

High Level Synthesis from the Synchronous Language Esterel 1068.00

Raising the level of abstraction above RTL

Prof. Stephen A. Edwards

Students: Cristian Soviani, Jia Zeng (2007?)

Mike Kishinevsky, Intel

(somebody from TI? Motorola?)

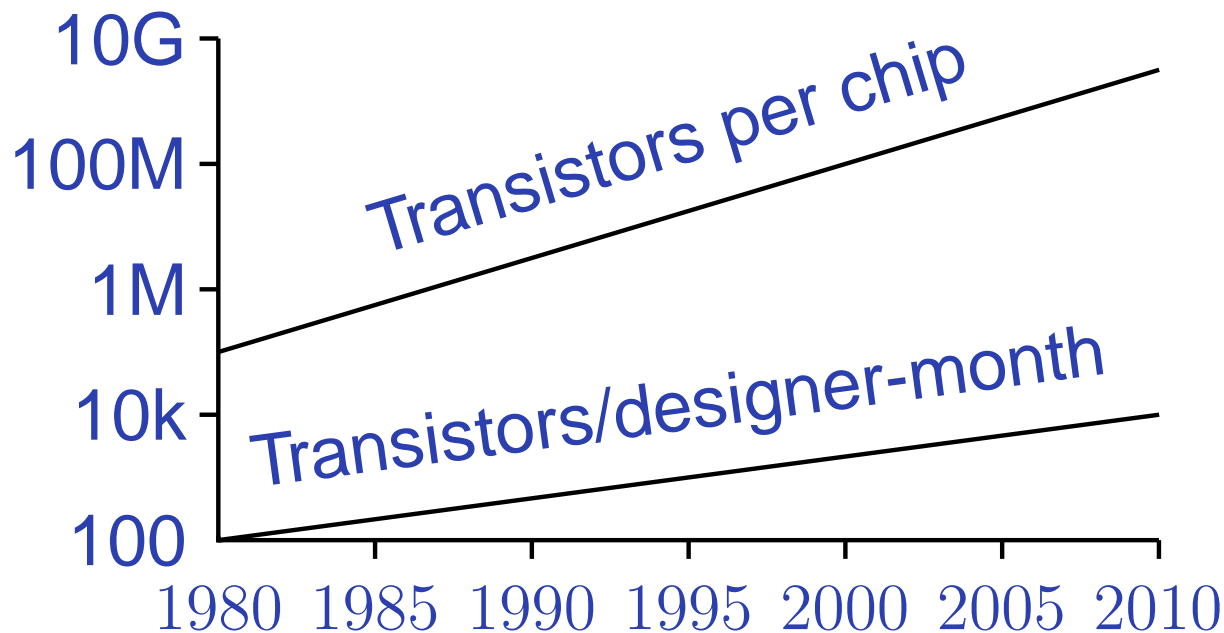
We intend to make Esterel a viable hardware description language for control-dominated systems by developing a compiler that produces optimized circuits from it.

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

```
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 (ic_u_miss & !cacheable) begin
      next_state = NC_REQ_STATE ;
    end
    else if (ic_u_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 | 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 [ic_u_miss and
        not cacheable] do
    await [normal_ack or error_ack]
  end
  case [ic_u_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:

```
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 *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

An Example

emit B; ← Force signal present in this cycle
present C then ← Make D present if C is
emit D end;

An Example

```
await A; ← Wait for next cycle where A is present  
emit B;  
present C then  
  emit D end;  
pause ← Wait for next cycle
```

An Example

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

An Example

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

Same-cycle bidirectional communication

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

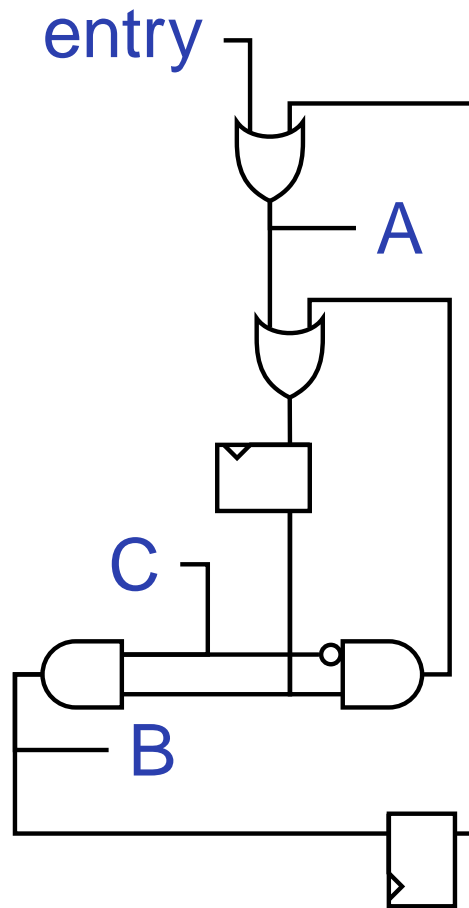
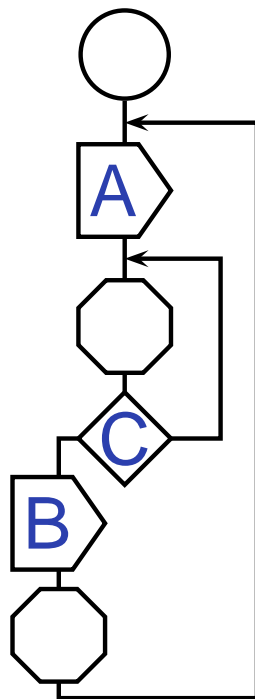
Bad at manipulating data

Esterel V7 variant proposed
to address this

Basic Circuit Generation

```

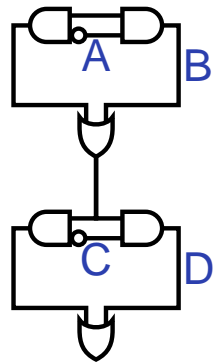
loop
  emit A; await C;
  emit B; pause
end
  
```



Basic Circuit Generation

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.



Generating Fast Circuits

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
]
```

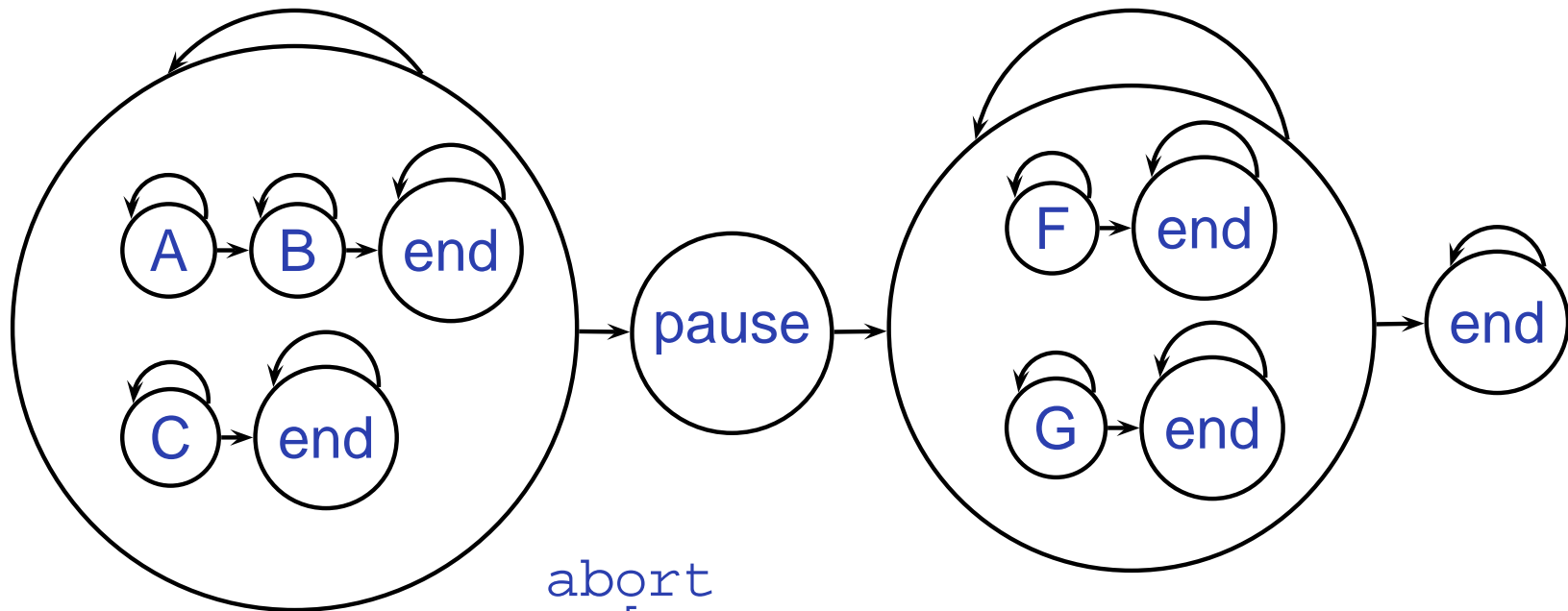
Hierarchical States

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

```
pause;
```

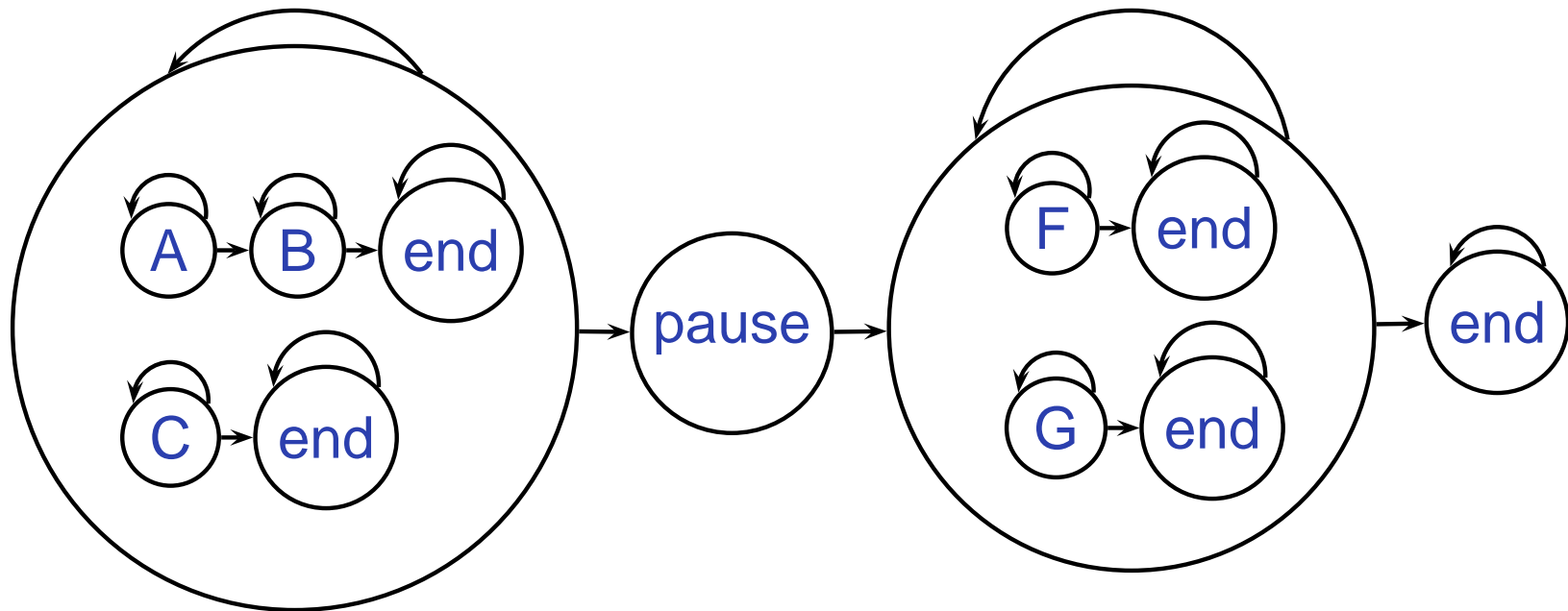
```
[  
  await F  
  ||  
  await G  
]
```

Five Simple FSMs



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

Five Simple FSMs

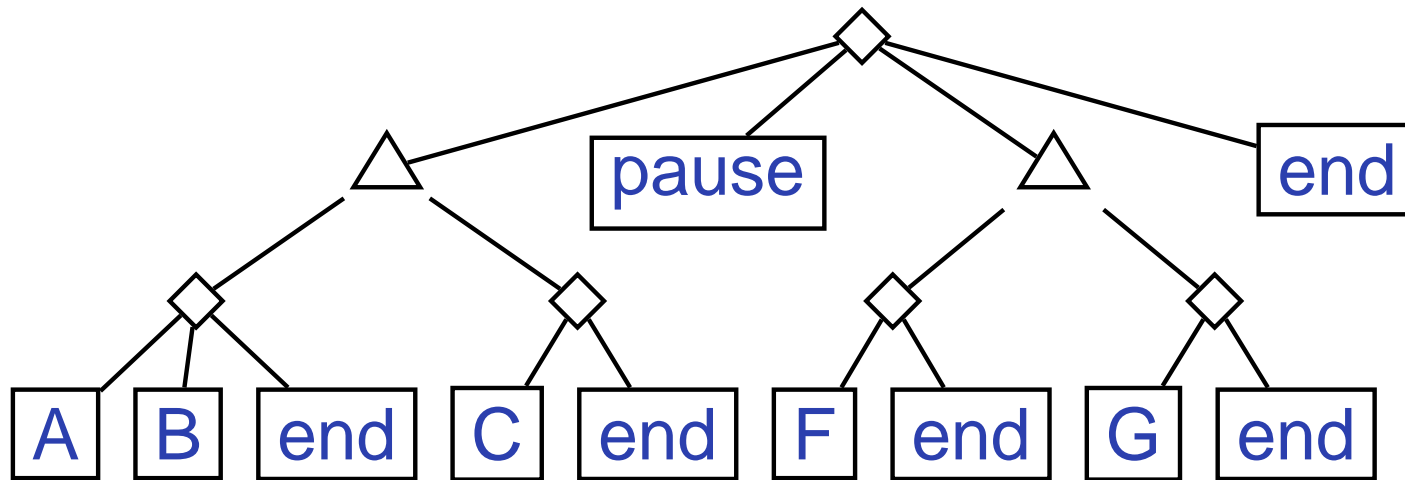


Obvious questions:

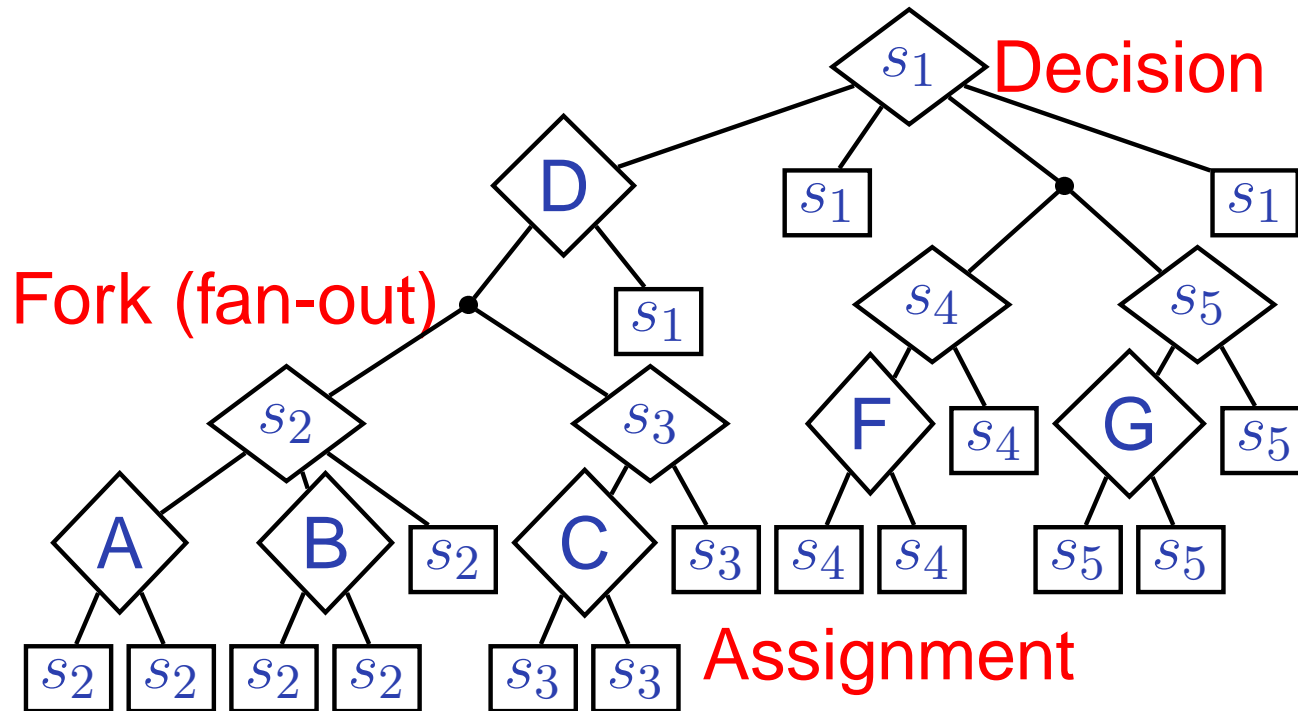
- How should each state machine be encoded?
- Should state be shared between the AB/F and C/G machines?

General Problem Statement

States in an Esterel program an arbitrary tree of sequential and parallel state machines.



Choosing an Encoding



- How should s_1, \dots, 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

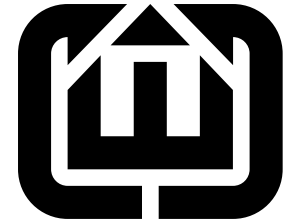
Results

Example	SIS									Xilinx					
	Literals			Latches			Levels			Slices			Period (ns)		
	V5	CEC	hand	V5	CEC	hand	V5	CEC	hand	V5	CEC	hand	V5	CEC	hand
Figure 1a	23	15	15	6 (0)	5	5	4	3	3	7	4	4	4.7	4.6	4.4
dacexample	41	23	22	7 (0)	5	5	5	3	3	10	5	5	6.2	6.0	5.5
jacky1	39	22	20	5 (0)	4	4	4	3	3	6	5	4	5.4	6.1	5.0
runner	218	145	144	30 (24)	20	20	11	10	10	56	36	35	10.6	8.4	8.1
greycounter	240	173	142	34 (6)	18	15	11	13	9	40	34	17	12.4	13.4	8.9
scheduler	519	380		74 (52)	55		8	8		80	66		11.3	8.9	
servos	407	287		60 (16)	47		10	10		105	66		16.7	13.4	
abcd	167	165		17 (0)	13		7	8		43	43		12.8	12.5	
tcint	508	414		95 (14)	60		17	9		115	81		10.8	10.9	

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?

Publications 1

Stephen A. Edwards, Vimal Kapadia, and Michael Halas.
Compiling Esterel into Static Discrete-Event Code.
In *Proceedings of Synchronous Languages, Applications, and Programming (SLAP 2004)*. Barcelona, Spain, March 28, 2004.

Stephen A. Edwards.
Making Cyclic Circuits Acyclic.
In *Proceedings of the 40th Design Automation Conference (DAC 2003)*. Anaheim, California, June 2-6, 2003. pp. 159-162.

Stephen A. Edwards.
Compiling Concurrent Languages for Sequential Processors.
ACM Transactions on Design Automation of Electronic Systems (TODAES) 8(2):141-187, April 2003.

Publications 2

Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone.
The Synchronous Languages 12 Years Later.
Proceedings of the IEEE 91(1):64-83, January 2003.

Stephen A. Edwards.
High-level Synthesis from the Synchronous Language Esterel.
In *Proceedings of the International Workshop of Logic and Synthesis (IWLS)*. New Orleans, Louisiana, June, 2002.

Stephen A. Edwards.
ESUIF: An Open Esterel Compiler.
In *Proceedings of Synchronous Languages, Applications, and Programming (SLAP)*. Grenoble, France, April 13, 2002.