Raising the level of abstraction above RTL

Prof. Stephen A. Edwards

Students: Cristian Soviani, Jia Zeng (2007?)

Mike Kishinevsky, Intel

*We intend to make Esterel a viable hardware description language for control-dominated systems by developing a compiler that produces optimized circuits from it.*
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 = 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;
    end
    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
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`
An Overview of Esterel

Synchronous model of time: implicit global clock

Communication through wire-like signals

Two flavors of statement:

**Combinational**
*Execute in one cycle*
- emit
- present / if
- loop

**Sequential**
*Take multiple cycles*
- pause
- await
- sustain
Basic Circuit Generation

```plaintext
loop
  emit A; await C;
  emit B; pause
end
```

Diagram:
- Entry node connected to:
  - A
  - B
  - C
- A connected to:
  - B
  - C
- B connected to:
  - C
- C connected to:
  - Entry
Berry’s technique [1992] works, but is fairly inefficient:

- Many combinational redundancies. E.g., present A then emit B end; present C then emit D end produces two redundant OR gates.

- Many sequential redundancies. One flop per pause can be very wasteful.

Touati, Toma, Sentovich, and Berry [1993–1997] proposed techniques to eliminate many, but requires reachable state space and only works on circuit.
Esterel’s semantics match hardware. Translation is straightforward.

Nice feature: state space is well-defined and hierarchical (e.g., due to abort and concurrency).

Enables a hierarchical state assignment/synthesis procedure.
A State Assignment Example

abort
[
    await A; await B
    ||
    await C
]
when D;
emit E;
pause;
[
    await F
    ||
    await G
]
Hierarchical States

abort

[ await A; await B
  ||
  await C
]

when D;
emit E;

pause;

[ await F
  ||
  await G
]
Five Simple FSMs

```
abort
[
    await A; await B
    ||
    await C
]
when D;
emit E;
pause;
[
    await F
    ||
    await G
]
```

High-level Synthesis from the Synchronous Language Esterel – p. 10/?
What does it take to select a good encoding?

Compared expensive automatic flow

V5 → SIS with sequential optimization

to human cleverness

CEC → manual encoding → SIS (combinational)
Discoveries

Many local optimizations possible

Matching SIS required knowing some state reachability

Really need a combination of both for quality results
Locally redundant constructs:

\[
\text{pause; sustain } S \\
\text{equivalent to} \\
\text{loop} \\
\text{pause; emit } S \\
\text{end loop}
\]
Simple Sequential Don’t-Cares

loop
  await ConflictOnSEL;
  do
    every immediate SEL do
      emit RejectSEL
  end
  watching AcceptSEL
end loop

Needed to know that ConflictOnSEL was never present in the first cycle.
Machines running in lock-step

This generated many sequential don’t-cares that were slowing the logic.
Sequential Redundancy

Signal emitted in a cycle where it is never tested.

% during first cycle :
% * start sustaining pWREQ: on the next cycle, we
% shall have WREQ and DMA address ready cycle after
% * prepare Lca drive for next cycle

emit pLcaDrives;
await tick;
% setup data path from pam to host
emit pPamDrives;
% ... emit pHostDrives;
## Results

<table>
<thead>
<tr>
<th>example</th>
<th>size</th>
<th>method</th>
<th>levels</th>
<th>LUTs</th>
<th>latches</th>
</tr>
</thead>
<tbody>
<tr>
<td>graycounter</td>
<td>91</td>
<td>V5 + blifopt</td>
<td>5</td>
<td>66</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manual</td>
<td>4</td>
<td>51</td>
<td>17</td>
</tr>
<tr>
<td>abcdef</td>
<td>142</td>
<td>V5 + blifopt</td>
<td>5</td>
<td>114</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manual</td>
<td>3</td>
<td>128</td>
<td>8</td>
</tr>
<tr>
<td>mem-ctrl</td>
<td>80</td>
<td>V5 + blifopt</td>
<td>3</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CEC + comb</td>
<td>3</td>
<td>52</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CEC + blifopt</td>
<td>3</td>
<td>27</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manual</td>
<td>2</td>
<td>31</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Original VHDL</td>
<td>2</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>mem-ctrl2</td>
<td>36</td>
<td>V5 + blifopt</td>
<td>2</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CEC + comb</td>
<td>2</td>
<td>23</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CEC + blifopt</td>
<td>2</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manual</td>
<td>2</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JEDI + comb</td>
<td>2</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>tcint</td>
<td>689</td>
<td>V5 + blifopt</td>
<td>5</td>
<td>93</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>manual</td>
<td>3</td>
<td>118</td>
<td>52</td>
</tr>
</tbody>
</table>
Shannon Decomp. for Retiming
Shannon Decomposition
After Retiming
More aggressive decomposition
"Tech mapping" Shannon

High-level Synthesis from the Synchronous Language Esterel – p. 23/7
“Tech mapping” Shannon
Overall Algorithm

- Registers become nodes with \(-p\) delay (negative clock period)
- Compute “complex” arrival times for variants at each node.
  - Bellman-Ford relaxation algorithm on the cyclic graph.
  - Number of variants pruned aggressively.
- Reconstruct the circuit: choose variant(s) of each node that satisfies these arrival times.
- Run normal retiming.
Delay/area tradeoff: 128-bit adder

Fast: 24s to compute 120 points (88s incl. SIS)
## ISCAS benchmark results

<table>
<thead>
<tr>
<th></th>
<th>reference period area</th>
<th>retimed period area</th>
<th>Sh. + ret. period area</th>
<th>time (s)</th>
<th>speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>s510</td>
<td>8 184</td>
<td>8 203</td>
<td>8 203</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>s641</td>
<td>11 115</td>
<td>10 147</td>
<td>8 210</td>
<td>0.9</td>
<td>25%</td>
</tr>
<tr>
<td>s713</td>
<td>11 118</td>
<td>10 150</td>
<td>9 212</td>
<td>0.7</td>
<td>11%</td>
</tr>
<tr>
<td>s820</td>
<td>7 206</td>
<td>7 258</td>
<td>7 258</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>s832</td>
<td>7 217</td>
<td>6 235</td>
<td>6 234</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>s838</td>
<td>10 154</td>
<td>11 235</td>
<td>8 373</td>
<td>1.9</td>
<td>25%</td>
</tr>
<tr>
<td>s1196</td>
<td>9 265</td>
<td>9 443</td>
<td>9 444</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>s1423</td>
<td>24 498</td>
<td>19 559</td>
<td>13 846</td>
<td>3.6</td>
<td>46%</td>
</tr>
<tr>
<td>s1488</td>
<td>6 453</td>
<td>6 485</td>
<td>6 484</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>s1494</td>
<td>6 456</td>
<td>6 488</td>
<td>6 487</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>s9234</td>
<td>11 662</td>
<td>9 851</td>
<td>7 1037</td>
<td>5.4</td>
<td>28%</td>
</tr>
</tbody>
</table>
Deliverables

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
Last Year’s Accomplishments

- LCTES paper on software backend
- IWLS paper on state-encoding experiments (submitted)
- IWLS paper on Shannon for Retiming (submitted)
- SLAP paper on SHIM language for hardware/software codesign
- IWLS paper on hardware synthesis from C
- LCTES paper on language for device drivers
Next Year’s Goals

- Shannon/Retiming flow on higher-level models
- Improved Shannon area synthesis
- Peephole state optimization algorithm
- Global, approximate reachability algorithm
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.  
