plus retiming
  Soviani, Tardieu, & Edwards, DATE 2006

- High-level synthesis for router pipelines
  Soviani, Hadžić, & Edwards, DAC 2006 (submitted)

- More efficient “decyclification” algorithm
  Neiroukh, Edwards, & Song, ISVLSI 2006

- Separate compilation for Esterel (software)
  Zeng & Edwards, ICESS 2005

- Approximate Esterel reachability (formal)
  Tardieu & Edwards, ATVA 2005

- An Esterel virtual machine for small memories
  Plummer, Khajanchi, & Edwards, SLAP 2006
The Columbia Esterel Compiler

http://www1.cs.columbia.edu/~sedwards/cec/

V5-compliant open-source Esterel compiler

Backends for C, Verilog, BLIF, and VHDL

Written in C++

Source and Linux binaries available
case (cur_state) // synopsys parallel_case
IDLE: begin
    if (pcsu_powerdown & !jmp_e & !valid_diag_window) begin
        next_state = STANDBY_PWR_DN;
    end
    else if (valid_diag_window | ibuf_full | jmp_e) begin
        next_state = cur_state;
    end
    else if (icu_miss | cacheable) begin
        next_state = NC_REQ_STATE;
    end
    else if (icu_miss & !cacheable) begin
        next_state = REQ_STATE;
    end
    else next_state = cur_state;
end
NC_REQ_STATE: begin
    if (normal_ack | error_ack) begin
        next_state = IDLE;
    end
    else next_state = cur_state;
end
REQ_STATE: begin
    if (normal_ack) begin
        next_state = FILL_2ND_WD;
    end
    else if (error_ack) begin
        next_state = IDLE;
    end
    else next_state = cur_state;
end
FILL_2ND_WD: begin
    if (normal_ack) begin
        next_state = REQ_STATE2;
    end
    else if (error_ack) begin
        next_state = IDLE;
    end
    else next_state = cur_state;
end
REQ_STATE2: begin
    if (normal_ack) begin
        next_state = FILL_4TH_WD;
    end
    else if (error_ack) begin
        next_state = IDLE;
    end
    else next_state = cur_state;
end
FILL_4TH_WD: begin
    if (normal_ack) begin
        next_state = STANDBY_PWR_DN;
    end
    else if (error_ack) begin
        next_state = IDLE;
    end
    else next_state = cur_state;
end
STANDBY_PWR_DN: begin
    if (pcsu_powerdown | !jmp_e) begin
        next_state = IDLE;
    end
    else next_state = STANDBY_PWR_DN;
end
default: next_state = 7'bx;
endcase

loop
await
case [icu_miss and not cacheable] do
    await [normal_ack or error_ack]
end

case [icu_miss and cacheable] do
    abort
    await 4 normal_ack;
    when error_ack
end
case [pcsu_powerdown and not jmp_e and not valid_diag_window] do
    await [pcsu_powerdown and not jmp_e]
end
end;
pause
end
Why is Esterel More Succinct?

Verilog:

```verilog
REQ_STATE2: begin
    if(normal_ack) begin
        next_state = FILL_4TH_WD;
    end
    else if (error_ack) begin
        next_state = IDLE;
    end
    else next_state = cur_state;
end
```

Esterel:

```esterel
abort
await normal_ack
when error_ack
```

- Esterel provides cross-clock control-flow
- State machine logic represented implicitly
- Higher-level constructs like `await`
Shannon and Retiming

Cristian Soviani, Olivier Tardieu, and Stephen A. Edwards.

Optimizing Sequential Cycles through Shannon Decomposition and Retiming.

*Proceedings of Design Automation and Test in Europe (DATE).*

Munich, Germany, March 2006.
Motivating Example

Tight feedback loop improved by combining the two techniques
Treat Shannon as a Covering Problem

unchanged

Shannon with $\bullet$ as sel

unchanged

Shannon with $\bullet$ as sel

unchanged

Shannon

start Shannon

stop Shannon

extend Shannon

---

- p. 8/37
Considering Node Variants

\[ (10,10,14) \]

unchanged  Shannon  start Shannon  stop Shannon  extend Shannon
Considering Node Variants

unchanged  Shannon  start Shannon  stop Shannon  extend Shannon

(15)
Pruning Node Variants

\[
\text{fat}(f) = (14)(13,13,11) \\
\text{fat}(g) = (8)(7,7,7) \\
\text{fat}(h) = (15)(10,10,14) \\
\text{fat}(i) = (15)(14,14,14)
\]

PRUNING

minimum output arrival time
## Results on ISCAS89 Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Reference Period</th>
<th>Reference Area</th>
<th>Retimed Period</th>
<th>Retimed Area</th>
<th>Sh. + ret. Period</th>
<th>Sh. + ret. Area</th>
<th>Time (s)</th>
<th>Speed Up</th>
<th>Speed Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>s510</td>
<td>8</td>
<td>184</td>
<td>8</td>
<td>184</td>
<td>8</td>
<td>184</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s641</td>
<td>11</td>
<td>115</td>
<td>11</td>
<td>115</td>
<td>9</td>
<td>122</td>
<td>1.1</td>
<td>22%</td>
<td>6%</td>
</tr>
<tr>
<td>s713</td>
<td>11</td>
<td>118</td>
<td>11</td>
<td>118</td>
<td>10</td>
<td>121</td>
<td>0.9</td>
<td>10%</td>
<td>3%</td>
</tr>
<tr>
<td>s820</td>
<td>7</td>
<td>206</td>
<td>7</td>
<td>206</td>
<td>7</td>
<td>206</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s832</td>
<td>7</td>
<td>217</td>
<td>7</td>
<td>217</td>
<td>7</td>
<td>217</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s838</td>
<td>10</td>
<td>154</td>
<td>10</td>
<td>154</td>
<td>8</td>
<td>162</td>
<td>2.6</td>
<td>25%</td>
<td>5%</td>
</tr>
<tr>
<td>s1196</td>
<td>9</td>
<td>365</td>
<td>9</td>
<td>365</td>
<td>9</td>
<td>365</td>
<td>0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1423</td>
<td>24</td>
<td>408</td>
<td>21</td>
<td>408</td>
<td>13</td>
<td>460</td>
<td>3.8</td>
<td>61%</td>
<td>12%</td>
</tr>
<tr>
<td>s1488</td>
<td>6</td>
<td>453</td>
<td>6</td>
<td>453</td>
<td>6</td>
<td>453</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1494</td>
<td>6</td>
<td>456</td>
<td>6</td>
<td>456</td>
<td>6</td>
<td>456</td>
<td>0.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s9234</td>
<td>11</td>
<td>662</td>
<td>8</td>
<td>656</td>
<td>8</td>
<td>684</td>
<td>6.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s13207</td>
<td>14</td>
<td>1382</td>
<td>11</td>
<td>1356</td>
<td>9</td>
<td>1416</td>
<td>18.0</td>
<td>22%</td>
<td>4%</td>
</tr>
<tr>
<td>s38417</td>
<td>14</td>
<td>7706</td>
<td>14</td>
<td>7652</td>
<td>13</td>
<td>7871</td>
<td>113</td>
<td>7%</td>
<td>3%</td>
</tr>
</tbody>
</table>

Synthesis of High-Performance Packet Processing Pipelines.

Submitted to the Design Automation Conference (DAC).

San Francisco, California, July 2006.
Packet Switch Architecture

Ingress Packet Processor → Ingress Traffic Manager → Switching Fabric

Egress Packet Processor ← Egress Traffic Manager

Line Card
Typical Packet Pipeline

from fabric

VLAN pop

VLAN push

MPLS push

TTL update

ARP resolve
to network

memory lookup

memory lookup
Block Up into 64-bit Words
Add Cycle Boundaries and Delays

RTL synthesis straightforward from here

Able to achieve 40 GB/s on an FPGA: as good as by hand

Much easier than hand-coding RTL

Tool handles tedious bookkeeping, FSM synthesis
Osama Neiroukh, Stephen A. Edwards, and Xiaoyu Song.

An Efficient Algorithm for the Analysis of Cyclic Circuits.

*Proceedings of the International Symposium on VLSI (ISVLSI).*

Karlsruhe, Germany, March 2006.
Example
1: Apply controlling values

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Frontier</th>
<th>At Frontier</th>
<th>Acyclic</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a = 0}</td>
<td>{}</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>{b = 0}</td>
<td>{V}</td>
<td>R = 0</td>
<td></td>
</tr>
<tr>
<td>{c = 0}</td>
<td>{V}</td>
<td>U = 0</td>
<td></td>
</tr>
<tr>
<td>{d = 1}</td>
<td>{V}</td>
<td>U = 0</td>
<td></td>
</tr>
<tr>
<td>{e = 0}</td>
<td>{Z}</td>
<td>W = 1</td>
<td></td>
</tr>
<tr>
<td>{f = 1}</td>
<td>{Z}</td>
<td>X = 1</td>
<td></td>
</tr>
<tr>
<td>{g = 0}</td>
<td>{Z}</td>
<td>Y = 1</td>
<td></td>
</tr>
<tr>
<td>{g = 1}</td>
<td>{Z}</td>
<td>X = 1</td>
<td></td>
</tr>
</tbody>
</table>

2: Merge to “break logjams”

<table>
<thead>
<tr>
<th>Gate</th>
<th>Assignment</th>
<th>Frontier</th>
<th>Acyclic</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>{b = 0, c = 0}</td>
<td>{}</td>
<td>✓</td>
</tr>
<tr>
<td>V</td>
<td>{b = 0, d = 1}</td>
<td>{}</td>
<td>✓</td>
</tr>
<tr>
<td>Z</td>
<td>{e = 0, f = 1, g = 0}</td>
<td>{}</td>
<td>✓</td>
</tr>
</tbody>
</table>

Result:

{a = 0}
{b = 0, c = 0}
{b = 0, d = 1}
{e = 0, f = 1, g = 0}
## Experimental Results

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Netlist Gates</th>
<th>SCC Gates</th>
<th>[Edwards 03]</th>
<th>New</th>
<th>Acyclic PAs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PAs</td>
<td>time</td>
<td>PAs</td>
</tr>
<tr>
<td>arbiter5</td>
<td>213</td>
<td>25</td>
<td>257</td>
<td>1.3</td>
<td>25</td>
</tr>
<tr>
<td>arbiter6</td>
<td>248</td>
<td>30</td>
<td>745</td>
<td>8</td>
<td>29</td>
</tr>
<tr>
<td>arbiter7</td>
<td>283</td>
<td>35</td>
<td>2205</td>
<td>69</td>
<td>33</td>
</tr>
<tr>
<td>arbiter8</td>
<td>318</td>
<td>40</td>
<td>6581</td>
<td>656</td>
<td>37</td>
</tr>
<tr>
<td>exp</td>
<td>124</td>
<td>69</td>
<td>54517</td>
<td>2868</td>
<td>23260</td>
</tr>
<tr>
<td>ex1</td>
<td>150</td>
<td>47</td>
<td>43777</td>
<td>2341</td>
<td>232</td>
</tr>
<tr>
<td>gary</td>
<td>177</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>290</td>
</tr>
<tr>
<td>planet</td>
<td>253</td>
<td>51</td>
<td>-</td>
<td>-</td>
<td>1489</td>
</tr>
<tr>
<td>s1488</td>
<td>272</td>
<td>61</td>
<td>-</td>
<td>-</td>
<td>588</td>
</tr>
<tr>
<td>table3</td>
<td>311</td>
<td>49</td>
<td>-</td>
<td>-</td>
<td>3604</td>
</tr>
</tbody>
</table>

Much faster than the DAC 2003 paper’s algorithm
Connecting two synchronous blocks tricky: in what order should they be simulated?

Our solution: compile A and B such that they respond to “don’t know yet” inputs.
Convert If-Else to If-Else-Don’t-Know
Experimental Results

<table>
<thead>
<tr>
<th>Example</th>
<th>Lines</th>
<th>Average cycle times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Esterel V5</td>
</tr>
<tr>
<td>comexp</td>
<td>88</td>
<td>1.67s</td>
</tr>
<tr>
<td>iwls3</td>
<td>70</td>
<td>1.04s</td>
</tr>
<tr>
<td>3vsim2</td>
<td>48</td>
<td>0.68s</td>
</tr>
<tr>
<td>multi3</td>
<td>120</td>
<td>1.39s</td>
</tr>
</tbody>
</table>

Shows the cost of adding code that handles the “don’t-know” case is reasonable.
Olivier Tardieu and Stephen A. Edwards.

Approximate Reachability for Dead Code Elimination in Esterel*.

In *Proceedings of the Third International Symposium on Automated Technology for Verification and Analysis (ATVA).*

Taipei, Taiwan, October 2005.

An Esterel Virtual Machine for Embedded Systems.

Vienna, Austria, March 2006.
An Esterel Virtual Machine

Goal: software code generation for small embedded systems.

Basic idea: trade speed for program size by building a language-specific virtual machine.

Contributions: instruction-level support for concurrency, mating code synthesis algorithm.
### Experimental Results

#### Code sizes (percentage saved by VM):

<table>
<thead>
<tr>
<th>Example</th>
<th>BAL</th>
<th>x86</th>
<th>H8</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacexample</td>
<td>369</td>
<td>917</td>
<td>60%</td>
</tr>
<tr>
<td>abcd</td>
<td>870</td>
<td>2988</td>
<td>71%</td>
</tr>
<tr>
<td>greycounter</td>
<td>1289</td>
<td>3571</td>
<td>64%</td>
</tr>
<tr>
<td>tcint</td>
<td>5667</td>
<td>11486</td>
<td>51%</td>
</tr>
<tr>
<td>atds-100</td>
<td>10481</td>
<td>38165</td>
<td>73%</td>
</tr>
</tbody>
</table>

#### Execution Speeds (slowdown due to VM):

<table>
<thead>
<tr>
<th>Example</th>
<th>x86</th>
<th>BAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>dacexample</td>
<td>0.06μs</td>
<td>1.1μs</td>
</tr>
<tr>
<td>tcint</td>
<td>0.28μs</td>
<td>1.1μs</td>
</tr>
<tr>
<td>atds-100</td>
<td>0.20μs</td>
<td>1.4μs</td>
</tr>
</tbody>
</table>


Jia Zeng and Stephen A. Edwards.
Separate Compilation of Synchronous Modules.

Olivier Tardieu and Stephen A. Edwards.
Approximate Reachability for Dead Code Elimination in Esterel*.

An Esterel Virtual Machine for Embedded Systems.
In *Proceedings of Synchronous Lanugages, Applications, and Programming (SLAP)*.
Vienna, Austria, March 2006.
Stephen A. Edwards and Olivier Tardieu.  
SHIM: A Deterministic Model for Heterogeneous Embedded Systems.  

Stephen A. Edwards and Olivier Tardieu.  
Deterministic Receptive Processes are Kahn Processes.  

Cristian Soviani and Stephen A. Edwards.  
Challenges in Synthesizing Fast Control-Dominated Circuits.  
Stephen A. Edwards.
SHIM: A Language for Hardware/Software Integration.

Stephen A. Edwards.
The challenges of hardware synthesis from C-like languages.
In *Proceedings of Design Automation and Test in Europe (DATE)*, Munich, Germany, March 2005.

Generating Fast Code from Concurrent Program Dependence Graphs.
