Raising the level of abstraction above RTL

Prof. Stephen A. Edwards

Students: Cristian Soviani, Jia Zeng (2007?)

Mike Kishinevsky, Intel

(somebody from TI? Motorola?)

We intend to make Esterel a viable hardware description language for control-dominated systems by developing a compiler that produces optimized circuits from it.
Motivation: Rising Design Cost

1981: 100 designer-months for leading-edge chip
   10k transistors, 100 transistors/month

2002: 30 000 designer-months
   150M transistors, 5000 transistors/month

Design cost increased from $1M to $300M
Why Consider Esterel for Hardware?

- Semantics more abstract than RTL
  More succinct: easier to write faster
- High-level semantics enable optimizations
  State assignment a hierarchical problem
- Semantics enable efficient simulation
  No event queue
  Closer to an imperative program
- Esterel’s semantics are deterministic
  Simulation-synthesis mismatches eliminated
Applications of Esterel

Systems with complex (non-pipelined) control-behavior:

- DMA controllers
- Cache controllers
- Communication protocols

(Not processors)
Verilog More Verbose Than Esterel

```verilog
// Example Verilog code

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;
    endNC_REQ_STATE: begin
        if (normal_ack | error_ack) begin
            next_state = IDLE;
        end
        else next_state = cur_state;
    endREQ_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;
    endFILL_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;
    endREQ_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;
    endFILL_4TH_WD: begin
        if (normal_ack | error_ack) begin
            next_state = IDLE;
        end
        else next_state = cur_state;
    endSTANDBY_PWR_DN: begin
        if (!pcsu_powerdown | jmp_e) begin
            next_state = IDLE;
        end
        else next_state = STANDBY_PWR_DN;
    enddefault: next_state = 7'bX;
endcase
```
Why is Esterel More Succinct?

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:

abort
await normal_ack
when error_ack

- Esterel provides cross-clock control-flow
- State machine logic represented implicitly
- Higher-level constructs like \texttt{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
emit B;
present C then
emit D end;

Force signal present in this cycle
Make D present if C is
An Example

\begin{verbatim}
await A;
emit B;
present C then
  emit D end;
pause
\end{verbatim}

Wait for next cycle where A is present

Wait for next cycle
An Example

loop
  await A;
  emit B;
  present C then
    emit D end;
  pause
end

Infinite Loop
An Example

```plaintext
loop
  await A;
  emit B;
  present C then
    emit D end;
  pause
end
```

```
Run Concurrently
```

```plaintext
loop
  present B then
    emit C end;
  pause
end
```
every R do
    loop
        await A;
        emit B;
        present C then emit D end;
        pause
    end
|   | loop
    present B then emit C end;
    pause
end
end

Restart on R
An Example

every R do
  loop
    await A;
    emit B;
    present C then
      emit D end;
    pause
  end
end

Same-cycle bidirectional communication

High-level Synthesis from the Synchronous Language Esterel – p. 13/30
An Example

every R do
  loop
    await A;
    emit B;
    present C then
      emit D end;
    pause
  end
end

Good for hierarchical FSMs

Bad at manipulating data

Esterel V7 variant proposed to address this
loop
  emit A; await C;
  emit B; pause
end
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

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

Obvious questions:

- How should each state machine be encoded?
- Should state be shared between the AB/F and C/G machines?
States in an Esterel program an arbitrary tree of sequential and parallel state machines.
Choosing an Encoding

How should $s_1, \ldots, s_4$ be encoded?

Should $s_2$ or $s_3$ be shared with $s_4$ or $s_5$?
Choosing a Good Encoding

Goal: The smallest circuit meeting a timing constraint

1. Start with large, fast circuit (one-hot, no sharing)
2. Estimate the slack at each state decision point by estimating how much the delay could be increased at that point while still meeting the timing requirement
3. Share states at the lowest decision point with largest slack or reencode the widest-fanout decision point with sufficient slack
4. Repeat steps 2–3 until no further gain
## Results

<table>
<thead>
<tr>
<th>Example</th>
<th>Literals</th>
<th>SIS</th>
<th>Latches</th>
<th>Levels</th>
<th>Slices</th>
<th>Period (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V5</td>
<td>CEC</td>
<td>V5</td>
<td>CEC</td>
<td>V5</td>
<td>CEC</td>
</tr>
<tr>
<td>Figure 1a</td>
<td>23</td>
<td>15</td>
<td>15</td>
<td>(0)</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>dacexample</td>
<td>41</td>
<td>23</td>
<td>22</td>
<td>(0)</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>jacky1</td>
<td>39</td>
<td>22</td>
<td>20</td>
<td>(0)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>runner</td>
<td>218</td>
<td>145</td>
<td>144</td>
<td>(24)</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>greycounter</td>
<td>240</td>
<td>173</td>
<td>142</td>
<td>(6)</td>
<td>34</td>
<td>18</td>
</tr>
<tr>
<td>scheduler</td>
<td>519</td>
<td>380</td>
<td>74</td>
<td>(52)</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>servos</td>
<td>407</td>
<td>287</td>
<td>60</td>
<td>(16)</td>
<td>47</td>
<td>10</td>
</tr>
<tr>
<td>abcd</td>
<td>167</td>
<td>165</td>
<td>17</td>
<td>(0)</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>tcint</td>
<td>508</td>
<td>414</td>
<td>95</td>
<td>(14)</td>
<td>60</td>
<td>17</td>
</tr>
</tbody>
</table>

20% smaller, run at comparable speeds.

*Not the final word.*
Deliverables

The Columbia Esterel Compiler

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

V5-compliant open-source Esterel compiler

Backends for C, Verilog, BLIF, and (soon) VHDL

Written in C++

Source and Linux binaries available
Last Year’s Accomplishments

- CEC hardware backend released
- DATE paper on hardware backend (rejected)
- CEC software backend released
- SLAP 2004 paper on software backend
- New software backend created (not released)
- LCTES paper on new software backend (submitted)
- DAC 2003 paper on attacking cyclic circuits
Next Year’s Goals

- Release of second software backend
- Release of VHDL backend
- Automated state assignment algorithm
- Publication on Esterel state assignment
- Verification?
- Software synthesis with timing constraints?


