High-level Synthesis from the Synchronous Language Esterel

2004 MDC Conference

Stephen A. Edwards

Columbia University
Technical Challenges

Real-time

Concurrency

Complexity

Legacy Languages

Photo by Thomas Denoghue
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
Domain-Specific Languages

Little languages that fit the problem
More succinct description that are

1. Quicker to create
2. Easier to get right

More opportunities for optimization and analysis

General-purpose languages hindered by undecidability
Domain-specific languages much simpler
Device drivers are those pieces of software that you absolutely need that never seem to work.

Big security/reliability hole: run in Kernel mode.

Responsible for 80% of all Windows crashes.

Tedious, difficult-to-write.

Ever more important as customized hardware proliferates.
Ongoing Work

Develop language for network card drivers under Linux (Chris Conway)

Sharing drivers between Linux and FreeBSD (Tom Heydt-Benjamin)

Ultimate vision: compiler takes two programs: device spec. and OS spec. and synthesizes appropriate driver.

OS vendor makes sure OS spec. is correct; Hardware designer makes sure hardware spec. is correct.
ioports ne2000 {
  bits cr {
    bit stop, sta, transmit;
    enum:3 { 001=remRead, 010=remWrite,
            011=sendPacket, 1**=DMAdone }
    enum:2 { 00=page0, 01=page1, 10=page2 }
  }
  paged p {
    page0 { cr.page0; } {
      twobyte clda;
      byte bnry;
      bits tsr {
        bit ptx,1,col,abt,crs,0,cdh,owc;
      }
    }
    page1 { cr.page1; } {
      byte:6 par;
      byte curr;
      byte:8 mar;
    }
  }
}
Synchronous language developed by Gérard Berry in France

Basic idea: use global clock for synchronization in software like that in synchronous digital hardware.

Challenge: How to combine concurrency, synchronization, and instantaneous communication
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
An Example

emit B;

present C then
emit D end;

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

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

- `await A;` Wait for next cycle where A is present.
- `pause` Wait for next cycle.
An Example

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

Run Concurrently

loop
    present B then
        emit C end;
    pause
    end
An Example

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

| |
| loop
  present B then
  emit C end;
  pause
end
end
An Example

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

Same-cycle bidirectional communication

| loop
| present B then
| emit C end;
| pause
end
end
An Example

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

Good for hierarchical FSMs
Bad at manipulating data
Esterel V7 variant proposed to address this
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)
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_MD;
end
else if (error_ack) begin
next_state = IDLE;
end
else next_state = cur_state;
end
FILL_2ND_MD: 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_MD;
end
else if (error_ack) begin
next_state = IDLE;
end
else next_state = cur_state;
end
FILL_4TH_MD: begin
if (normal_ack) begin
next_state = Req_STATE;
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
endcase

High-level Synthesis from the Synchronous Language Esterel – p. 20/27
Basic Circuit Generation

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

entry

A

C

B

High-level Synthesis from the Synchronous Language Esterel – p. 21/2
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.
Hierarchical States

```plaintext
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. 23/27
Five Simple FSMs

abort
[
    await A; await B
    ||
    await C
]
when D;
emit E;
pause;
[
    await F
    ||
    await G
]
States in an Esterel program an arbitrary tree of sequential and parallel state machines.

Assign states to local machines to optimize global circuit.
### 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 CEC hand</td>
<td>V5 CEC hand</td>
<td>V5 CEC hand</td>
<td>V5 CEC hand</td>
<td>V5 CEC hand</td>
<td>V5 CEC hand</td>
</tr>
<tr>
<td>Figure 1a</td>
<td>23</td>
<td>15</td>
<td>15</td>
<td>6 (0)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>dacexample</td>
<td>41</td>
<td>23</td>
<td>22</td>
<td>7 (0)</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>jacky1</td>
<td>39</td>
<td>22</td>
<td>20</td>
<td>5 (0)</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>runner</td>
<td>218</td>
<td>145</td>
<td>144</td>
<td>30 (24)</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>greycounter</td>
<td>240</td>
<td>173</td>
<td>142</td>
<td>34 (6)</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>scheduler</td>
<td>519</td>
<td>380</td>
<td>74 (52)</td>
<td>8</td>
<td>8</td>
<td>80</td>
</tr>
<tr>
<td>servos</td>
<td>407</td>
<td>287</td>
<td>60 (16)</td>
<td>10</td>
<td>10</td>
<td>105</td>
</tr>
<tr>
<td>abcd</td>
<td>167</td>
<td>165</td>
<td>17 (0)</td>
<td>7</td>
<td>8</td>
<td>43</td>
</tr>
<tr>
<td>tcint</td>
<td>508</td>
<td>414</td>
<td>95 (14)</td>
<td>17</td>
<td>9</td>
<td>115</td>
</tr>
</tbody>
</table>

20% smaller, run at comparable speeds.  
*Not the final word.*
The Columbia Esterel Compiler

- Open-Source C++
- Hardware generation
- Software generation

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