Improving Controller Synthesis from Esterel

Cristian Soviani Jia Zeng Stephen A. Edwards

Department of Computer Science,
Columbia University

www.cs.columbia.edu/~{soviani,jia,sedwards}

{soviani,jia,sedwards}@cs.columbia.edu

Why controllers ?

Several state machines drive the bulk logic (data paths)

Small area. Delay is critical.

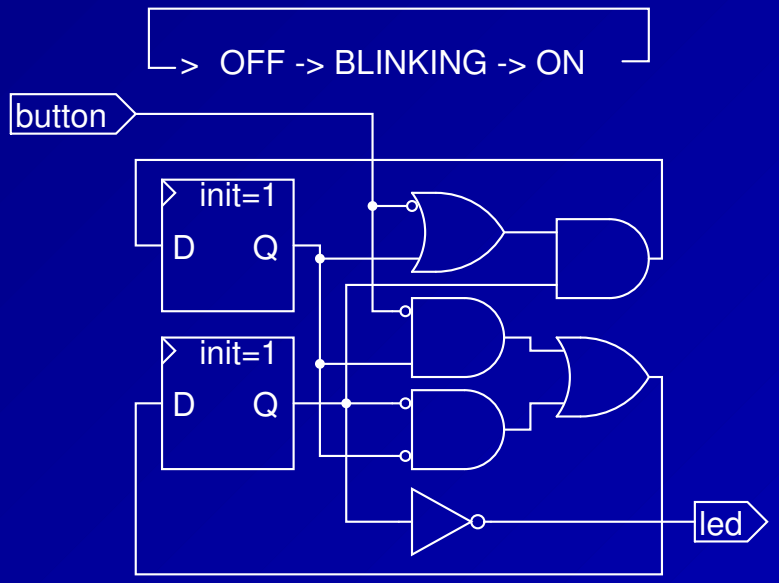
Even a simple “behaviour” leads to infernal RTL

Most bugs are here. Verification is critical.

Typical applications:

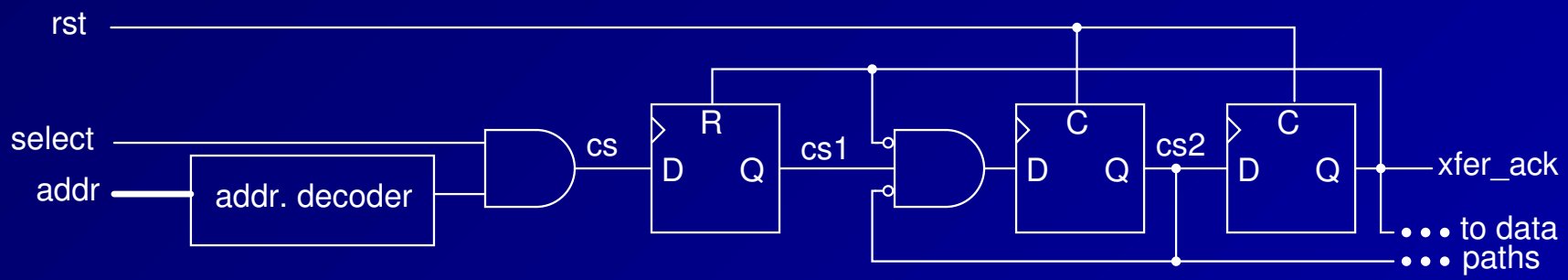
various device controllers (e.g. Eth. MAC)
bus interfaces & arbiters
scheduling pipelined units

Is it correct ? ——-



A delicate compromise to avoid

Corectness vs. Performance



from Xilinx, EDK3.1 docs, Designing Custom OPB Slave Peripherals for Microblaze

A master who assumed control of the bus may terminate, or abort, the transfer at any time by deasserting **select**. All slaves are required to terminate the transfer in progress and reset their state machines if the **select** signal is deactivated ... if the **select** is deactivated in the cycle in which the slave would have activated **xferAck**, then the slave must deactivate the **xferAck** signal in this cycle

from IBM, On-Chip Peripheral Bus, Architecture Specifications, v2.1

Simplified OPB SSRAM controller

```
 1 module opb_ram_ctrl:
 2 input SEL, RNW, A3, A2, A1, A0;
 3 output XFER_ACK;
 4 output OREG_CE, OREG_RES;
 5 output MEM_RD, MEM_WR;
 6 loop
 7  await [ SEL and A3 and ... ];
 8  abort
 9    pause;
10    present RNW then
11      emit MEM_RD; pause;
12      emit OREG_CE; pause;
13      emit XFER_ACK
14    else
15      emit MEM_WR; emit XFER_ACK
16    end
17  when [ not SEL ];
18  emit OREG_RES;
19 end loop
20 end module
```

You've already seen:

- reads require more cycles that writes
- deasserting select aborts the operation
- I included no comments on purpose

Easy to modify. Try:

- removing 8 and 17
- removing 9
- adding "pauses" between 11 and 12

I wrote the sample in 3'

Why Controllers in Esterel ?

What language do we want?

High level

simple to write / modify / understand : powerful sequential and concurrent flavored constructs

deterministic : we can't avoid mathematics

high level verification : this is different from simulation

EFFICIENT

Keep the abstraction near the technology

synchronous

intuitive translation

no "synthesis subset" jokes

Esterel is a good candidate

Previsious work. Our Results

Esterel technologies : Esterel v5, IC, one-hot enc.

D. Potop : GRC, hierarchical enc.

Primary target: s/w.

H/W synthesis relies on generic seq. optimization (sis/blifopt)

CEC : Columbia Esterel Compiler

Challenge : use high level info

CEC generates corect & efficient circuits

To do: improve the circuit delay

Surface & Depth - Termination levels

```
input A, B, C, D;          term. level   s/d
output X, Y, Z;

trap T in
 trap U, V in
     present A then pause end#        0,1/0
     ||
     present B then exit T end#       0,3/0
     ||
     pause;
     present C then exit U end#       1/0,2
     ||
     present D then exit V else      1,2/0,1
       pause; pause#
     end
 handle U do emit X
 handle V do emit Y
 end trap
handle T do emit Z
end trap
```

Surface : hard start
Depth : continue

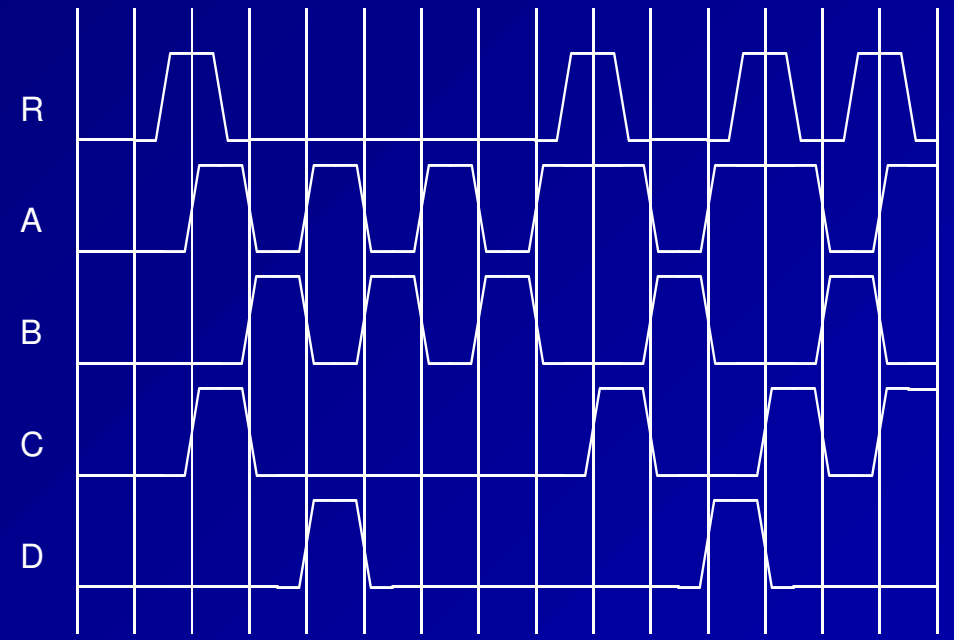
Term levels:
0 : terminated
1 : still running
2,3 ... exceptions

The biggest level wins

Sample Esterel code

```
module example:
input R;
output A, B, C, D;

every R do
  loop
    emit A;
    pause;
    emit B;
    pause
  end loop
||
  emit C;
  pause;
  pause;
  emit D
end every

end module
```

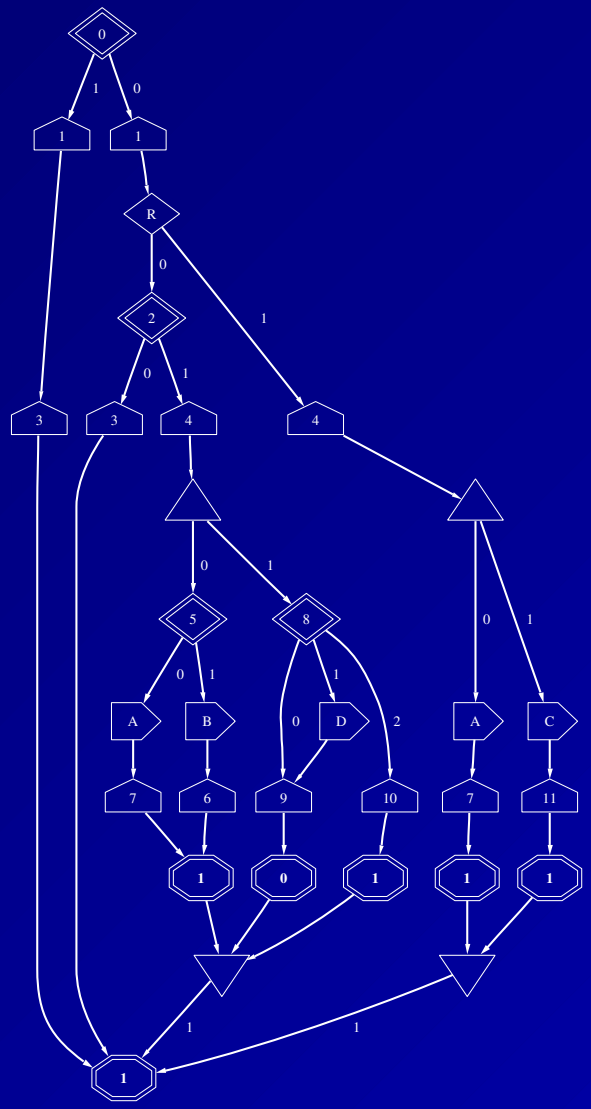
Sample timing diagram

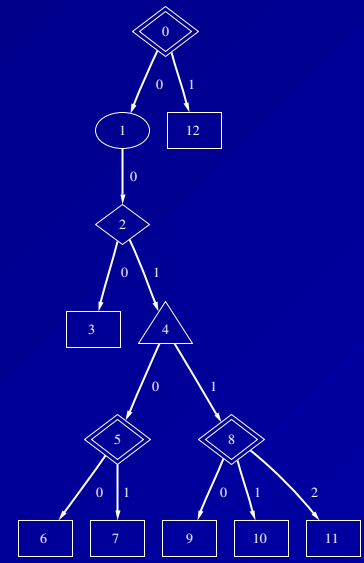


Note the “strong” priority of “every R” which aborts the curent instructions and immediately restarts its body

The CFG and ST

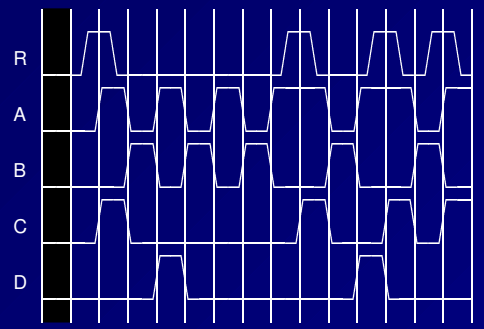
```
module example:
input R;
output A, B, C, D;

every R do
  loop
    emit A;
    pause;
    emit B;
    pause
  end loop
||
  emit C;
  pause;
  pause;
  emit D
end every

end module
```



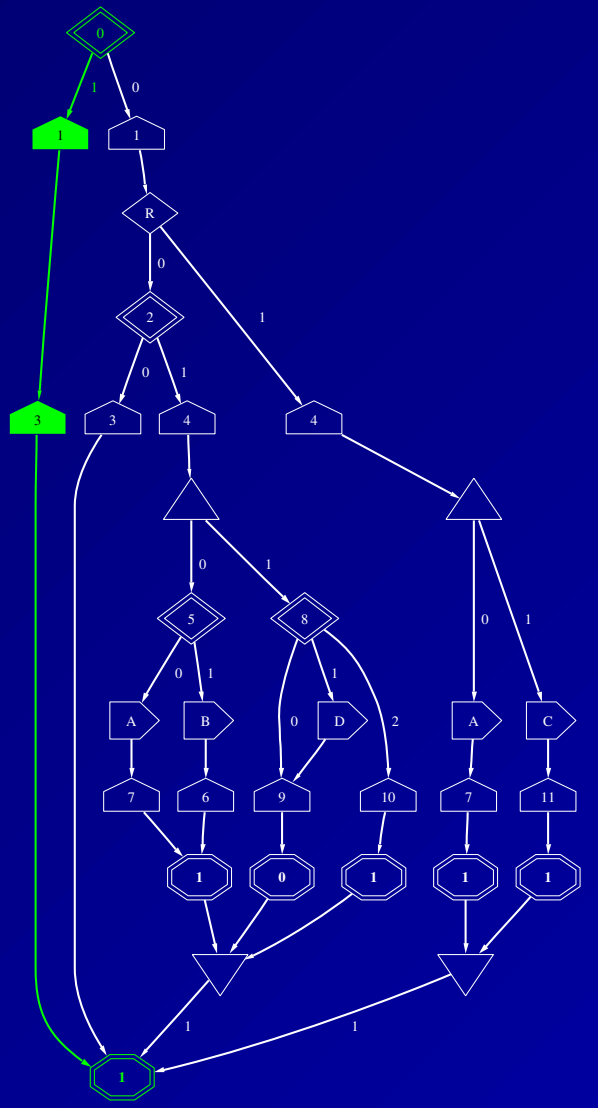


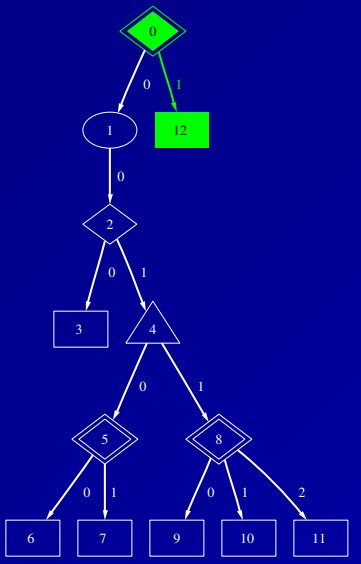
The Control Flow Ghaph (left) and the Selection Tree (above)

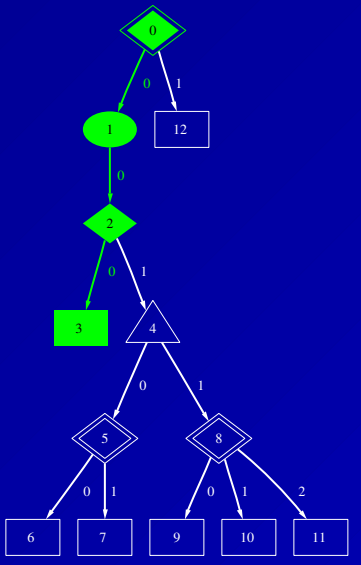
Clock 0 : R=0 A=0 B=0 C=0 D=0



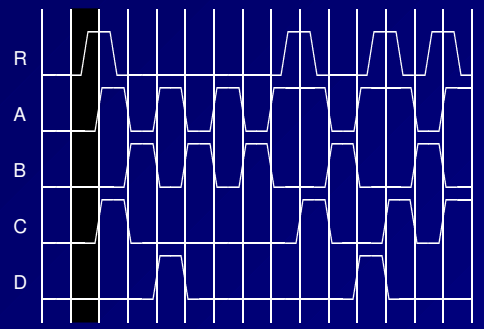

```
every R do
   loop
      emit A;
      pause;
      emit B;
      pause
   end loop
||
   emit C;
   pause;
   pause;
   emit D
end every
```



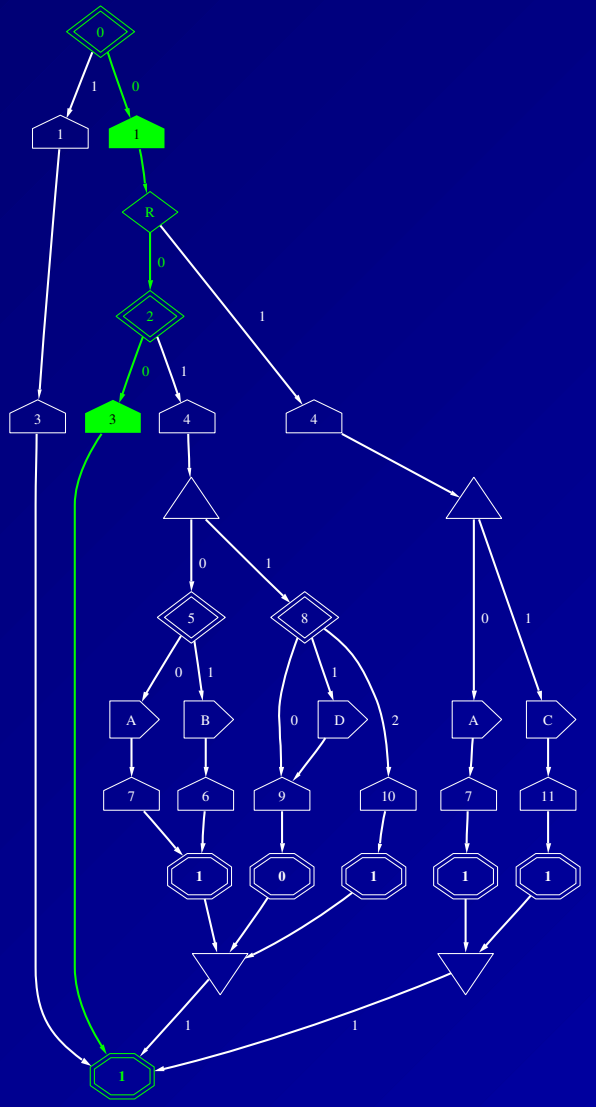


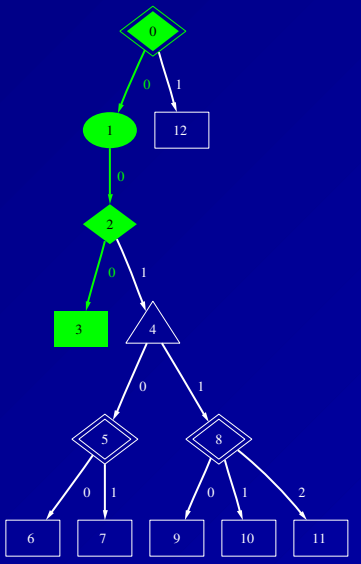


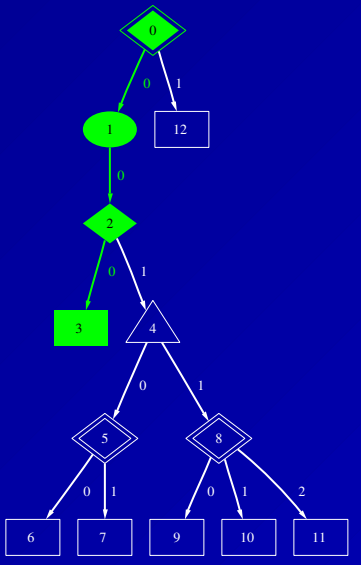
Clock 1 : R=0 A=0 B=0 C=0 D=0



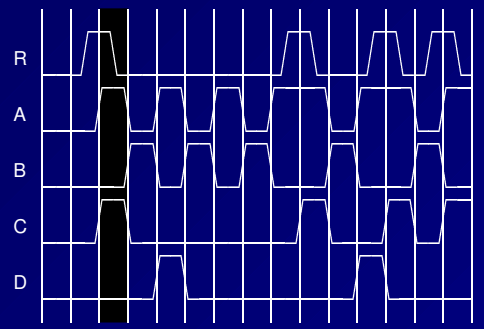

```
every R do
   loop
      emit A;
      pause;
      emit B;
      pause
   end loop
||
   emit C;
   pause;
   pause;
   emit D
end every
```



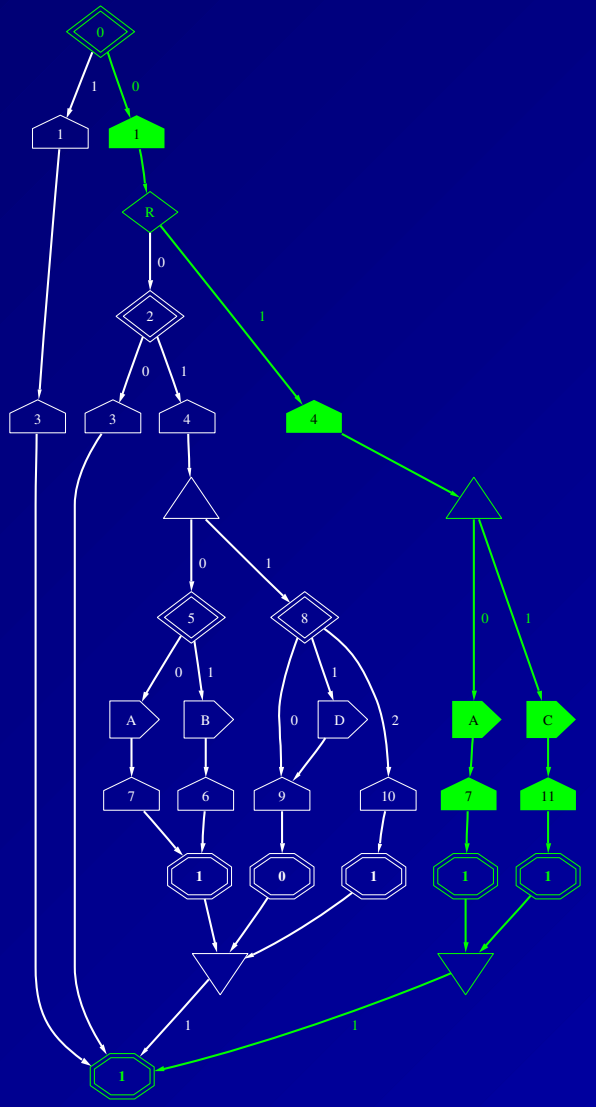


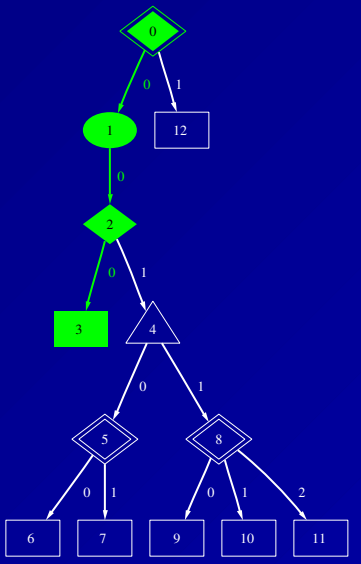


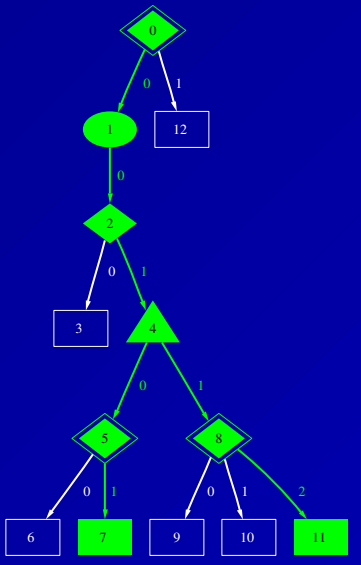
Clock 2 : R=1 A=1 B=0 C=1 D=0



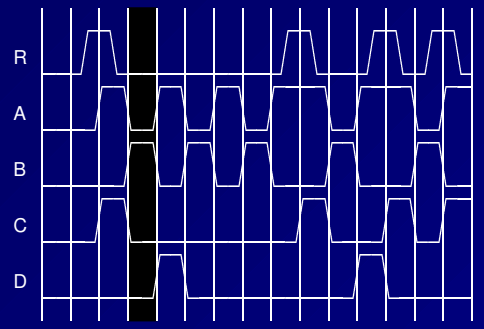

```
every R do
   loop
      emit A;
      pause;
      emit B;
      pause
   end loop
||
   emit C;
   pause;
   pause;
   emit D
end every
```



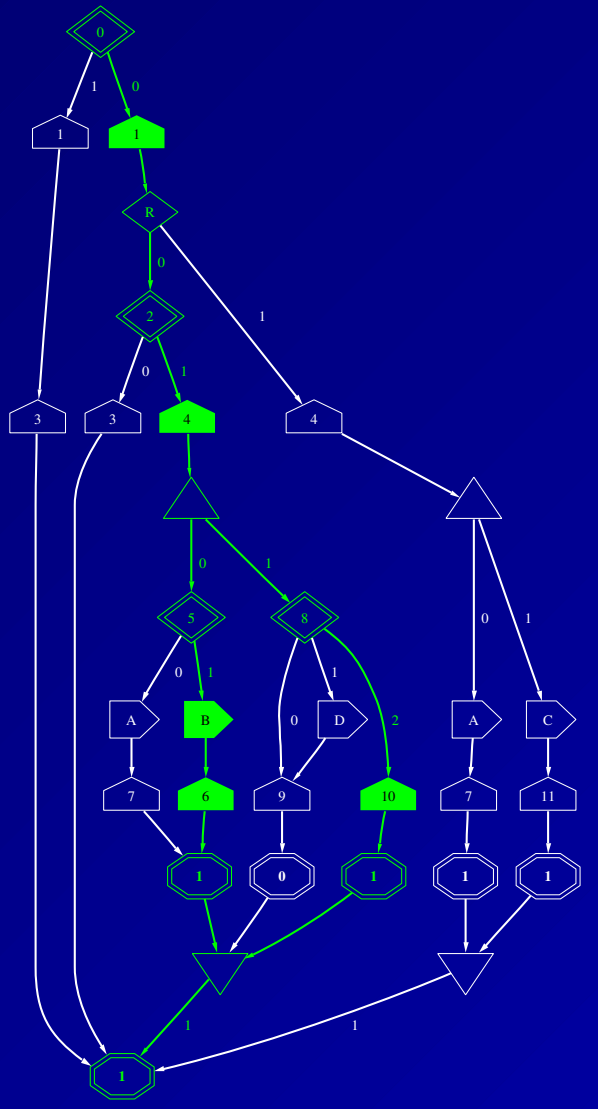


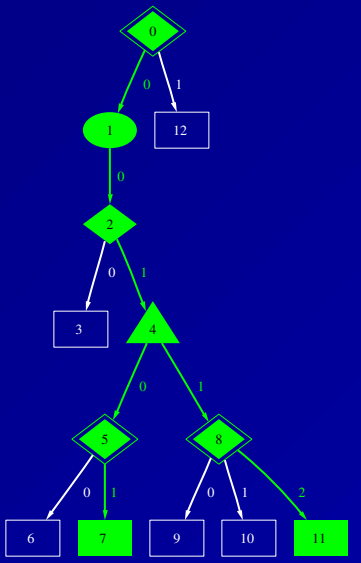


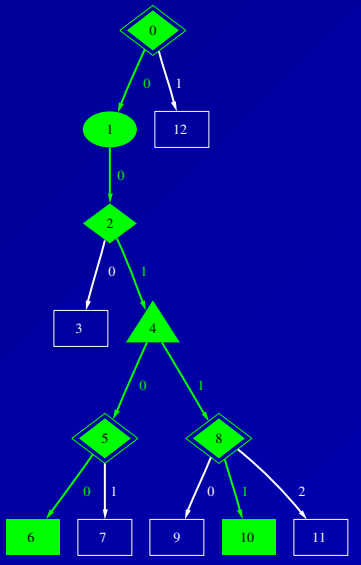
Clock 3 : R=0 A=0 B=1 C=0 D=0



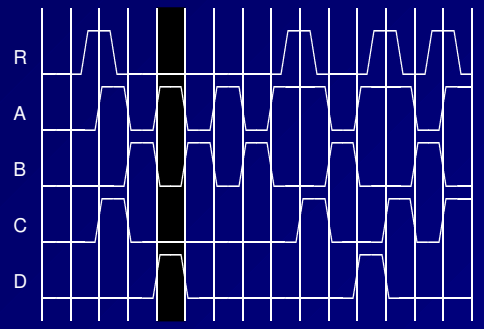

```
every R do
   loop
      emit A;
      pause;
      emit B;
      pause
   end loop
||
   emit C;
   pause;
   pause;
   emit D
end every
```



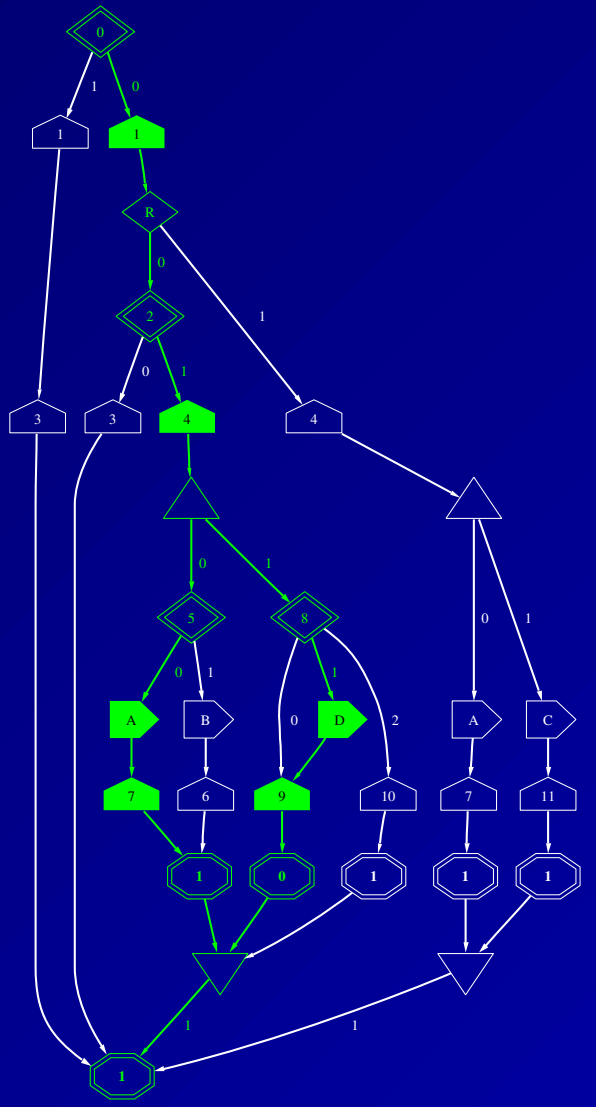


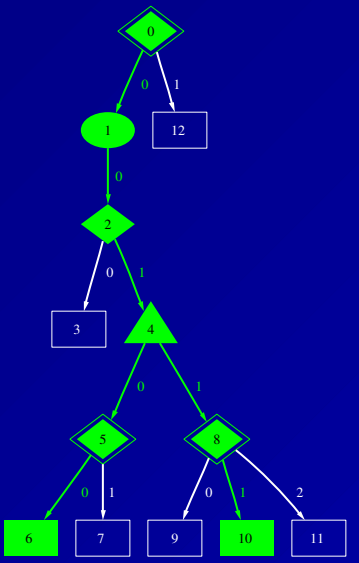


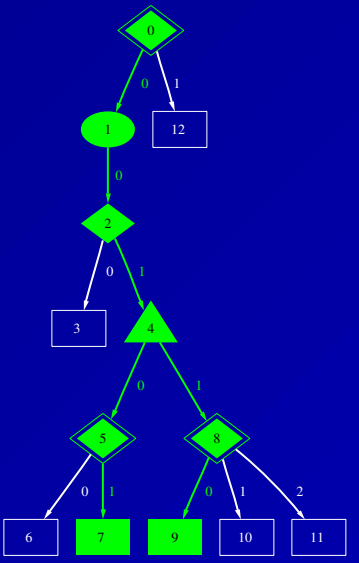
Clock 4 : R=0 A=1 B=0 C=0 D=1



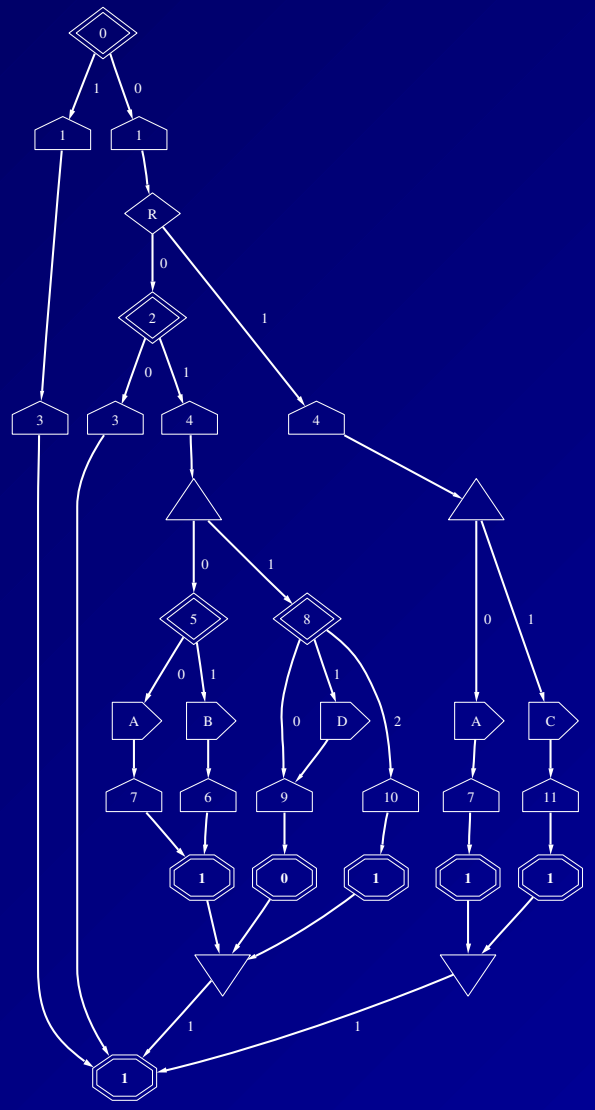

```
every R do
   loop
      emit A;
      pause;
      emit B;
      pause
   end loop
||
   emit C;
   pause;
   pause;
   emit D
end every
```

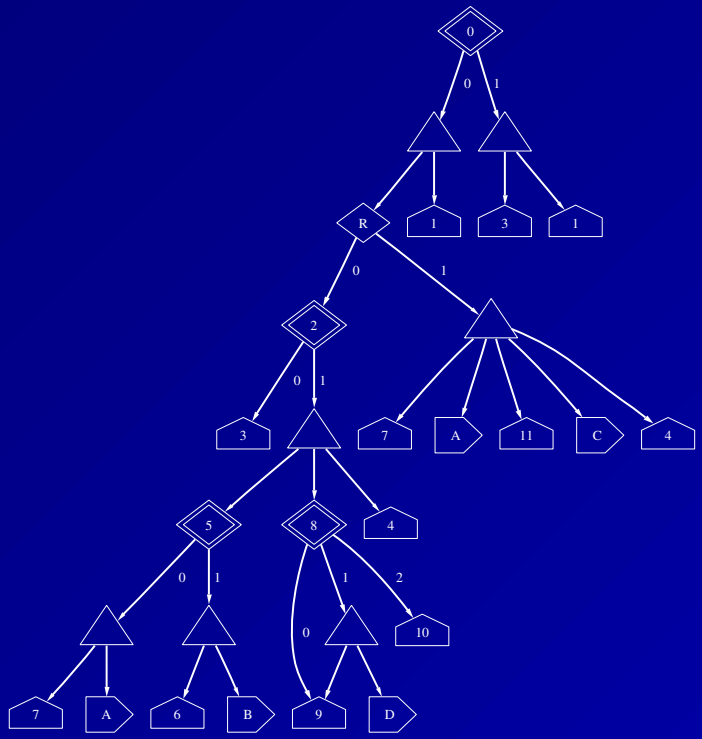






The CFG and PDG





The PDG (above) is a more concurrent representation of the CFG (left)

Note that nodes 4, A, 7, C, 11 (on the right side of the picture) have the same flow control

Reincarnation. Schizophrenia

```
module reincarnation:
input A;
output X,Y;

loop
  signal S in
    trap T in
      present A then pause; emit S end;
      present S then emit X else emit Y end;
    ||
      pause;
      exit T
    end trap
  end signal
end loop

end module
```

Can X and Y be both emitted in the same cycle?

`signal S`

declares a local signal w/ default value 0

Consequence : because of the loop, S can have 2 values in the same cycle

Some instructions can execute several times in the same cycle: different inputs - different results

Dementia praecox

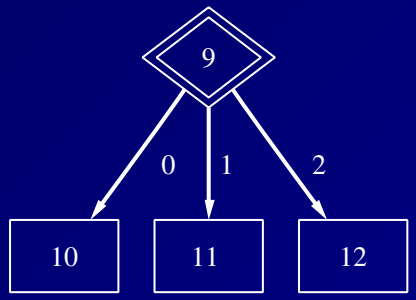
YES, they can

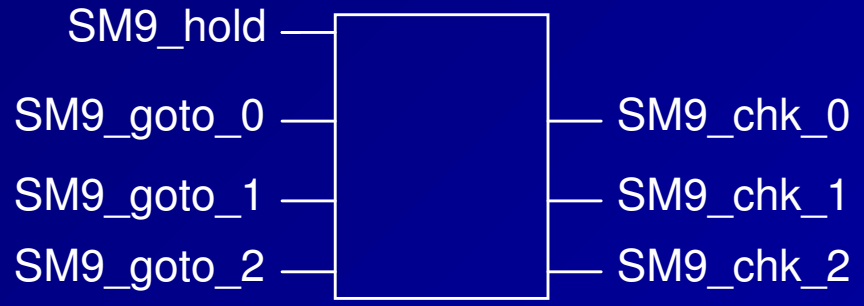
Token Ring. Causality

```
module token_ring: output HELLOA, HELLOB, HELLOC;
signal TKAB, TKBC, TKCA in
emit TKCA
|| run host [ signal HELLOA/HELLO, TKCA/TKIN, TKAB/TKOUT ; constant 2/N]
|| run host [ signal HELLOB/HELLO, TKAB/TKIN, TKBC/TKOUT ; constant 3/N]
|| run host [ signal HELLOC/HELLO, TKBC/TKIN, TKCA/TKOUT ; constant 4/N]
end signal end module

module host: constant N : integer; input TKIN; output TKOUT, HELLO;
signal REQ, ACK, GIFT in
loop
  present [ TKIN and not GIFT ] then pause else await TKIN; emit GIFT end;
  present REQ then emit ACK; pause end;
  emit TKOUT
end
||
loop
  await N tick;
  weak abort sustain REQ when immediate ACK;
  emit HELLO
end loop end signal end module
```

Blackbox State Machines

For each Exclusive node in the ST we build a state machine





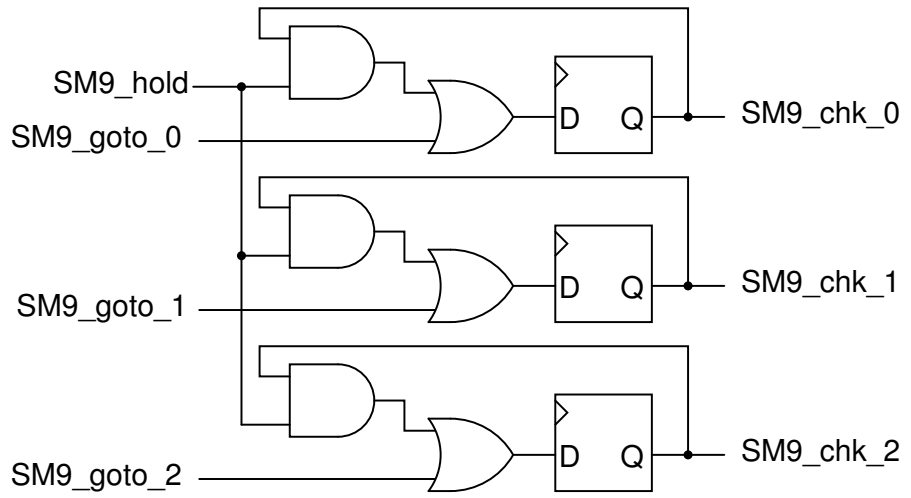
The “scaffolding” combinational logic is synthesised by translating the PDG

The blackbox state machines are synthesised using a given encoding for each one (heuristicaly determined, can be manually overwritten)

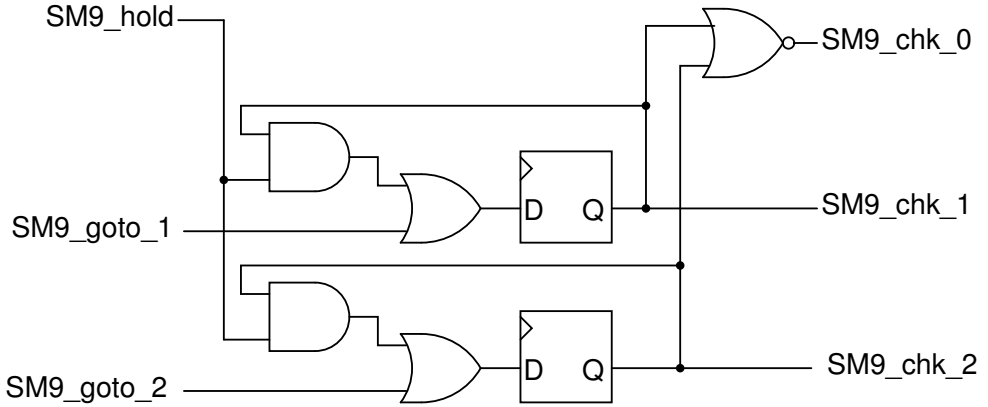
Different encodings

	Q2	Q1	Q0
S0	-	-	1
S1	-	1	-
S2	1	-	-

one hot encoding



	Q1	Q0
S0	0	0
S1	-	1
S2	1	-

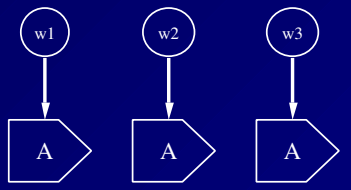


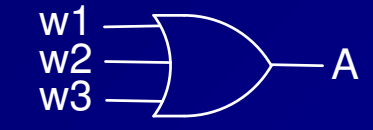
The inputs SMgoto and SMhold are assumed mutually exclusive

SMhold is used for the Suspend instruction (a powerful construct similar to UNIX Ctrl-Z)

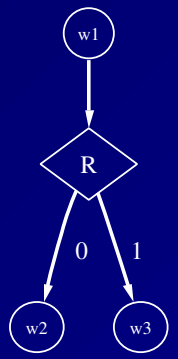
The generation of these signals is not trivial and requires some kind of priority arbitration
(see Enter and Suspend translation in the next slides)

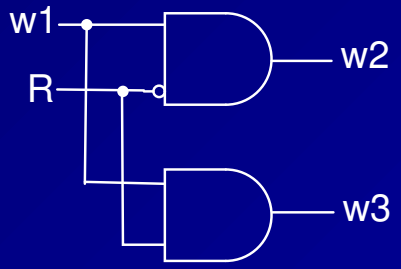
Translation of Emit, Test and Fork



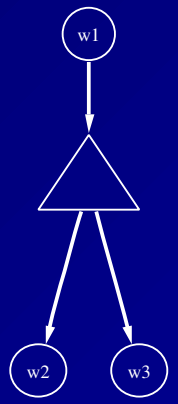


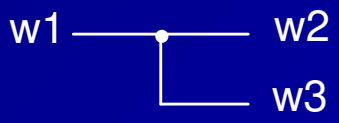
Emit





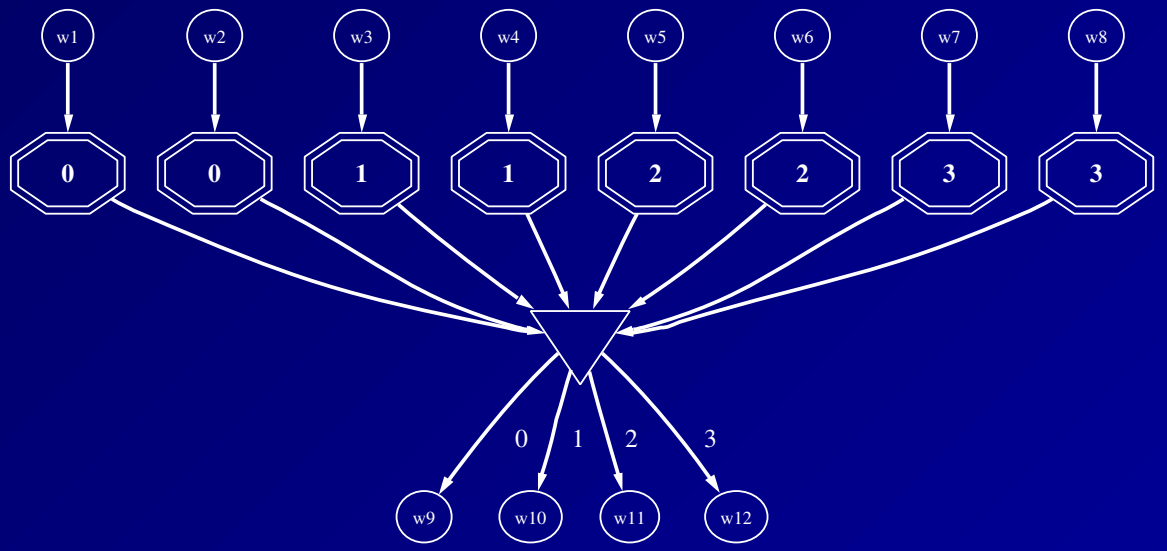
Test

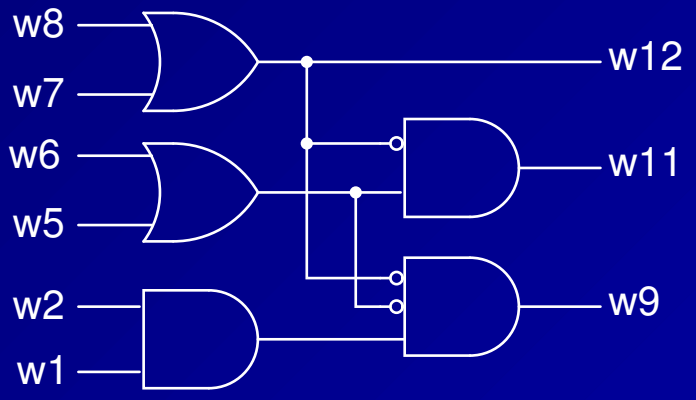




Fork

Translation of Sync



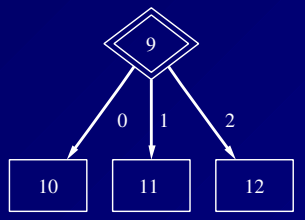


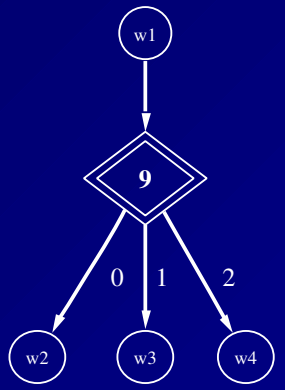
Note: w3, w4, w10 are not used

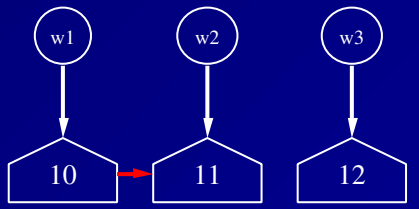
The Sync node computes the maximum termination level of all its threads

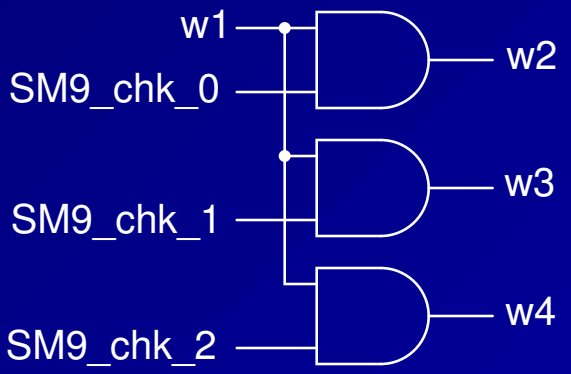
The translation is mainly a priority decoder
Termination level 1 is specially handled

Translation of Switch and Enter

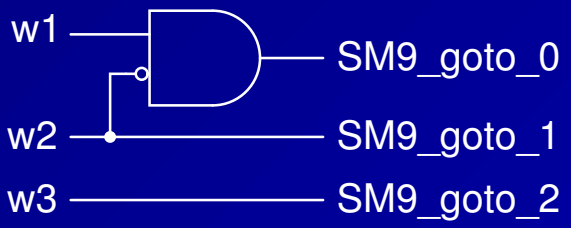








Switch



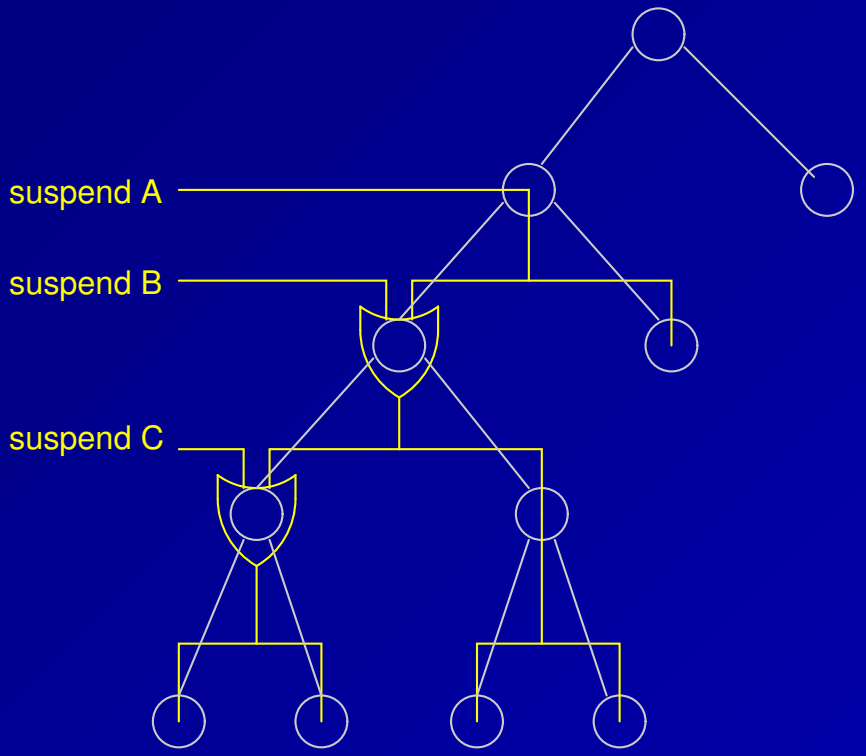
Enter

Translation of Suspend

```
suspend
  loop
    emit VECT_ADD
    pause;
    emit VECT_MUL;
    pause
  end loop
when [ not RDY ]
```

Intuitively works like UNIX Ctrl-Z. If RDY is not present, the Suspend body is “frozen”. The execution resumes when RDY is asserted.

Suspend instructions can be nested. We build a “suspend” OR net on the ST structure. This net drives the SMhold signals.

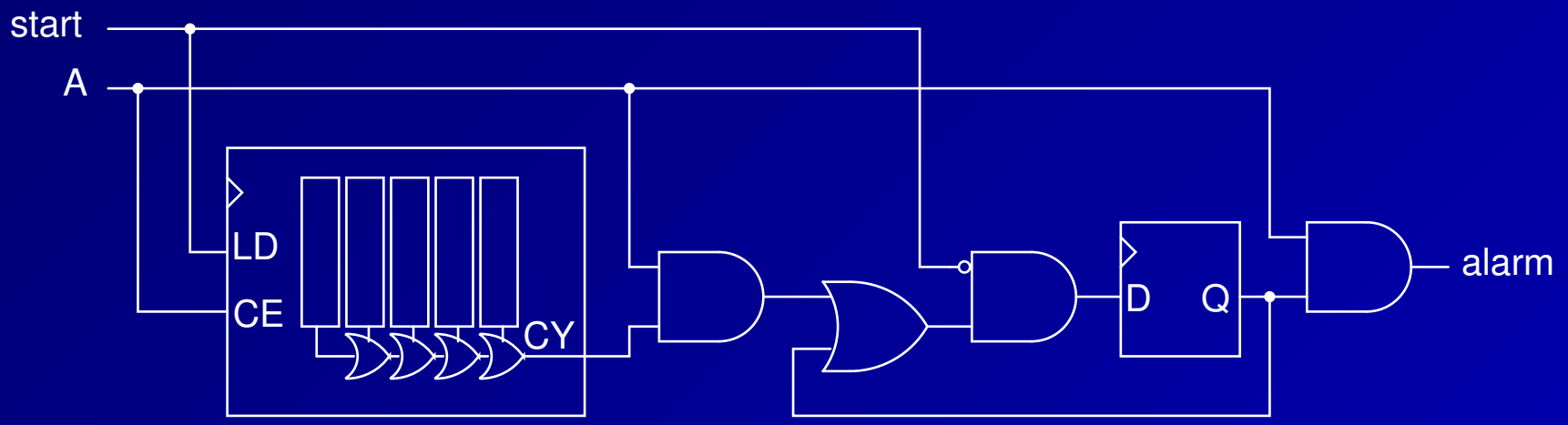


Translation of Counters

```
abort
 run Handshake
when
  case [ 20 tick ] do
    emit TIMEOUT
end abort
```

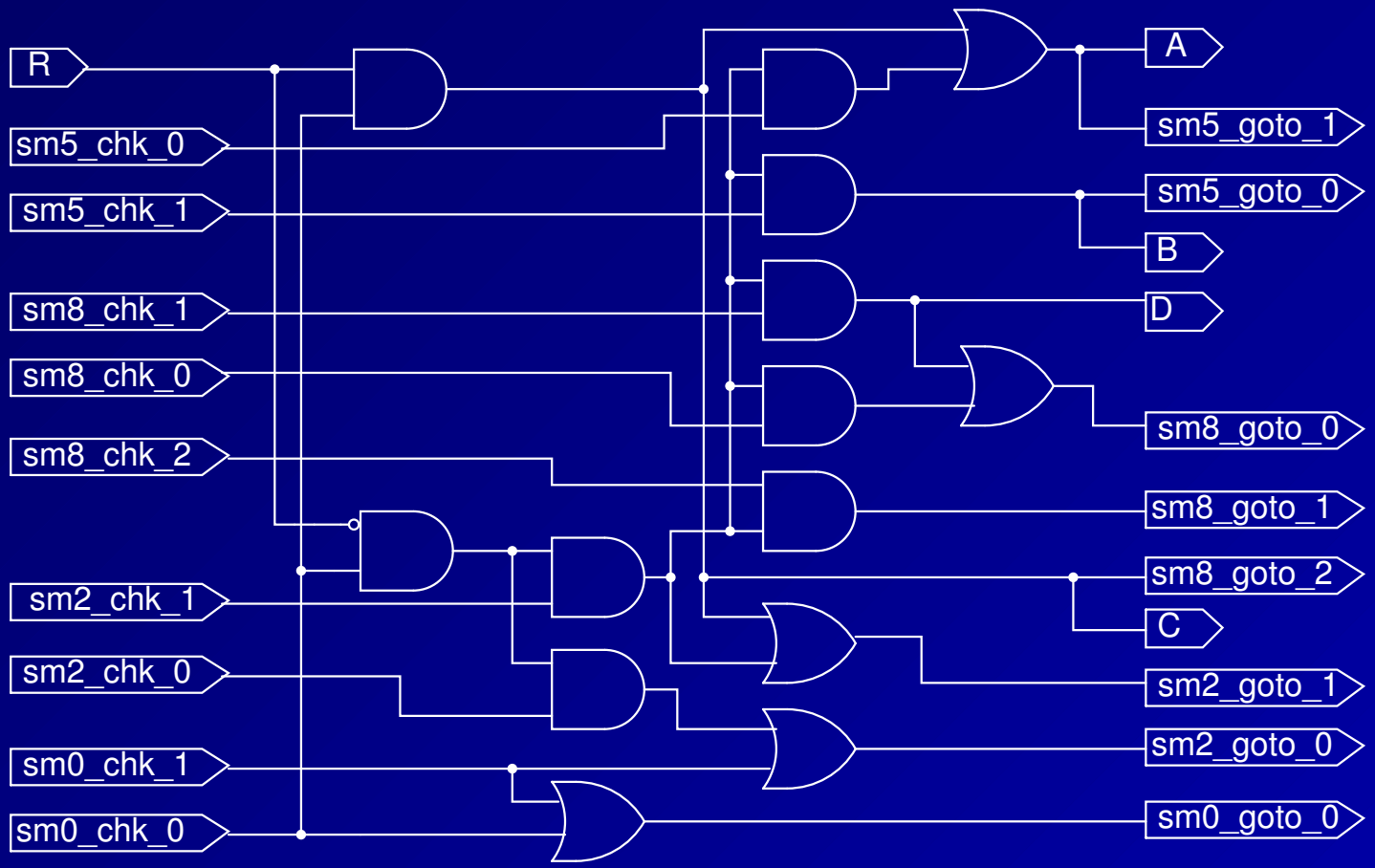
This sample runs the Handshake module. If the handshaking is not finished in 20 clocks, it is aborted and the TIMEOUT signal is asserted

Counted predicates are a very useful Esterel construct

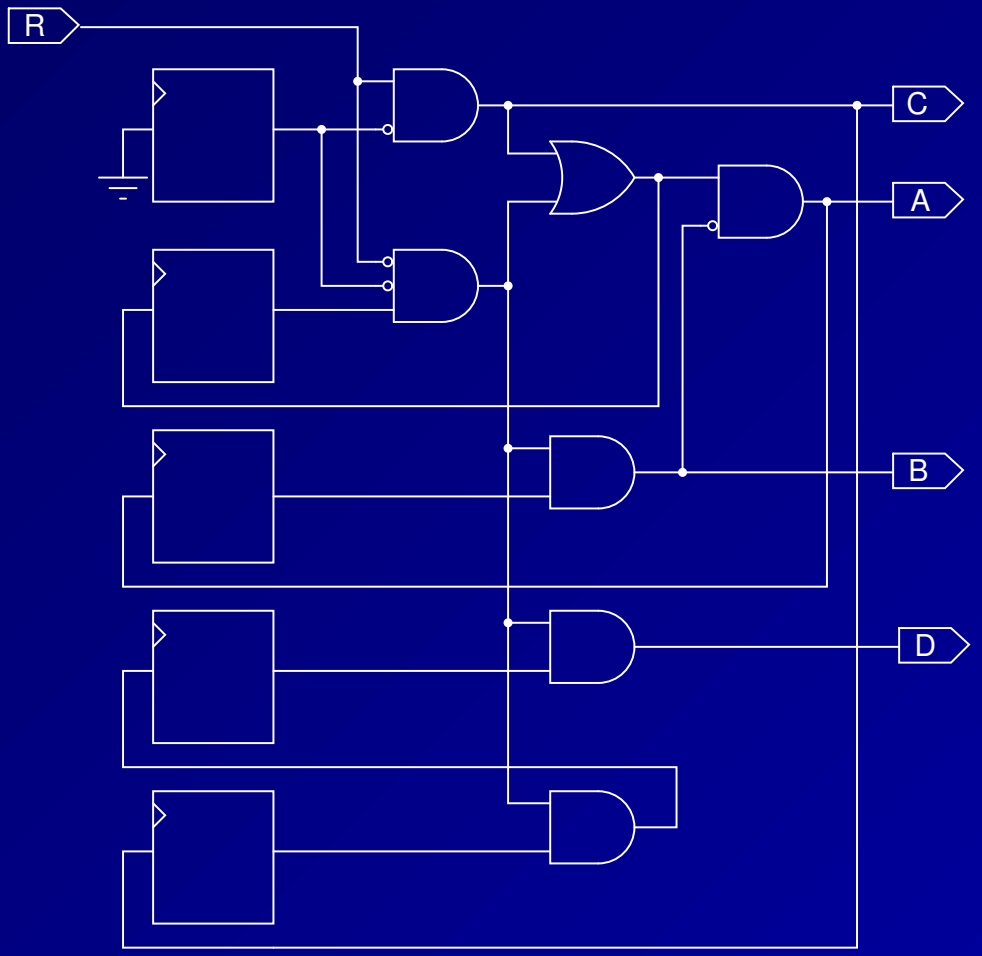


The last state before alarm is one hot encoded

The generated scaffolding circuit



The final SIS - optimized circuit



SIS : script.rugged

Xilinx XC2V2000-ff896-4 FPGA has 4 input LUTS
The circuit will be 1 level : 2ns period

For comparision, a 16 bit adder (registered I/O) has a 5.3 ns period

Last slide

Questions ?

Suggestions ?

CEC-0.2 can be downloaded at

`landc.cs.columbia.edu/projects.html`

Feel free to play. We are waiting for feedback.