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

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

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(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| 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:

```
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

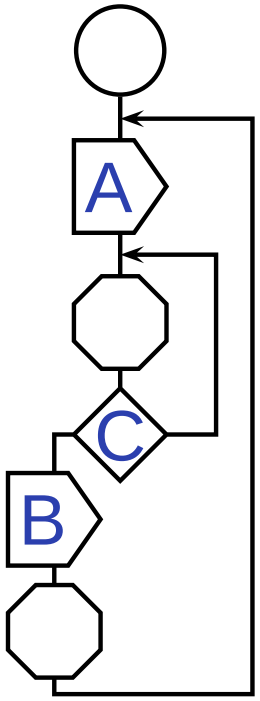
Communication through wire-like signals

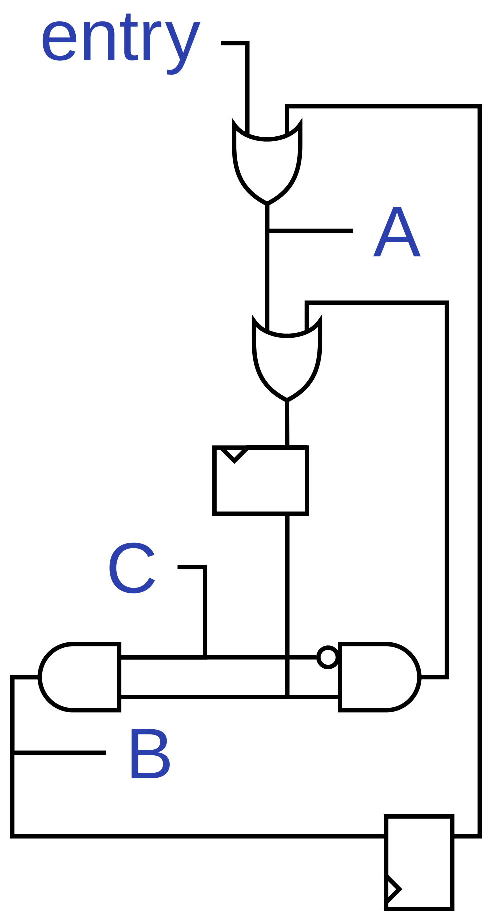
Two flavors of statement:

Combinational	**Sequential**
Execute in one cycle	*Take multiple cycles*
emit	pause
present / if	await
loop	sustain

Basic Circuit Generation

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

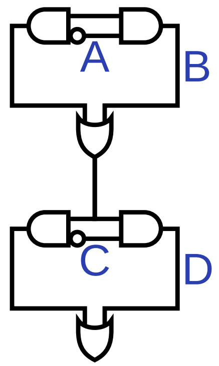




Basic Circuit Generation

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.

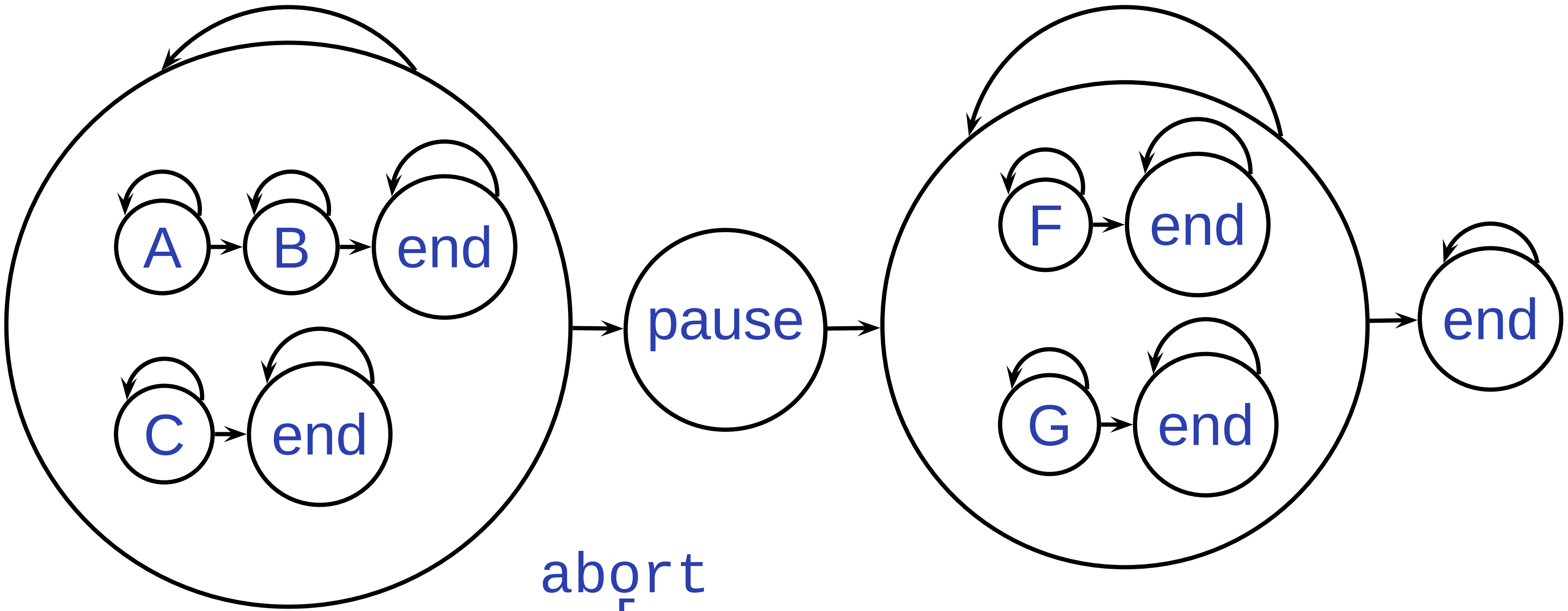
A State Assignment Example

```
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
]
```

Encoding Experiment

What does it take to select a good encoding?

Compared expensive automatic flow

V5 $\rightarrow$ SIS with sequential optimization

to human cleverness

CEC $\rightarrow$ manual encoding $\rightarrow$ SIS (combinational)

Discoveries

Many local optimizations possible

Matching SIS required knowing some state reachability

Really need a combination of both for quality results

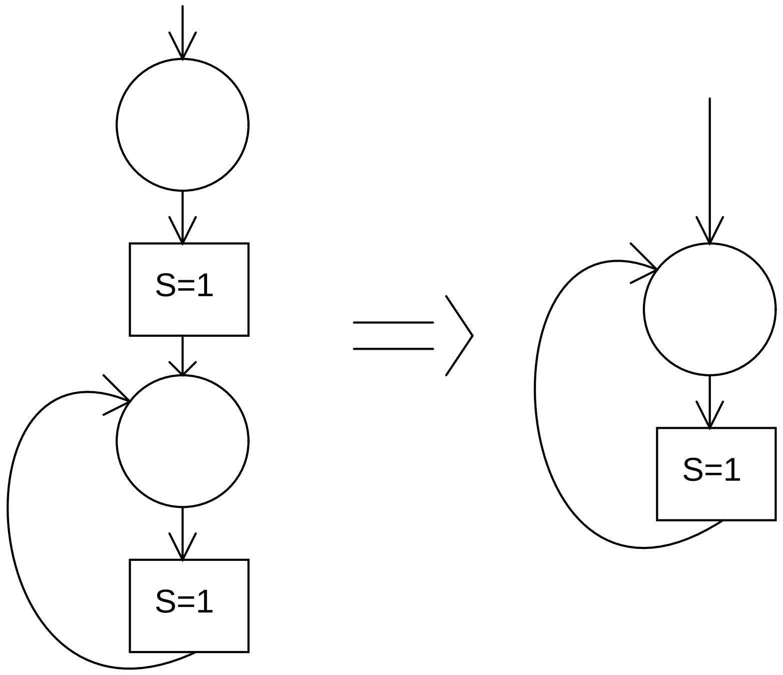
Local Redundancy

Locally redundant constructs:

```
pause;
sustain S
```

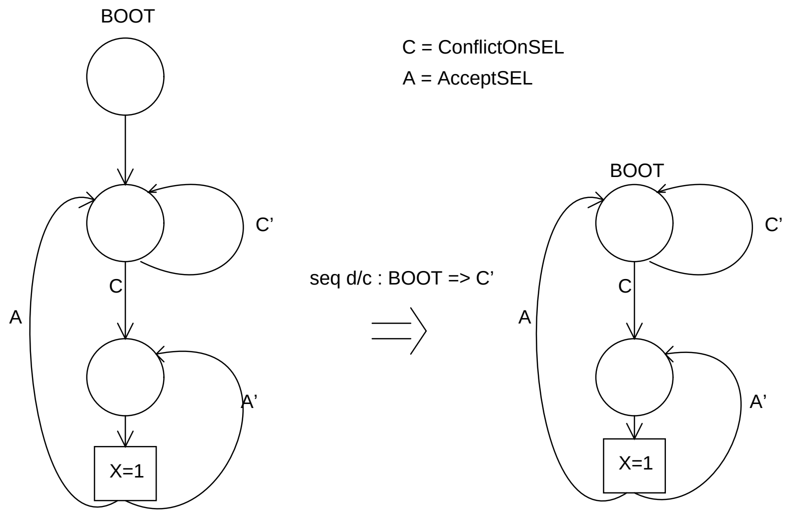
equivalent to

```
loop
  pause;
  emit S
end loop
```



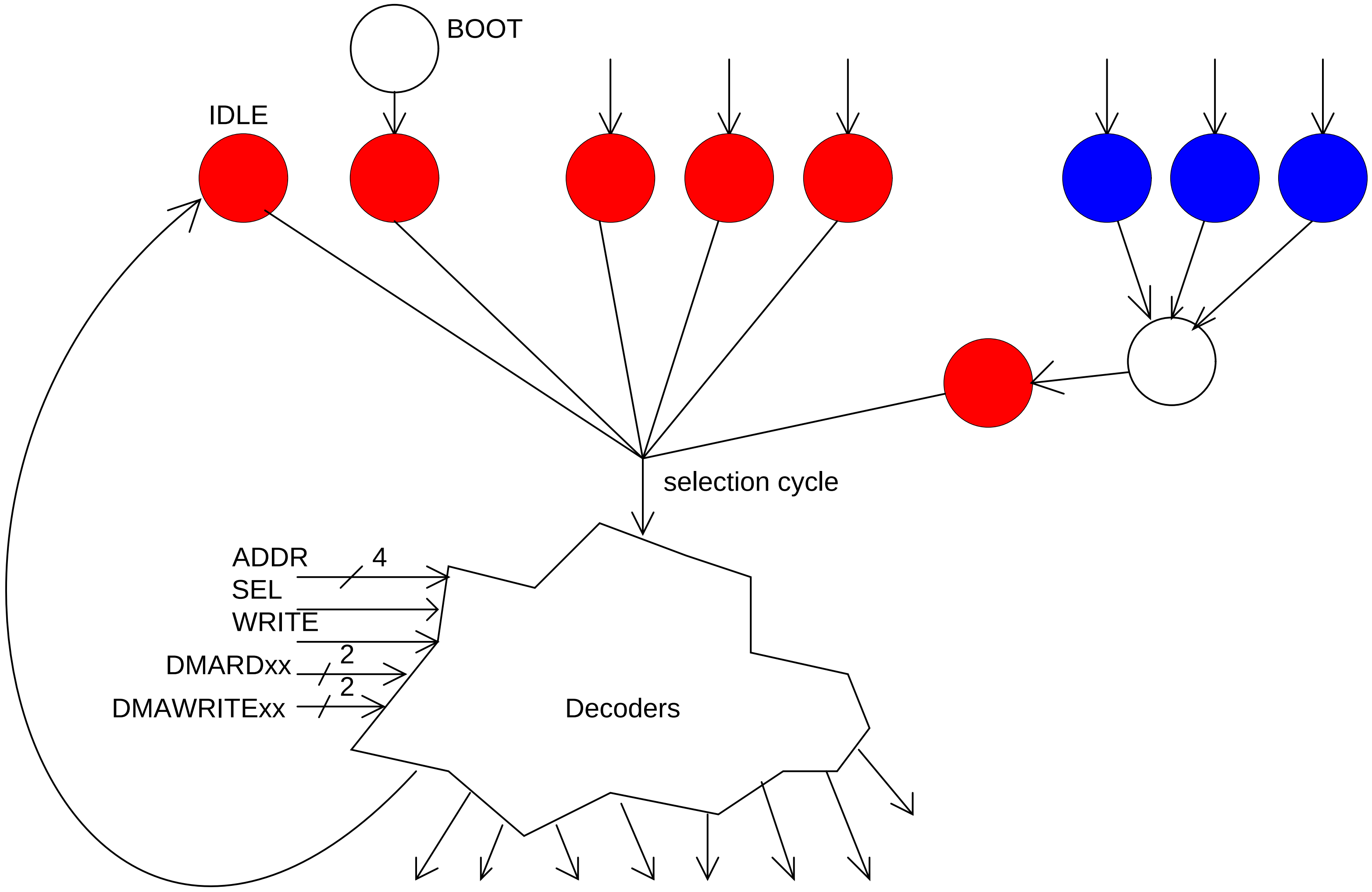
Simple Sequential Don't-Cares

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

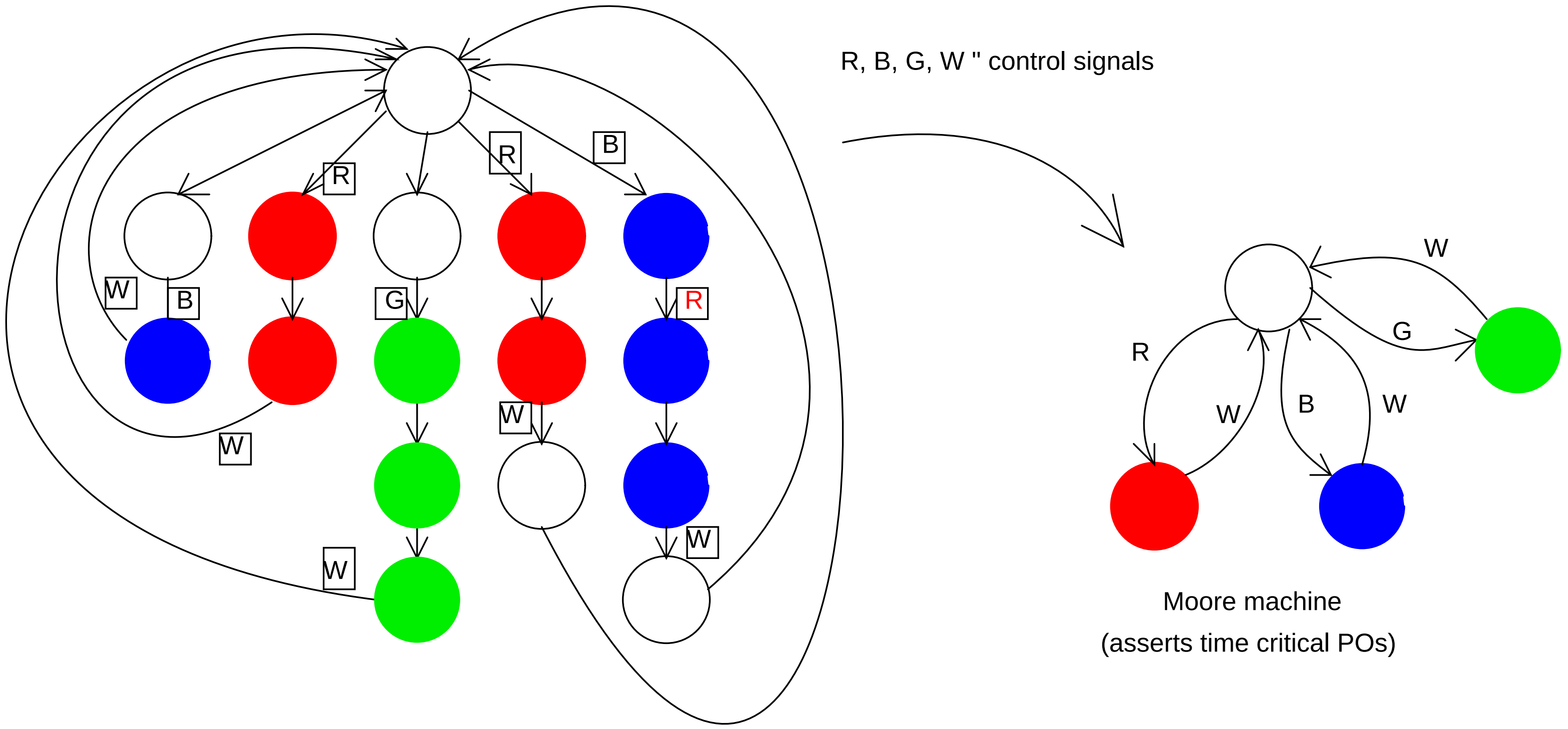


Needed to know that ConflictOnSEL was never present in the first cycle.

Equivalent States, Redundant Logic



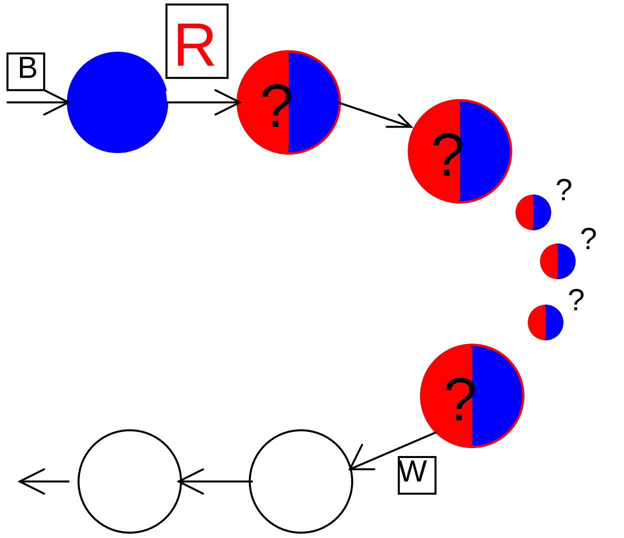
Machines running in lock-step

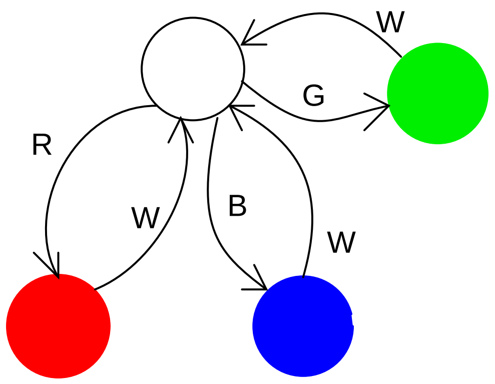


This generated many sequential don’t-cares that were slowing the logic.

Sequential Redundancy

Signal emitted in a cycle where it is never tested.





pLcaDrives : B
pPamDrives : R
pRomDrives : G

pHostDrives : W

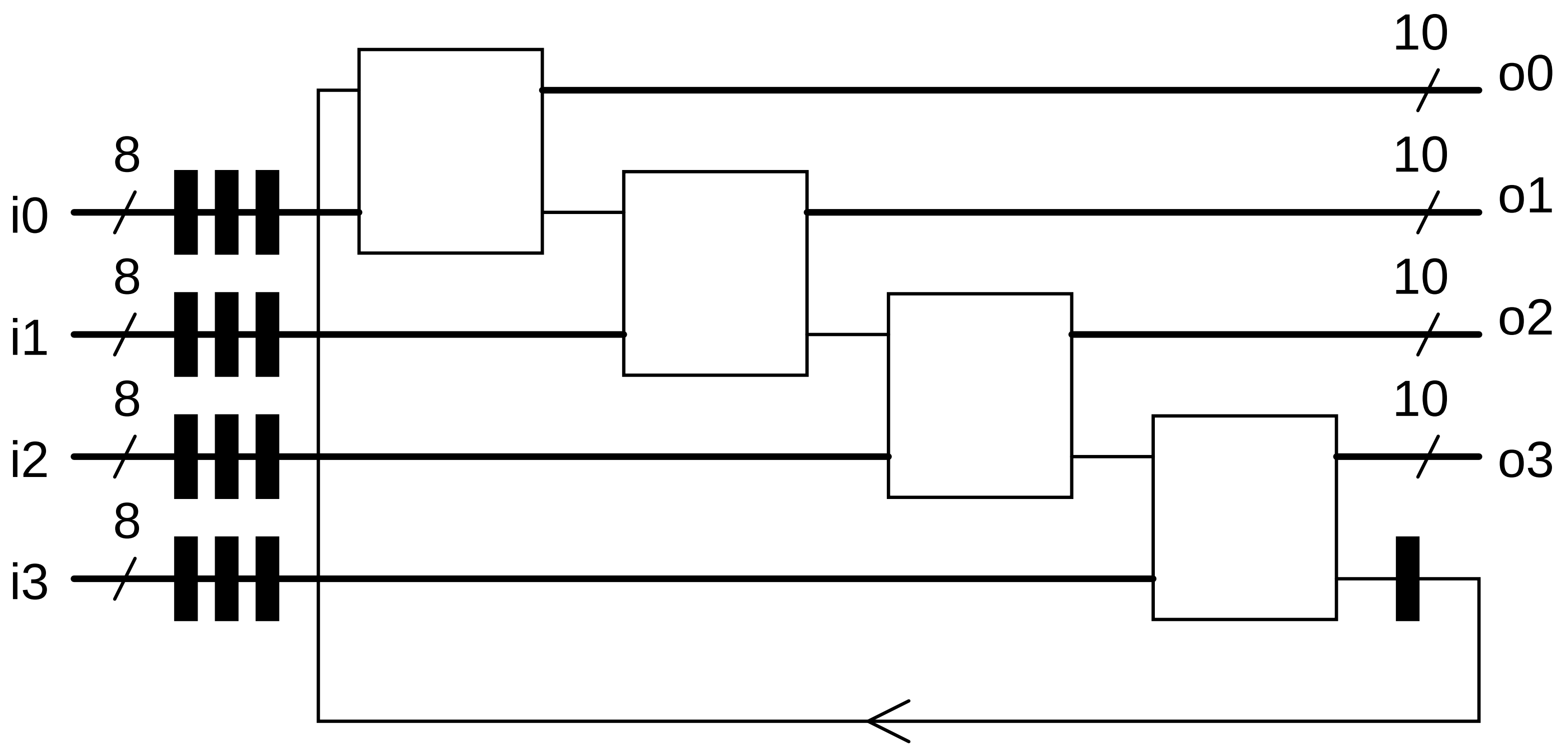
```
% during first cycle :
% * start sustaining pWREQ: on the next cycle, we
%   shall have WREQ and DMA address ready cycle after
% * prepare Lca drive for next cycle

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

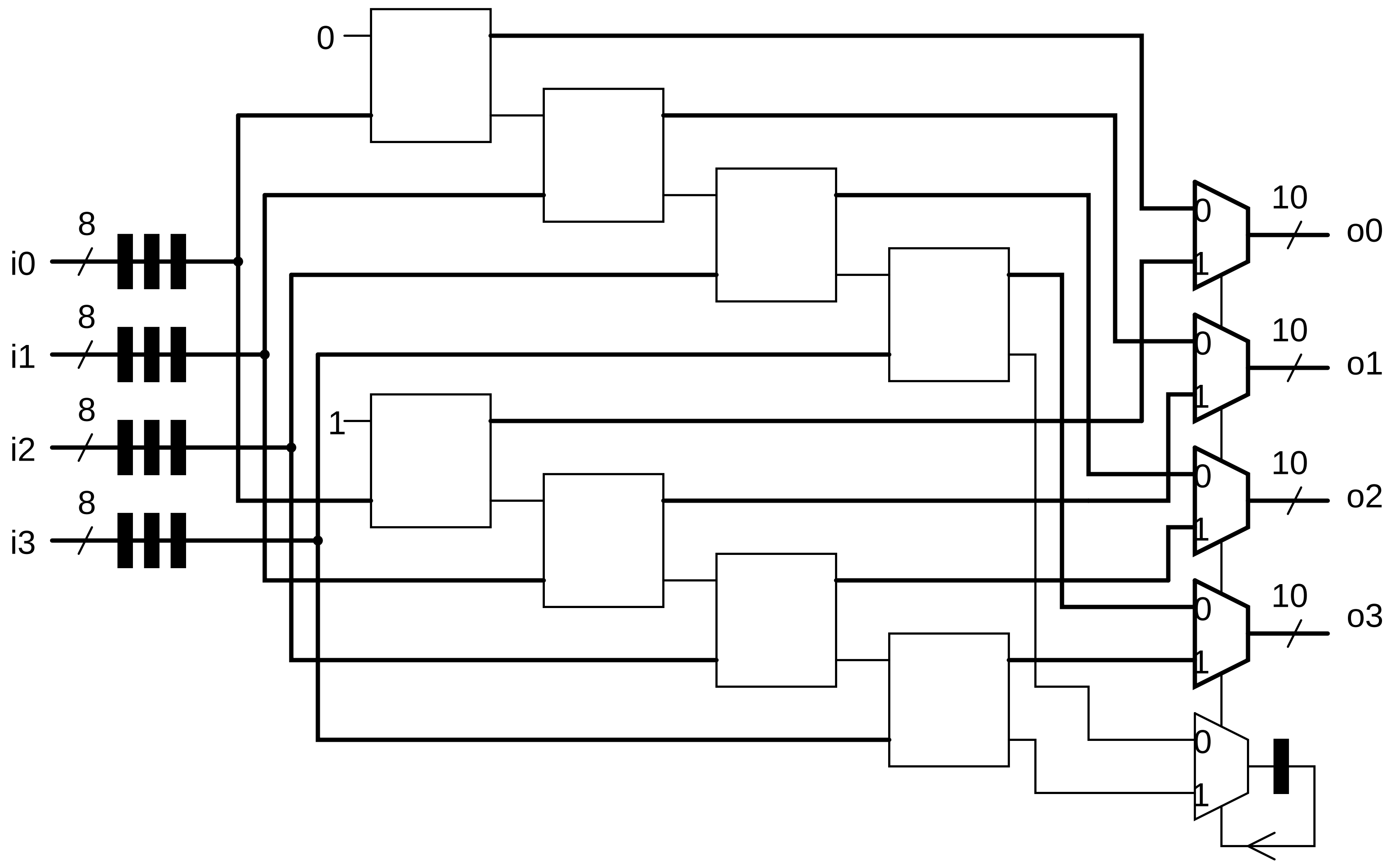
Results

example	size	method	levels	LUTs	latches
graycounter	91	V5 + blifopt	5	66	27
		manual	4	51	17
abcdef	142	V5 + blifopt	5	114	25
		manual	3	128	8
mem-ctrl	80	V5 + blifopt	3	24	16
		CEC + comb	3	52	17
		CEC + blifopt	3	27	15
		manual	2	31	13
		Original VHDL	2	17	11
mem-ctrl2	36	V5 + blifopt	2	17	8
		CEC + comb	2	23	9
		CEC + blifopt	2	18	8
		manual	2	14	3
		JEDI + comb	2	14	3
tcint	689	V5 + blifopt	5	93	52
		manual	3	118	52

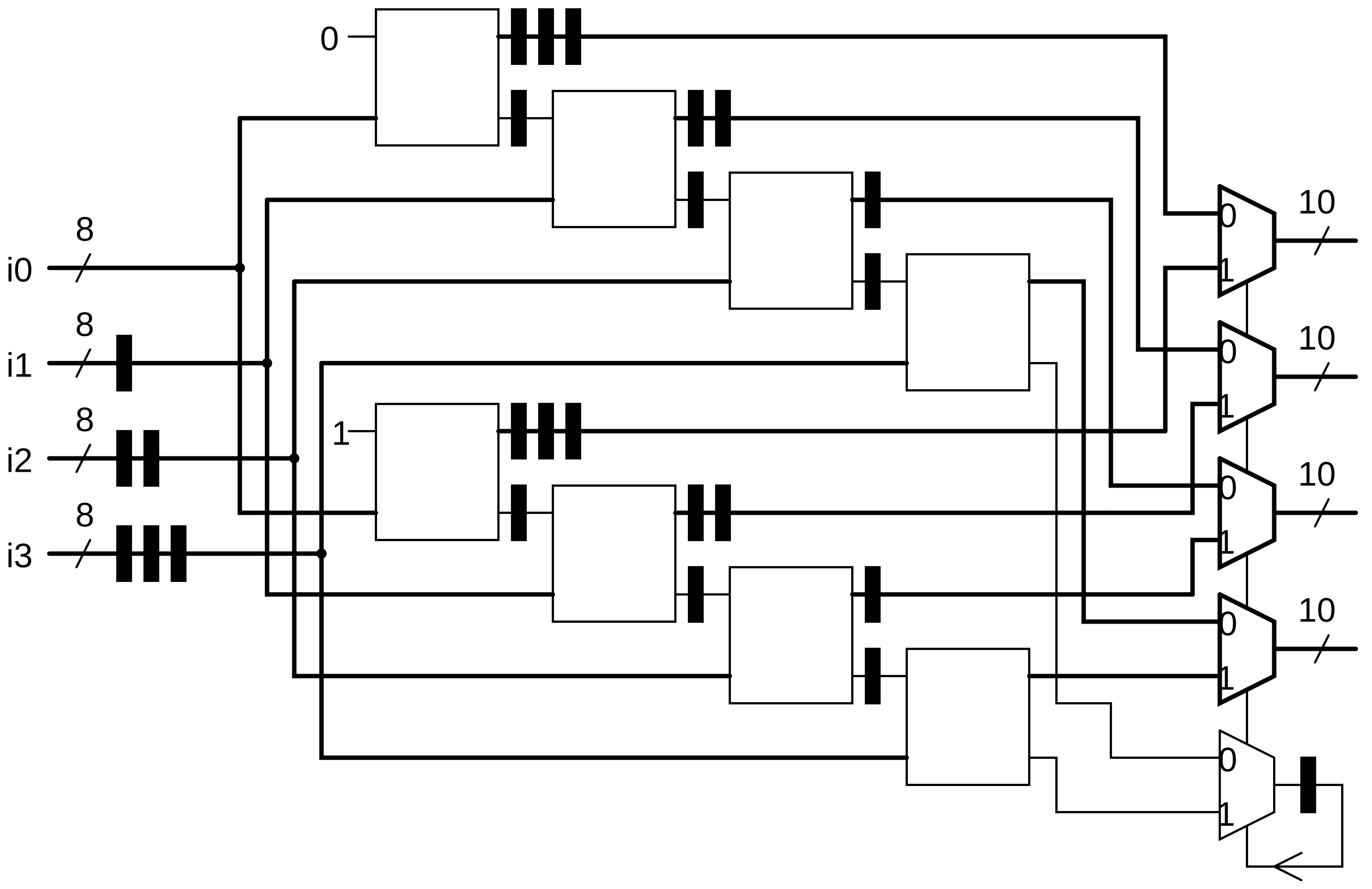
Shannon Decomp. for Retiming



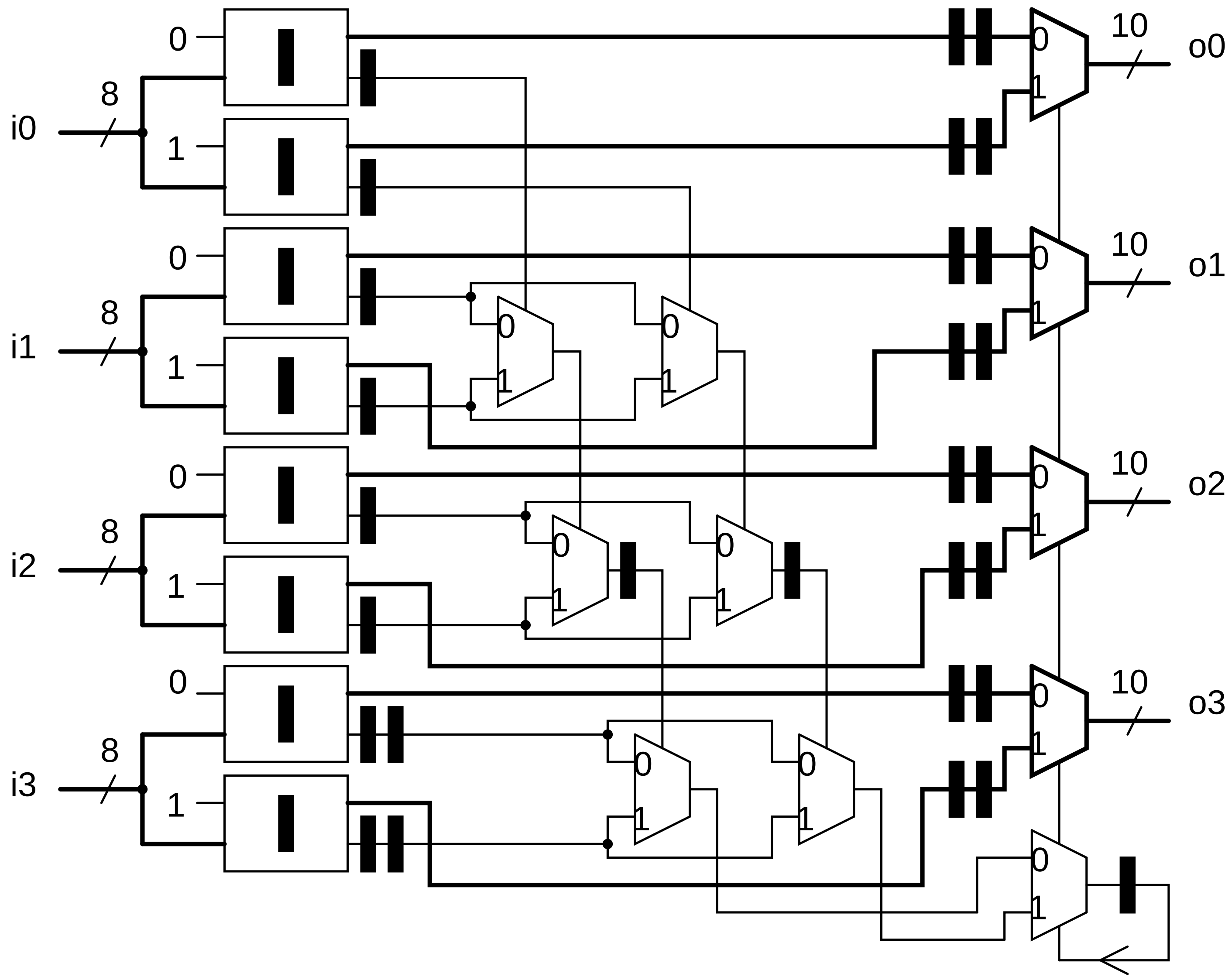
Shannon Decomposition



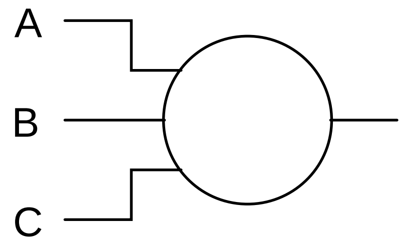
After Retiming



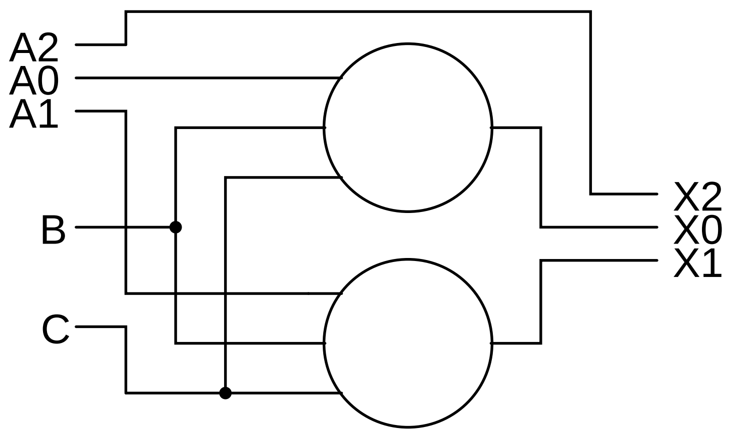
More aggressive decomposition



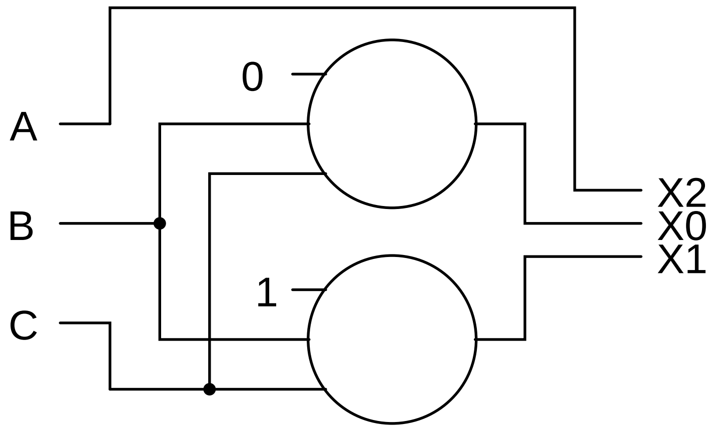
“Tech mapping” Shannon



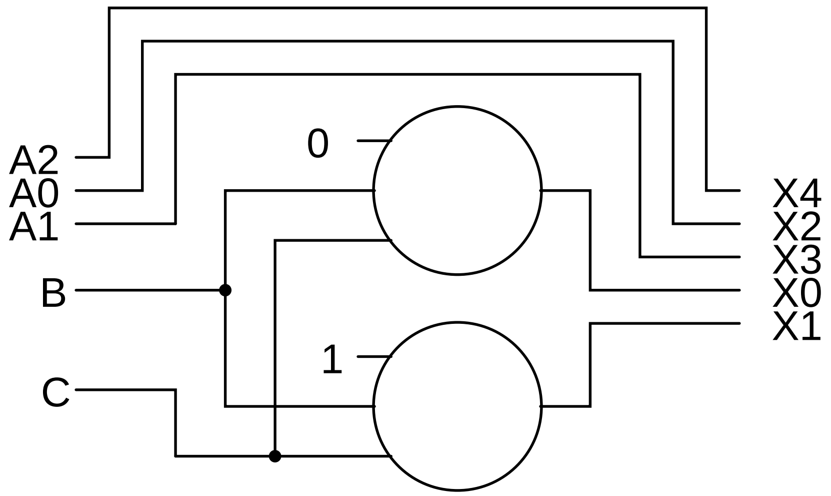
f



extend^{f}_{1}

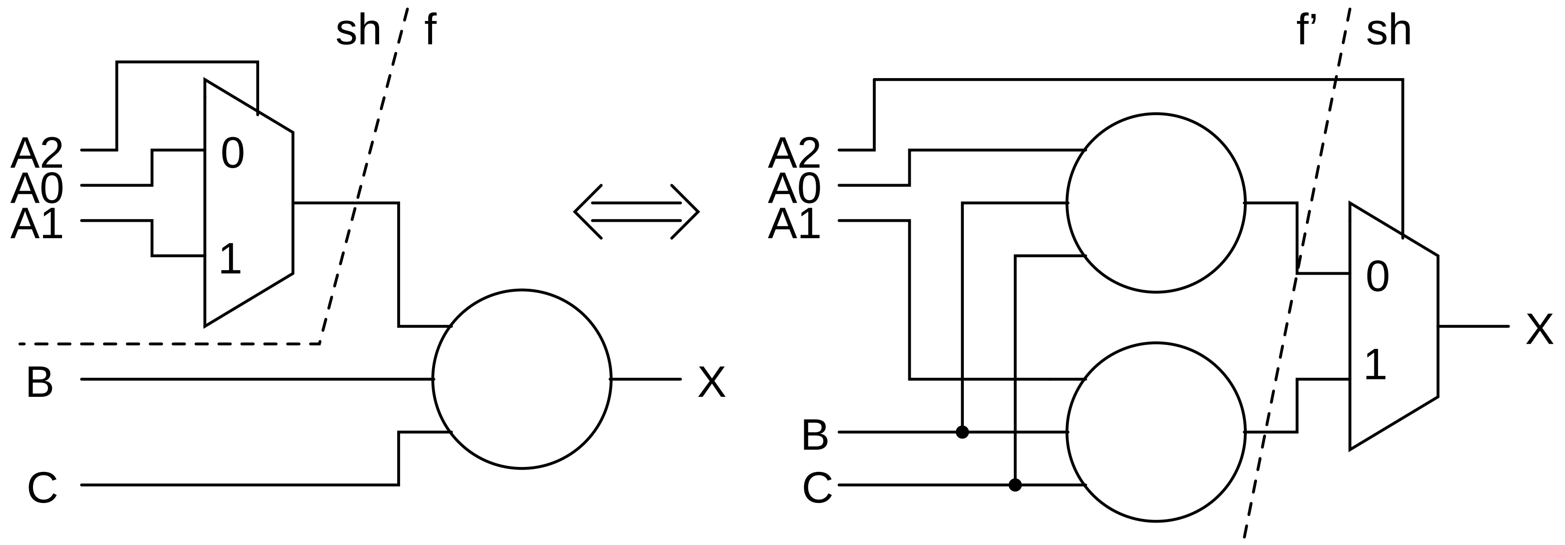


start^{f}_{0}



start^{f}_{1}

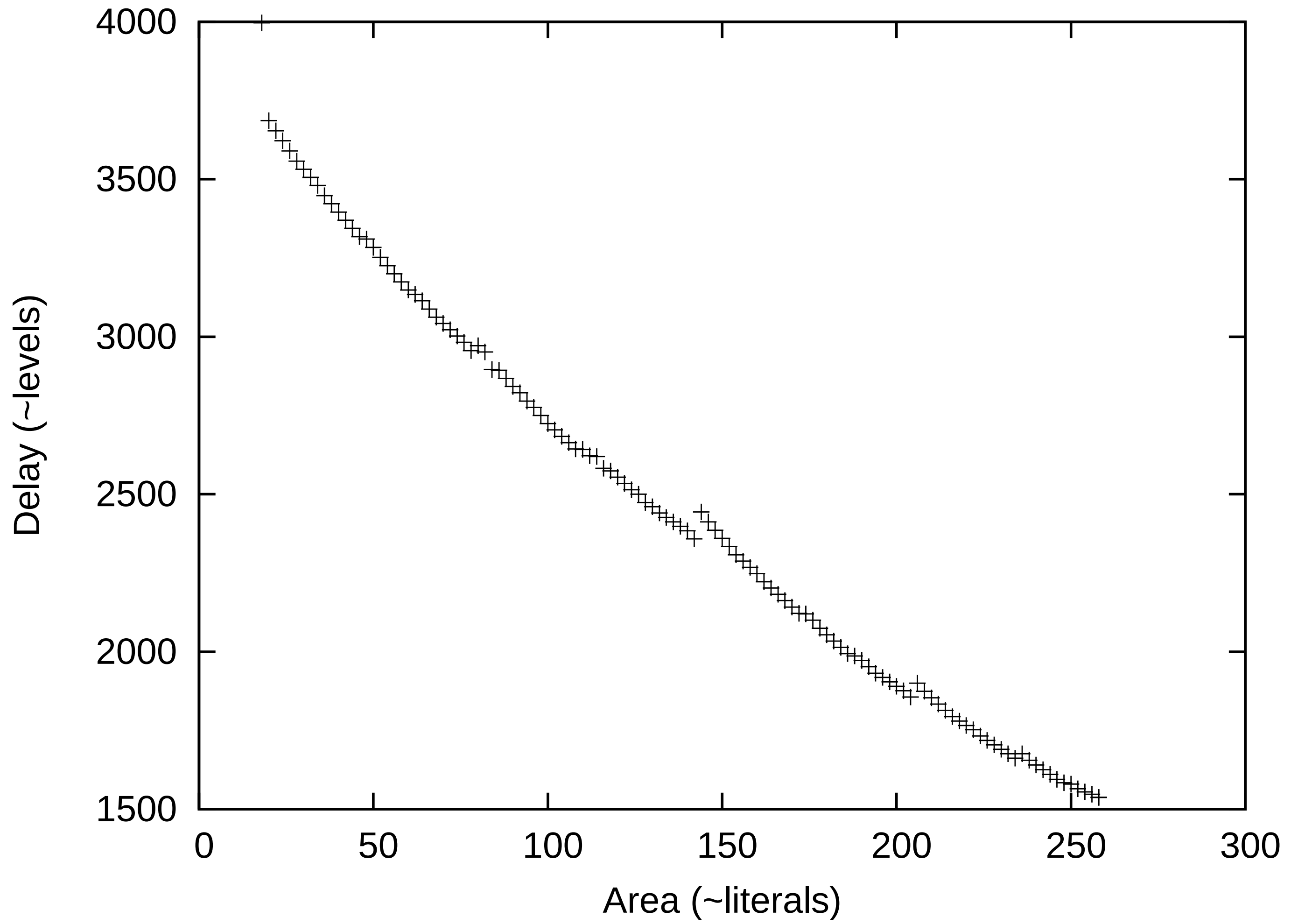
“Tech mapping” Shannon



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

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

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

Publications 1

Stephen A. Edwards.
SHIM: A Language for Hardware/Software Integration.
In *Proceedings of Synchronous Languages, Applications, and Programming (SLAP)*, Edinburgh, Scotland, April 2005.

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

Jia Zeng, Cristian Soviani, and Stephen A. Edwards.
Generating Fast Code from Concurrent Program Dependence Graphs.
In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, Washington, DC, June 2004.

Publications 2

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.