The Synchronous Model of Computation

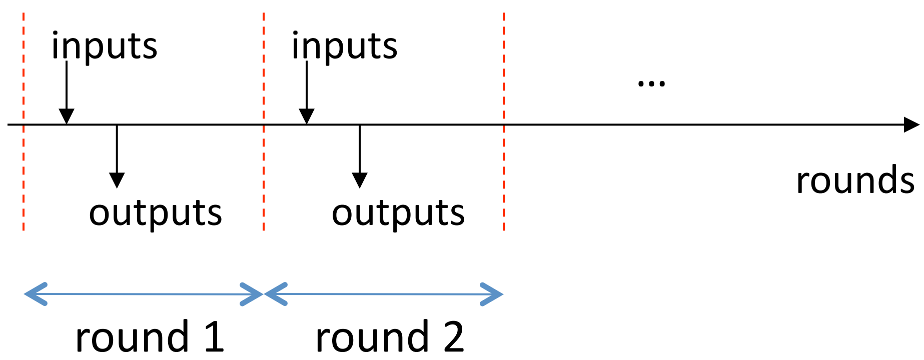
Stavros Tripakis
UC Berkeley

EE 249 Lecture – Sep 15, 2009

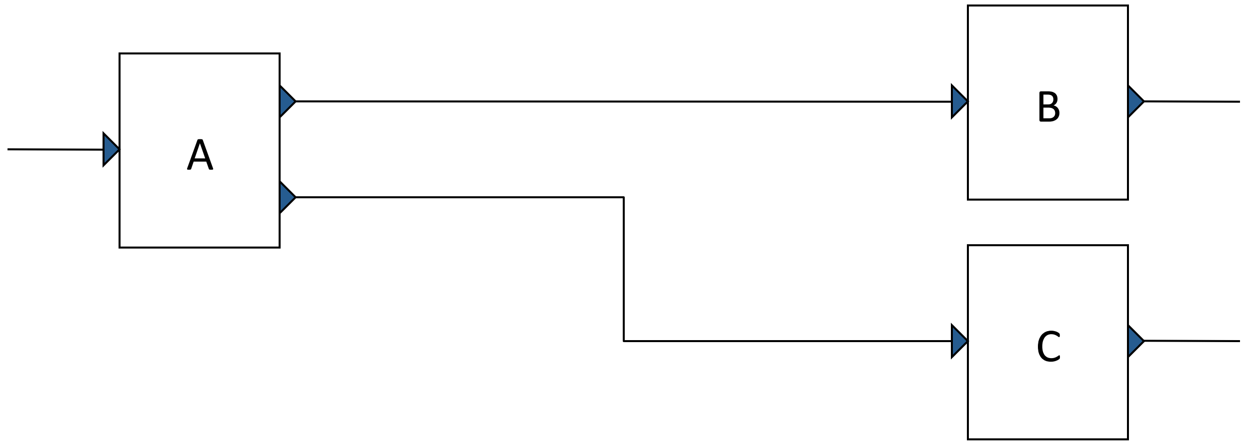
Fundamental characteristics of the synchronous MoC

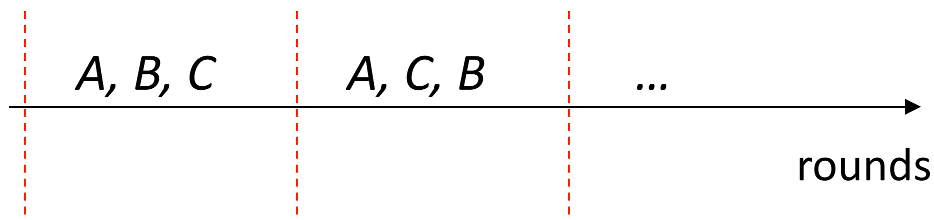
- Notion of ***synchronous round*** (or ***cycle***)
- Concurrency
- Determinism (most of the time)
  - Same (sequence of) inputs => same (sequence of) outputs
- Contrast this to:
  - Concurrency with threads:
    - Non-deterministic: results depend on interleaving
  - Concurrency in Kahn Process Networks:
    - Asynchronous (interleaving), but still deterministic
    - Needs unbounded buffers in general, for communication

The synchronous round

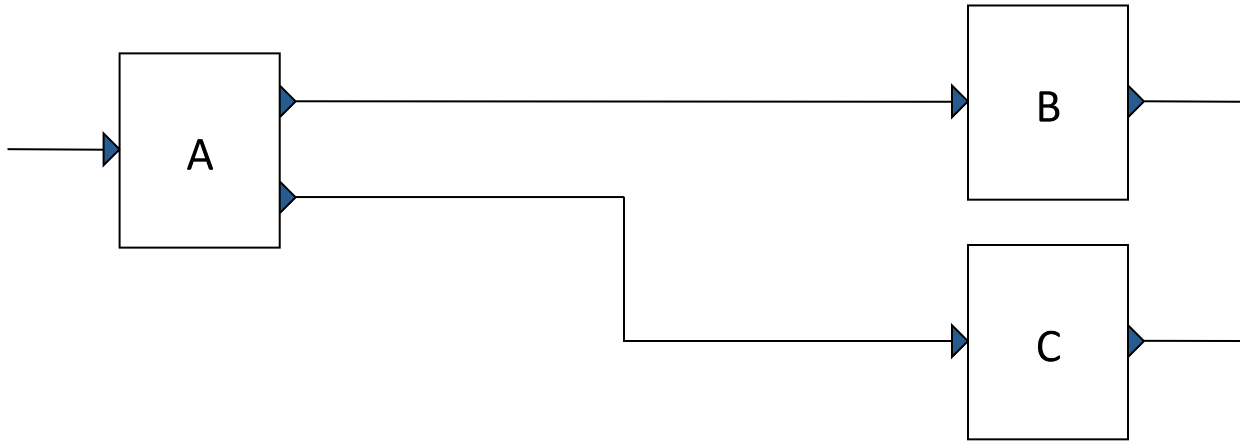


Example: synchronous block diagram

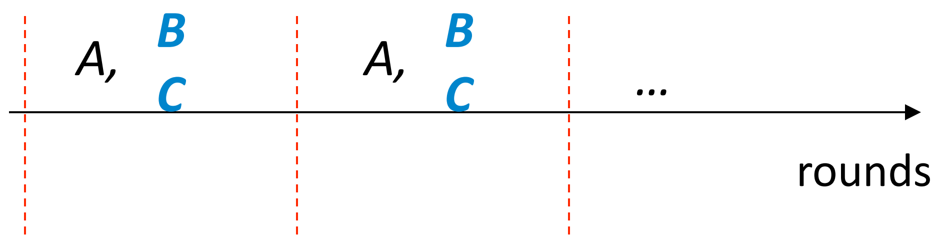




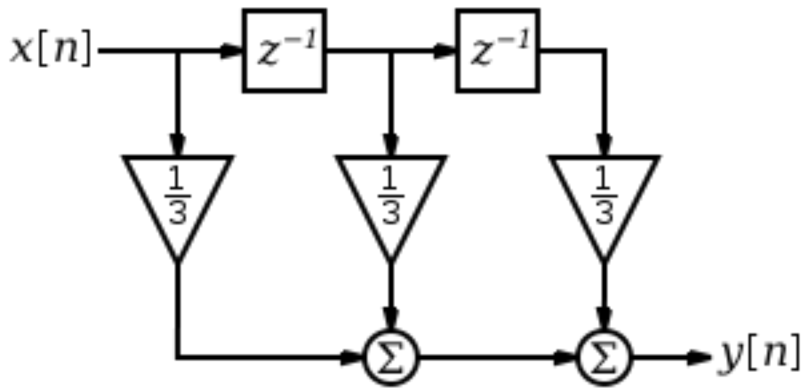
Example: synchronous block diagram



deterministic concurrency



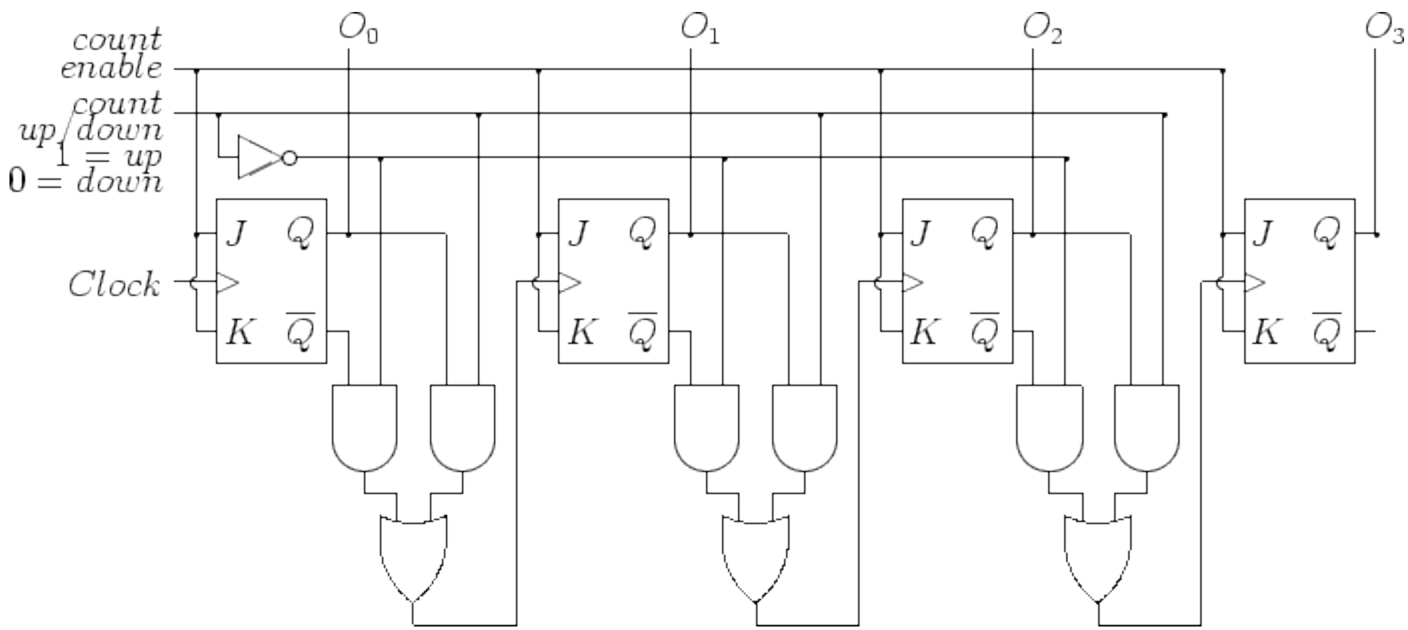
Example: FIR filter



$$y(n) = \frac{1}{3}x(n) + \frac{1}{3}x(n-1) + \frac{1}{3}x(n-2)$$

Where is the synchronous round here?

Example: sequential logic diagram



Where is the synchronous round here?

Example: control loop

```
initialize state;
while (true) do
  read inputs;
  compute outputs;
  update state;
  write outputs;
end while;
```

Where is the synchronous round here?

Example: control loop (v2)

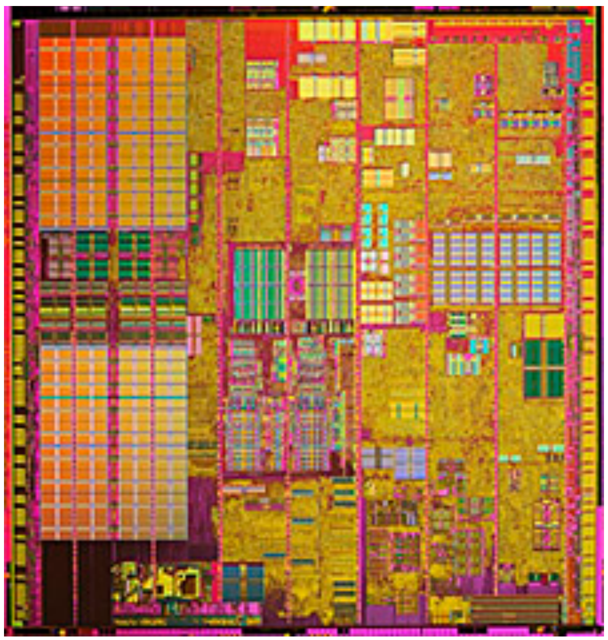
```
initialize state;
while (true) do
  await clock tick;
  read inputs;
  compute outputs;
  update state;
  write outputs;
end while;
```

Is this an important model of computation?

- Yes!
  - Extremely widespread, both in terms of models/ languages, and in terms of applications
- Examples of applications:
  - Synchronous digital circuits
  - 99% (?) of control software
    - Read-compute-write control loops
    - Nuclear, avionics, automotive, …
  - Multimedia, …

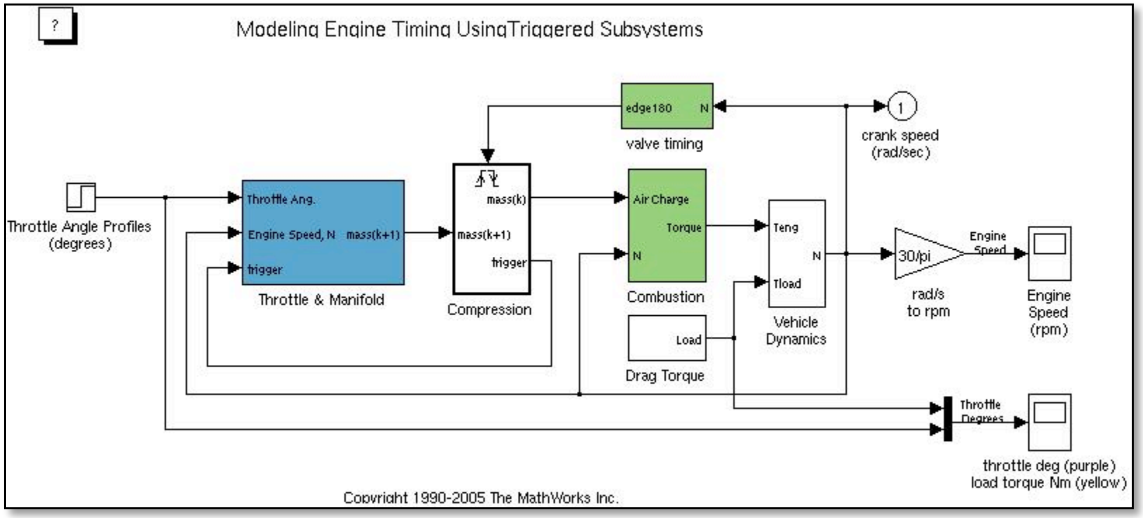
Is this an important model of computation?

HW



SW ++

c.f. Simulink to FPGA, or to HDL



Engine control model in Simulink

Copyright The Mathworks

Is this an important model of computation?

- Yes!
  - Extremely widespread, both in terms of models/languages, and in terms of applications
- Examples of models and languages:
  - Mealy/Moore machines
  - Verilog, VHDL, …
  - (discrete-time) Simulink
  - Synchronous languages
  - (Synchronous) Statecharts
  - The synchronous-reactive (SR) domain in Ptolemy II
  - …

Myths about synchronous models

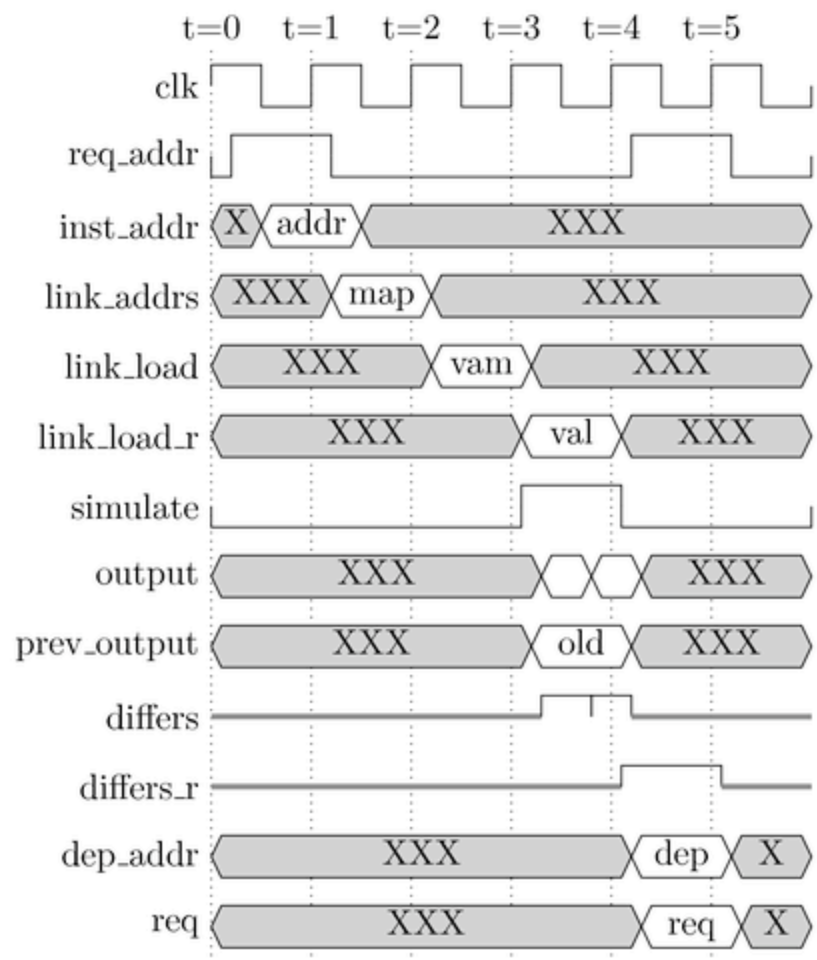
- Synchronous models have ***zero-time*** semantics
  - Synchronous semantics are essentially **untimed**: they do not have a quantitative notion of time.
  - Famous Esterel statements [Berry-Gonthier '92]:
    - `every 1000 MILLISEC do emit SEC end`
    - `every 1000 MILLIMETER do emit METER end`
  - Synchronous models can capture both **time-triggered** and **event-triggered** systems. E.g.:
    - Do something every 20ms
    - Do something whenever you receive an interrupt from the engine

Example: control loop (v3)

```
initialize state;
while (true) do
   await clock tick
     or any other interrupt;
   read inputs;
   compute outputs;
   update state;
   write outputs;
end while;
```

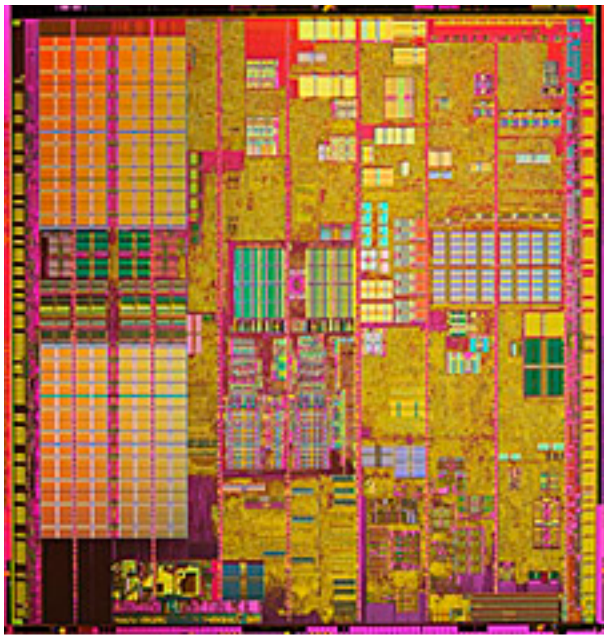
Myths about synchronous models

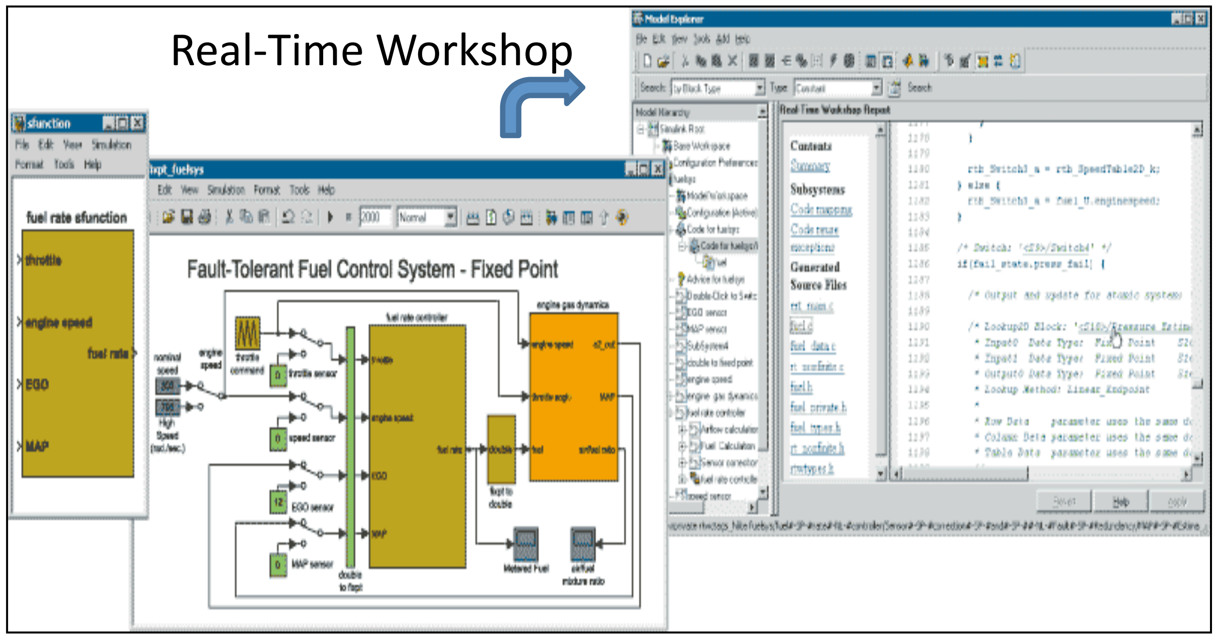
- But:
  - The synchronous cycles could be interpreted as discrete time: 0, 1, 2, 3, …, in which case we have a discrete-time semantics…
  - … and this can also be seen as an abstraction of real-time:
  - C.f. timing analysis of digital circuits
  - C.f. WCET analysis of synchronous control loops



Myths about synchronous models

- Synchronous models are non-implementable (because zero-time is impossible to achieve)
  - Hein?

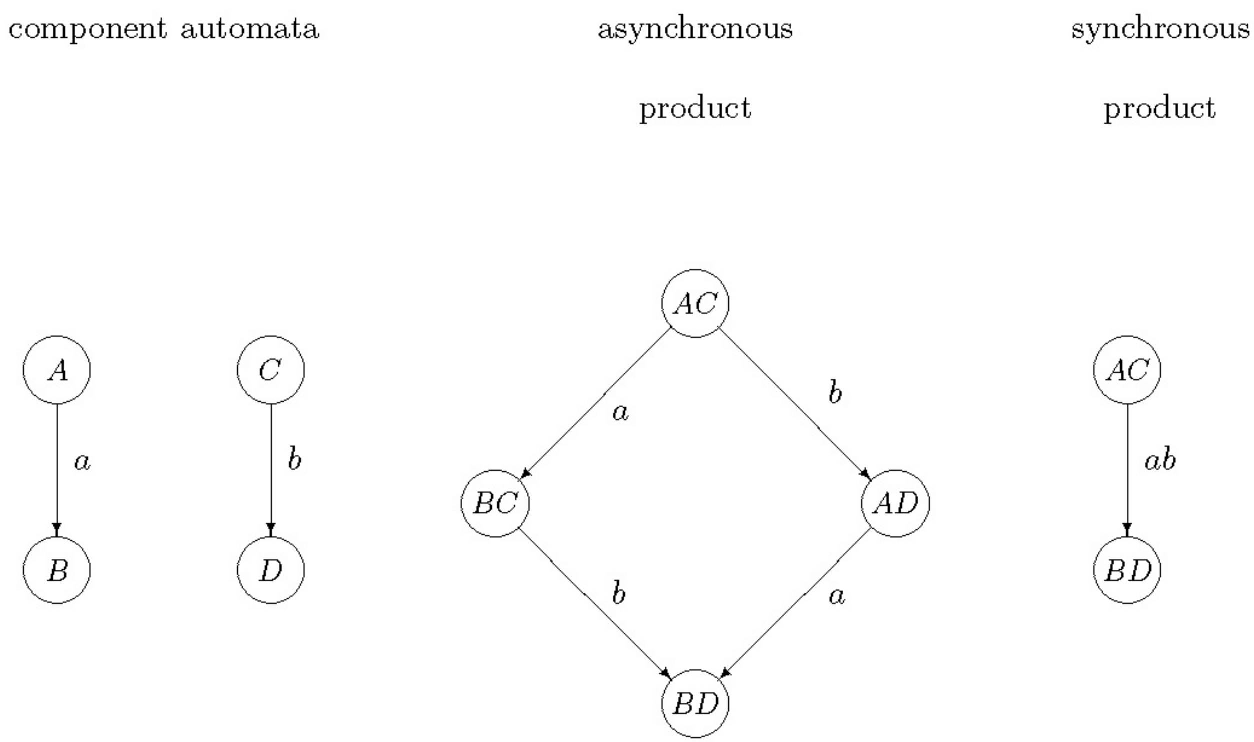




Benefits of synchronous models

- Often more light-weight than asynchronous
  - No interleaving => less state explosion
- Often deterministic
  - Easier to understand, easier to verify
- SW implementations:
  - No operating system required
  - Static scheduling, no memory allocations, no dynamic creation of processes, …
- Simple timing/schedulability analysis
  - Often simple WCET analysis also: no loops

Asynchronous vs. Synchronous Product



Lecture plan

- Part 1: Single-rate synchronous models
- Part 2: Multi-rate synchronous models
- Part 3: Feedback and Causality

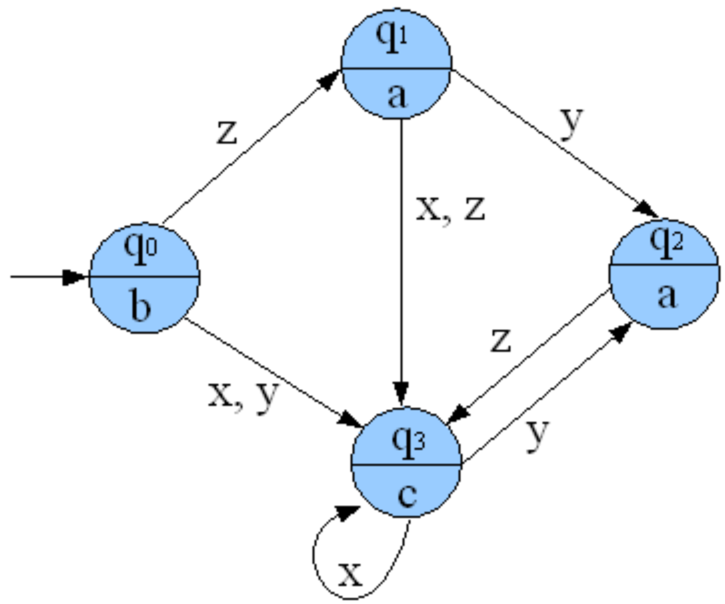
Part 1: Single-rate synchronous models

- Moore/Mealy machines
- Synchronous block diagrams
  - Inspired by discrete-time Simulink, and SCADE
- Lustre
- Esterel

Moore Machines

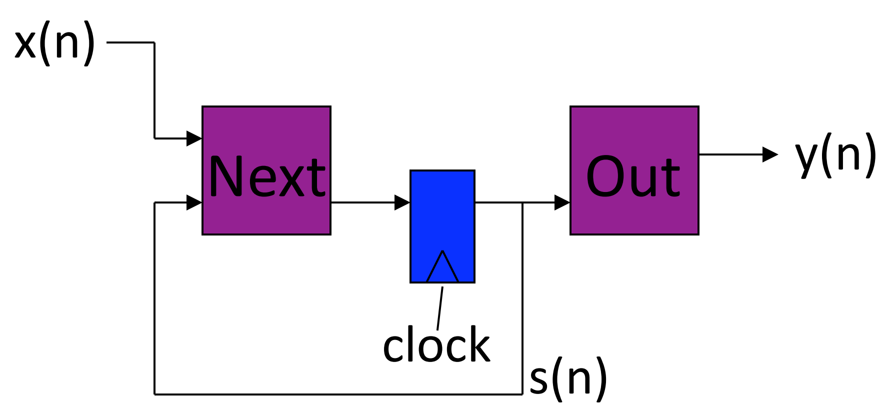
- States: {q0, q1, q2, q3}
- Initial state: q0
- Input symbols: {x,y,z}
- Output symbols: {a,b,c}
- Output function:
  – Out : States -> Outputs
- Transition function:
  – Next: States x Inputs -> States

deterministic



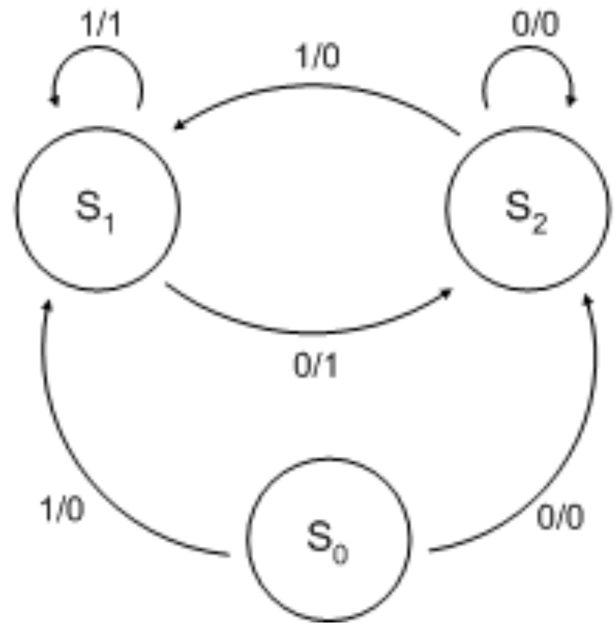
Where is the synchronous round here?

Moore machine: a circuit view



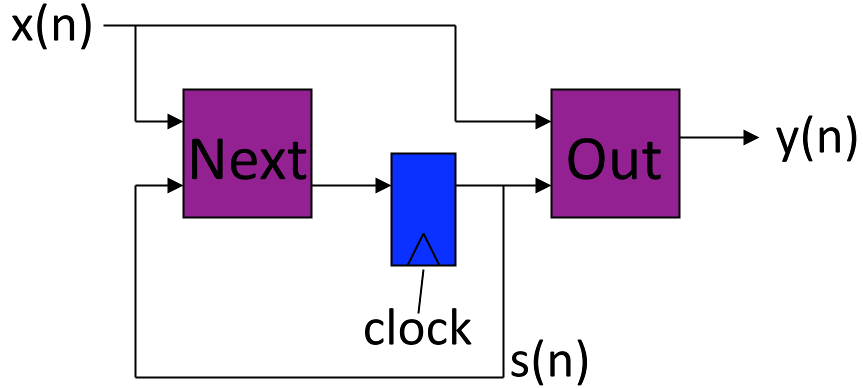
Mealy Machines

- States: {S0, S1, S2}
- Initial state: S0
- Input symbols: {0,1}
- Output symbols: {0,1}
- Output function:
  – Out : States x Inputs -> Outputs
- Transition function:
  – Next: States x Inputs -> States



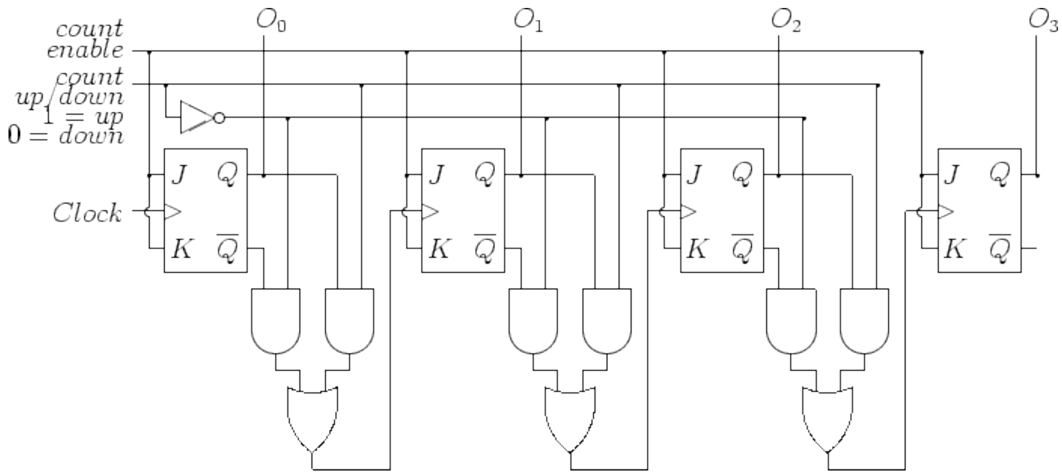
Where is the synchronous round here?

Mealy machine: a circuit view

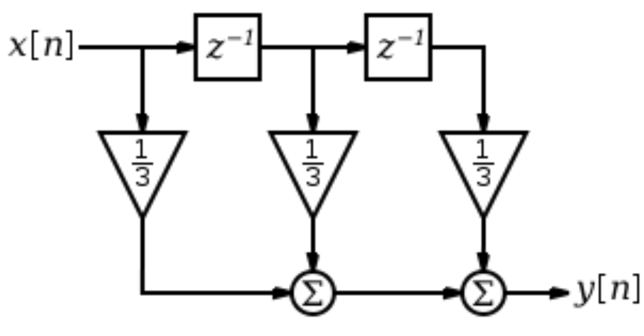


Is this a "purely synchronous" model?

Moore vs. Mealy machines



Moore or Mealy?



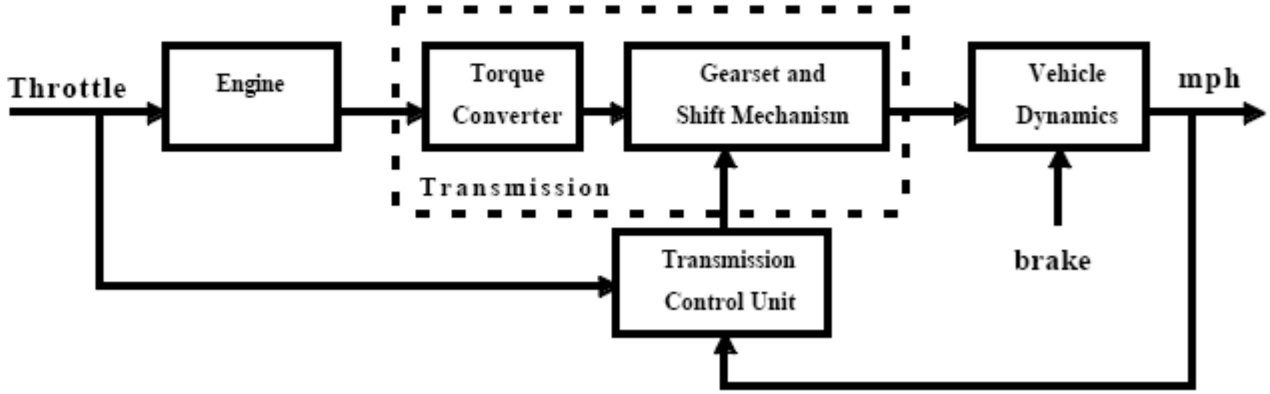
Moore or Mealy?

Moore vs. Mealy machines

- Every Moore machine is also a Mealy machine
  - Why?

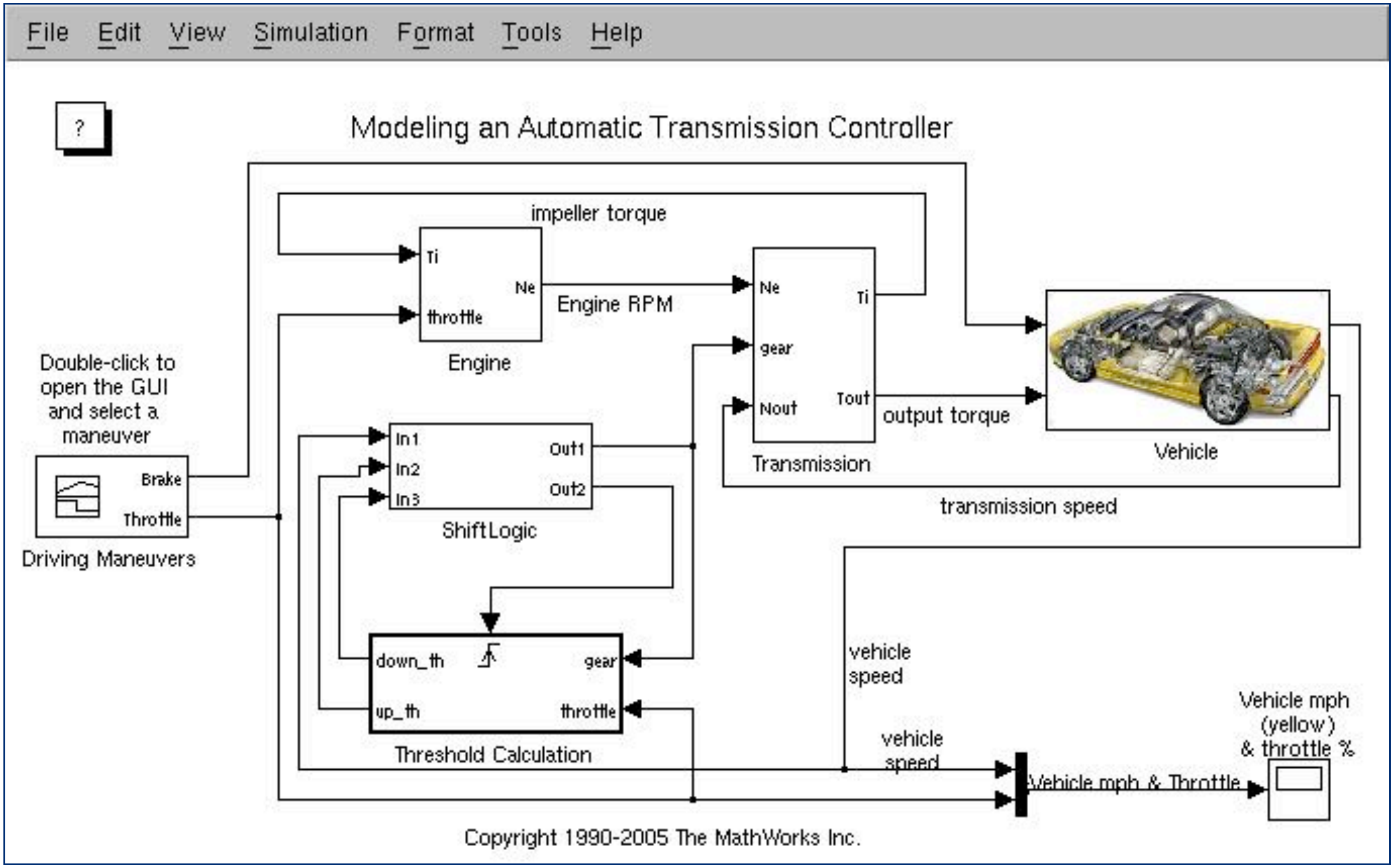
- Is it possible to transform a Mealy machine to a Moore machine?

Synchronous block diagrams

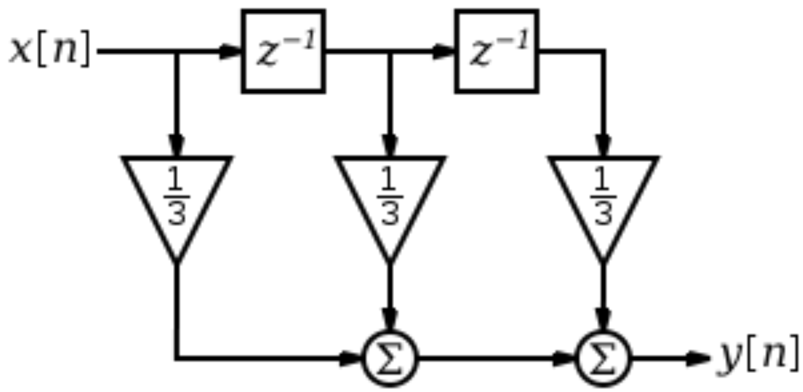


- Physical models often described in continuous-time
- Controller part (e.g., Transmission Control Unit) is discrete-time

Synchronous block diagrams



Example: FIR filter



$$y(n) = \frac{1}{3}x(n) + \frac{1}{3}x(n-1) + \frac{1}{3}x(n-2)$$

$$y(n) = \frac{1}{3}x(n) + \frac{1}{3}S_1(n) + \frac{1}{3}S_2(n)$$

$$S_1(n+1) = x(n)$$

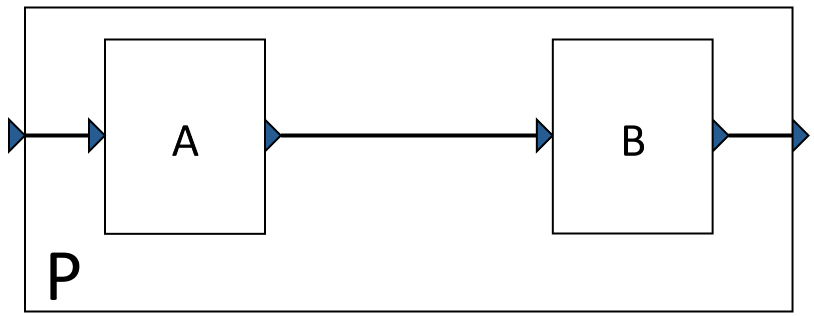
$$S_2(n+1) = S_1(n)$$

$$S_1(0) = \text{initial state}$$

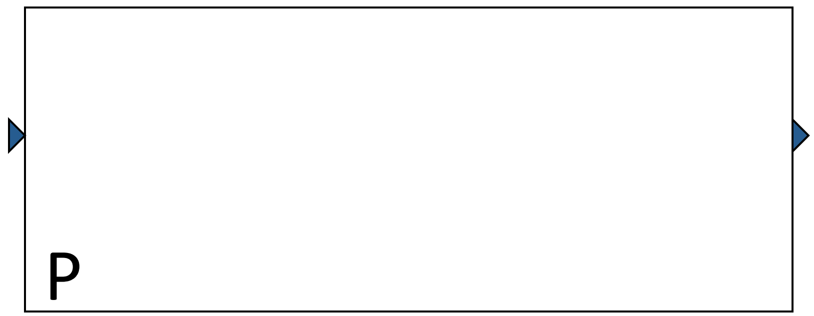
$$S_2(0) = \text{initial state}$$

What is the Mealy machine for this diagram?

Hierarchy in synchronous block diagrams



Hierarchy in synchronous block diagrams



Fundamental modularity concept

Semantics of hierarchical SBDs

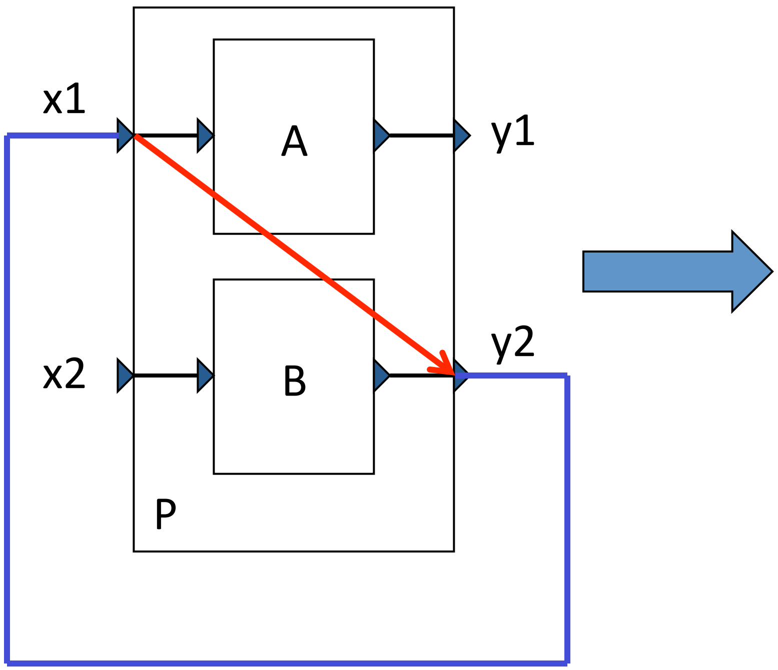
- Can we define the semantics of a composite SBD as a Mealy machine?
  - In particular, with a pair of (Out, Next) functions?

Problem with "monolithic" semantics

False I/O dependencies

=>

Model not usable in some contexts



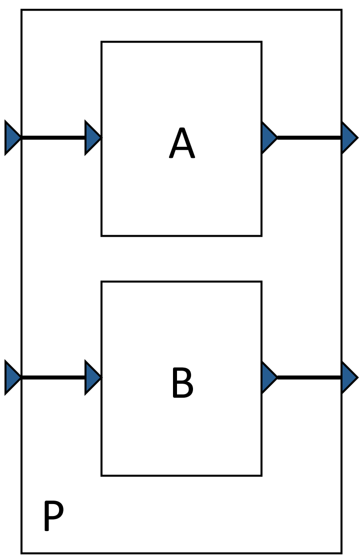

```
P.out(x1, x2) returns (y1, y2)
{
      y1 := A.out( x1 );
      y2 := B.out( x2 );

      return (y1, y2);
}
```

[DATE’08, RTAS’08, POPL’09]

Solution

- Generalize from a single, to MANY output functions



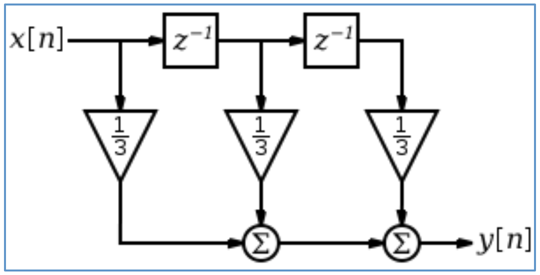

```
P.out1( in1 ) returns out1 {
   return A.out( in1 );
}

P.out2( in2 ) returns out2 {
   return B.out( in2 );
}
```

Lustre

- The FIR filter in Lustre:

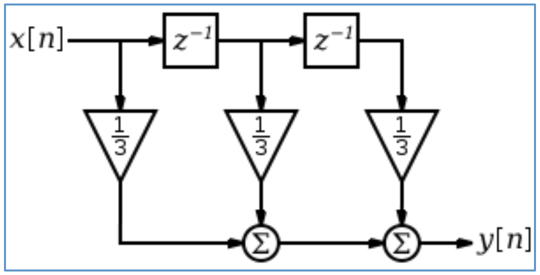
```
node fir (x : real) returns (y : real);
var
  s1, s2 : real;
let
  s1 = 0 -> pre x;
  s2 = 0 -> pre s1;
  y = x/3 + s1/3 + s2/3;
tel
```



Lustre

- The FIR filter in Lustre:

```
node fir (x : real) returns (y : real);
var
   s1, s2 : real;
let
   s1 = 0 -> pre x;
   s2 = 0 -> pre s1;
   y = x/3 + s1/3 + s2/3;
tel
```



Lustre

- The FIR filter in Lustre:

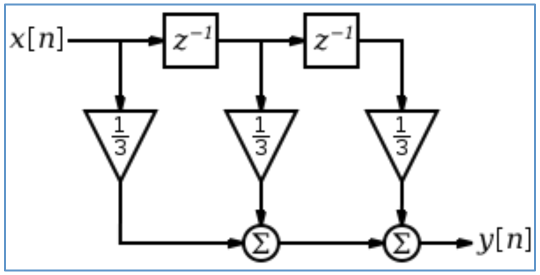
```
node fir (x : real) returns (y : real);
var
  s1, s2 : real;
let
  y = x/3 + s1/3 + s2/3;
  s2 = 0 -> pre s1;
  s1 = 0 -> pre x;
tel
```

What has changed? Is this correct?

Lustre

- The FIR filter in Lustre (no explicit state vars):

```
node fir (x : real) returns (y : real);
let
  y = x/3
    + (0 -> pre x)/3
    + (0 -> (0 -> pre pre x))/3;
tel
```



Esterel

- The FIR filter in Esterel:

```
module FIR:
  input  x : double;
  output y : double;

var s1 := 0 : double, s2 := 0 : double in
  loop
    await x ;
    emit y(x/3 + s1/3 + s2/3) ;
    s2 := s1 ;
    s1 := x ;
  end loop
end var.
```

Esterel

- The FIR filter in Esterel:

```
module FIR:
  input  x : double;
  output y : double;

var s1 := 0 : double, s2 := 0 : double in
  loop
    await x ;
    emit y(x/3 + s1/3 + s2/3) ;
    s1 := x ;
    s2 := s1 ;
  end loop
end var.
```

What has changed? Is this correct?

Esterel

- A speedometer in Esterel:

```
module SPEEDOMETER:
  input sec, cm;                       % pure signals
  output speed : double;               % valued signal
loop
  var cpt := 0 : double in
    abort
      loop
        await cm ;
        cpt := cpt + 1.0
      end loop
    when sec do
      emit speed(cpt)
    end abort
  end var
end loop.
```

Lustre

- The speedometer in Lustre:

```
node speedometer(sec, cm: bool) returns (speed: real);
var
  cpt1, cpt2 : int;
  sp1, sp2 : real;
let
  cpt1  = counter(cm, sec);
  sp1   = if sec then real(cpt1) else 0.0;
  cpt2  = counter(sec, cm);
  sp2   = if (cm and (cpt2 > 0))
          then 1.0/(real(cpt2))
          else 0.0;
  speed = max(sp1, sp2);
tel
```

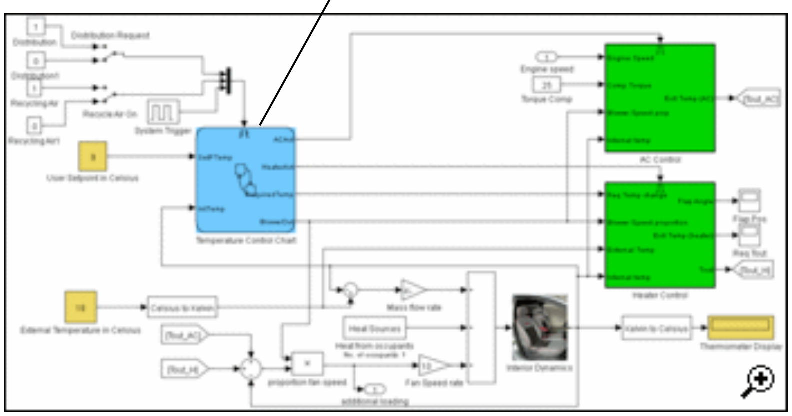
Part 2: Multi-rate synchronous models

- Synchronous block diagrams with triggers
  - Inspired by discrete-time Simulink, and SCADE
- Lustre with when/current
- What about Esterel?

Triggered and timed synchronous block diagrams

- Motivated by Simulink, SCADE

Triggered block



Simulink/Stateflow diagram

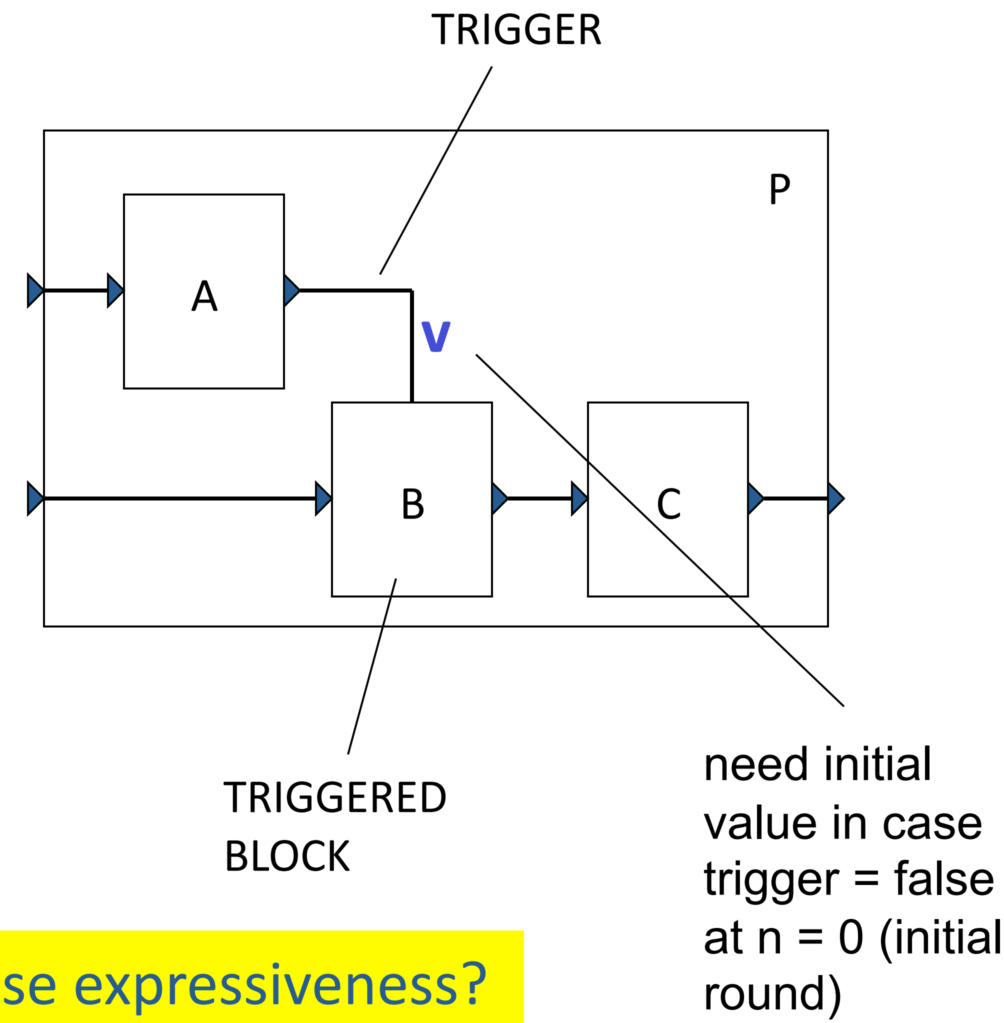
Inline Parameters = on

Sample time

Triggered synchronous block diagrams

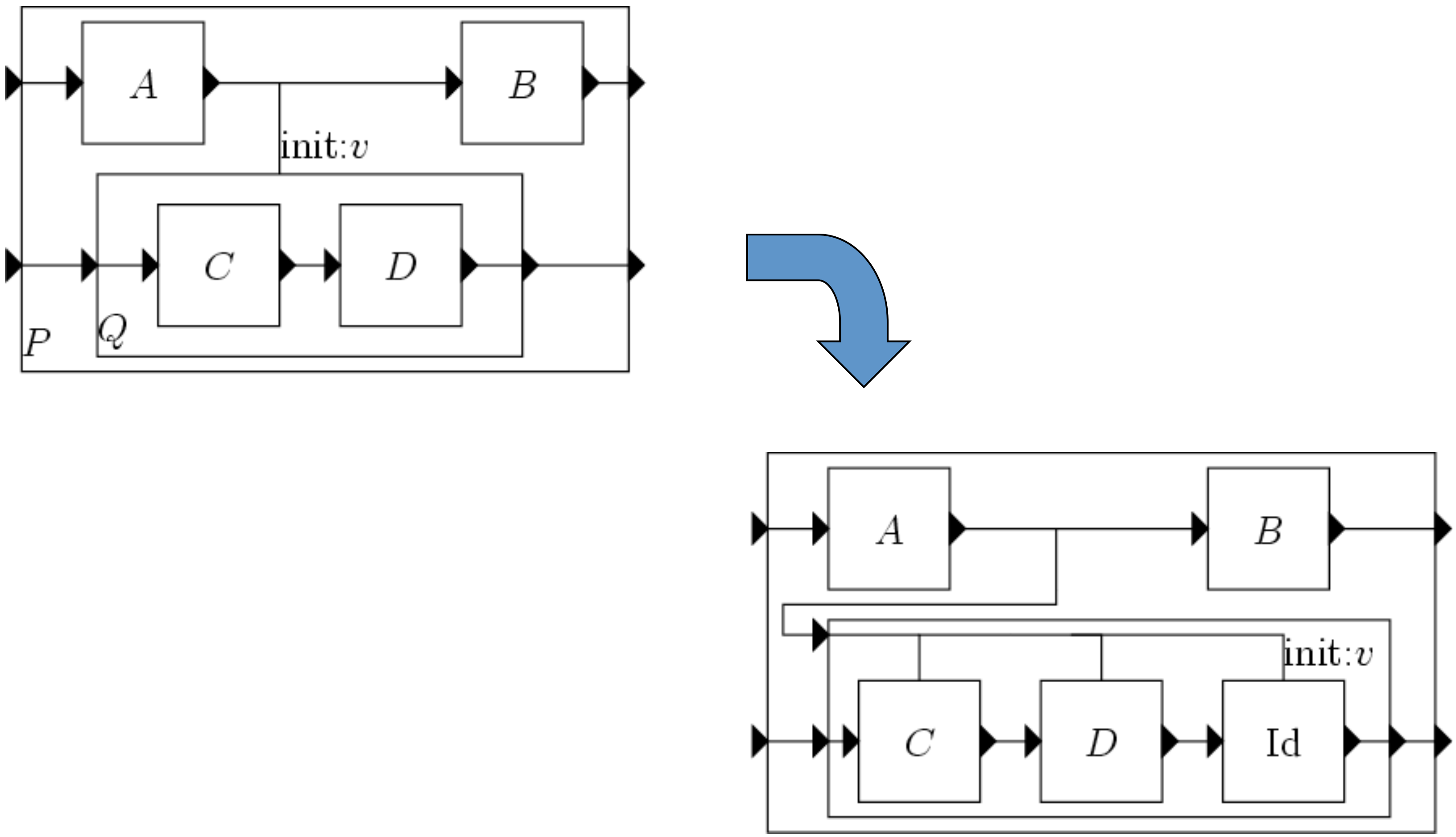
multi-rate models:

- B executed only when trigger = true
- All signals "present" always
- But not all updated at the same time
- E.g., output of B updated only when trigger is true

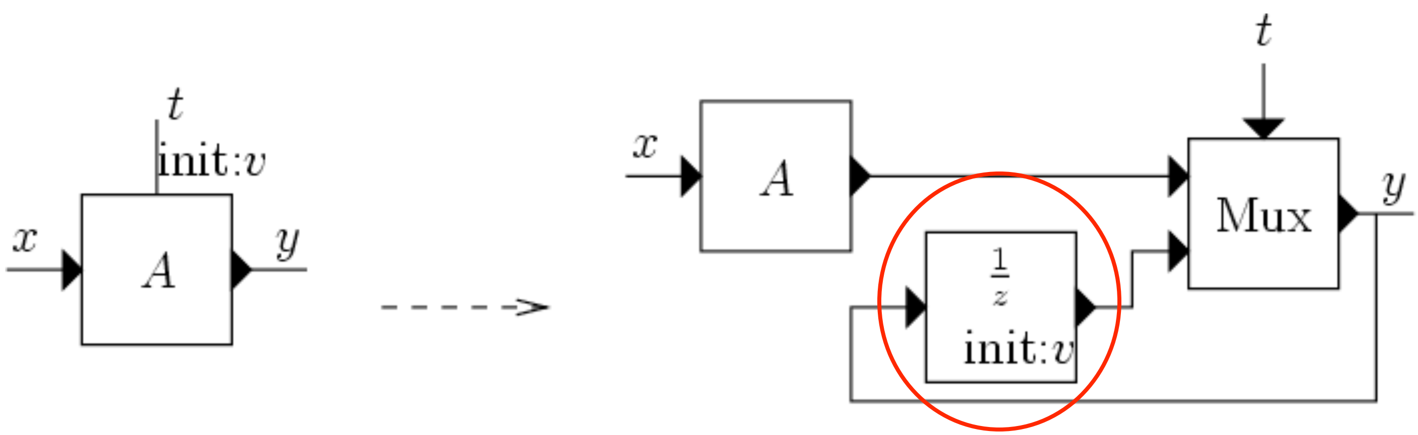


Question: do triggers increase expressiveness?

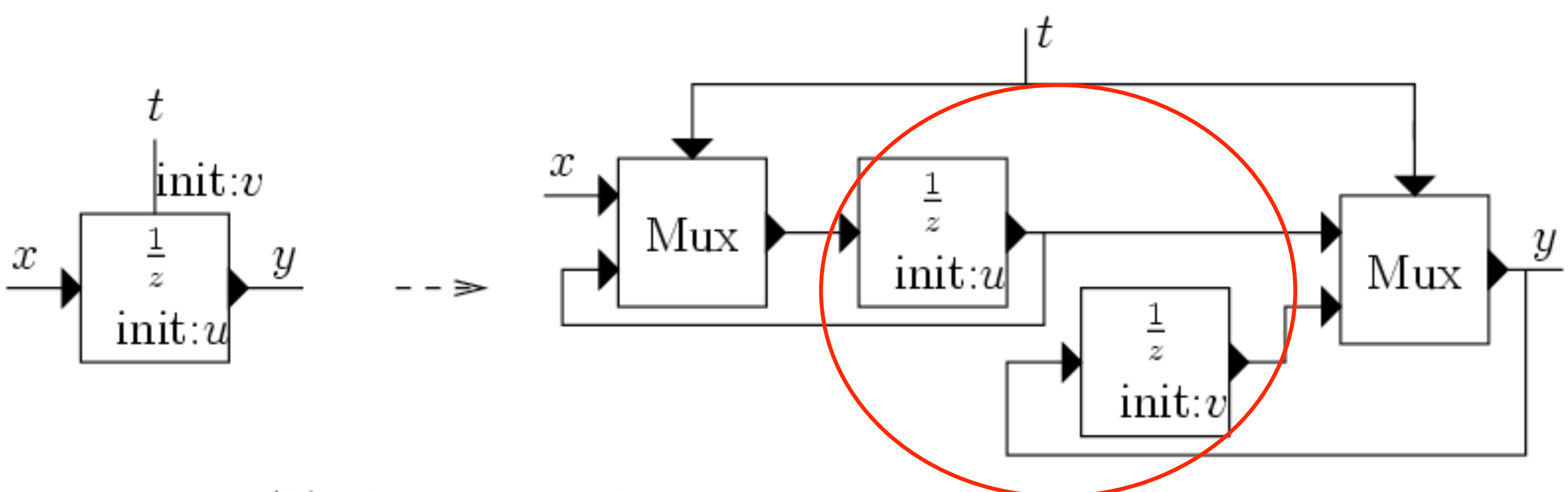
Trigger elimination



Trigger elimination: atomic blocks



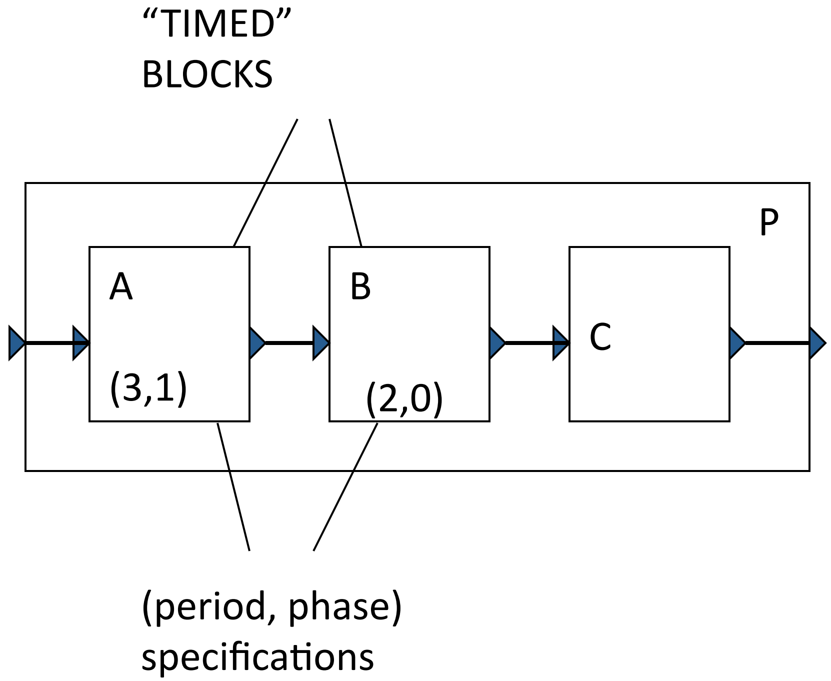
(a) eliminating the trigger from a combinational atomic block



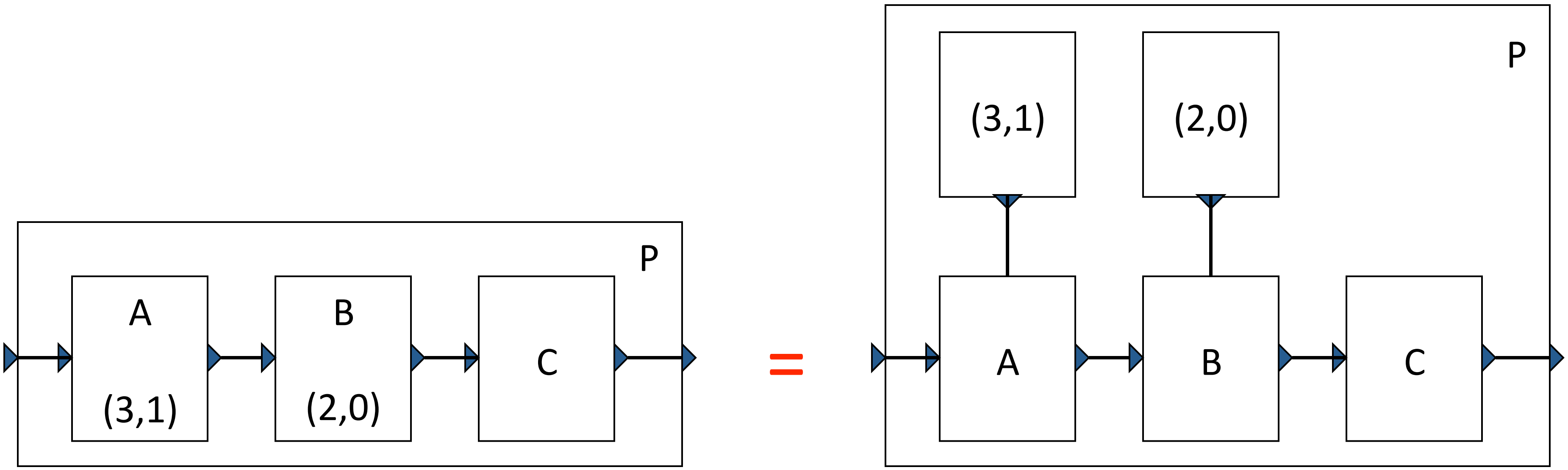
(b) eliminating the trigger from a unit-delay

Timed diagrams

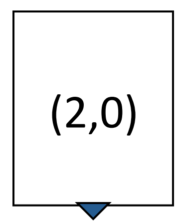
“static” multi-rate models



Timed diagrams = **statically** triggered diagrams



where



produces: **`true, false, true, false, …`**

Multi-clock synchronous programs in Lustre

- Then **`when`** and **`current`** operators:

```
node A(x: int, b: bool) returns (y: int);
let
  y = current (x when b);
tel
```

```
          x: 0 1 2 3 4 5 ...
          b: T F T F F T ...
   x when b: 0   2     5 ...
          y: 0 0 2 2 2 5 ...
```

Multi-clock synchronous programs in Lustre

```
node A(x1,x2: int, b: bool) returns (y: int);
let
  y = x1 + (x2 when b);
tel
```

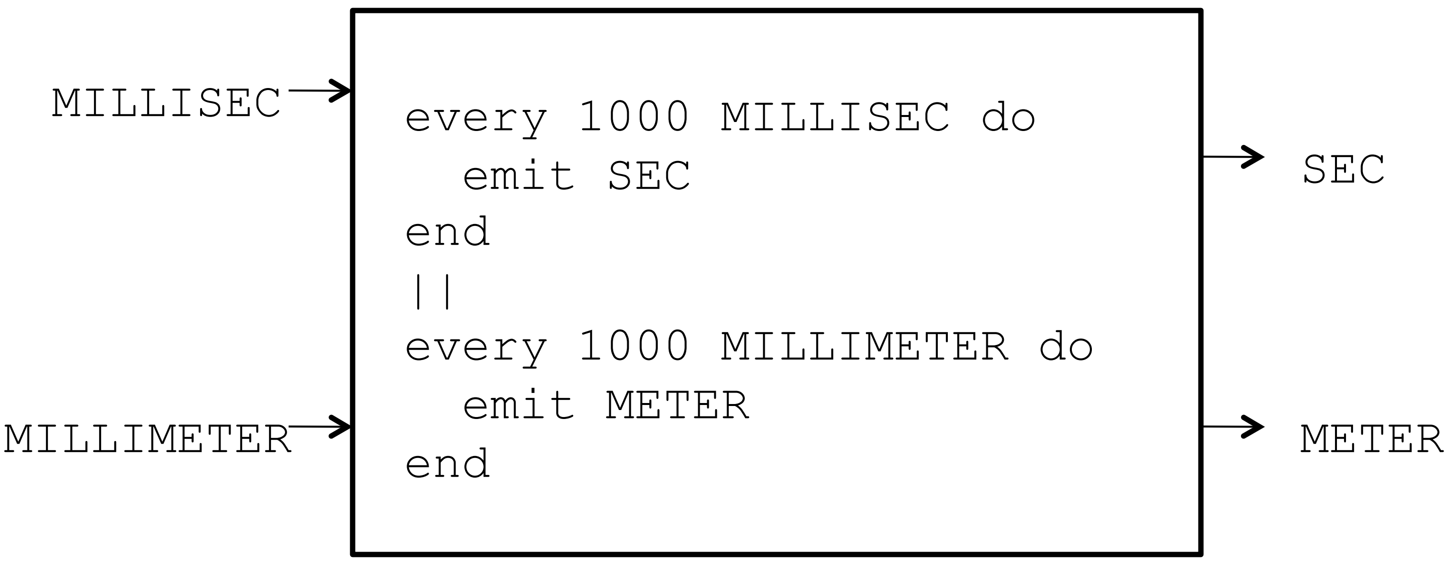
What is the meaning of this program?

Forbidden in Lustre

Multi-clock synchronous programs in Lustre

- In Lustre, every signal has a clock = “temporal” type
- The clock-calculus: a sort of type checking
  - Only signals with same clock can be added, multiplied, …
  - How to check whether two clocks (i.e., boolean signals) are the same?
    - Problem undecidable in general
    - In Lustre, check is syntactic

Multi-rate in Esterel




```
every 1000 MILLISEC do
  emit SEC
end
||
every 1000 MILLIMETER do
  emit METER
end
```

Part 3: Feedback and Causality

- Vanilla feedback:
  - Cyclic dependencies “broken” by registers, delays, …
- Unbroken cyclic dependencies:
  - Lustre/SBD solution: forbidden
  - Esterel/HW solution: forbidden unless if it makes sense
    - Malik’s example
    - Constructive semantics

Feedback in Lustre

```
node counter() returns (c : int);
let
  c = 0 -> (pre c) + 1;
tel
```

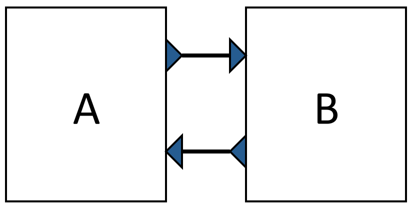
OK

```
node counter() returns (c : int);
let
  c = 0 -> c + 1;
tel
```

Rejected

Feedback in Synchronous Block Diagrams

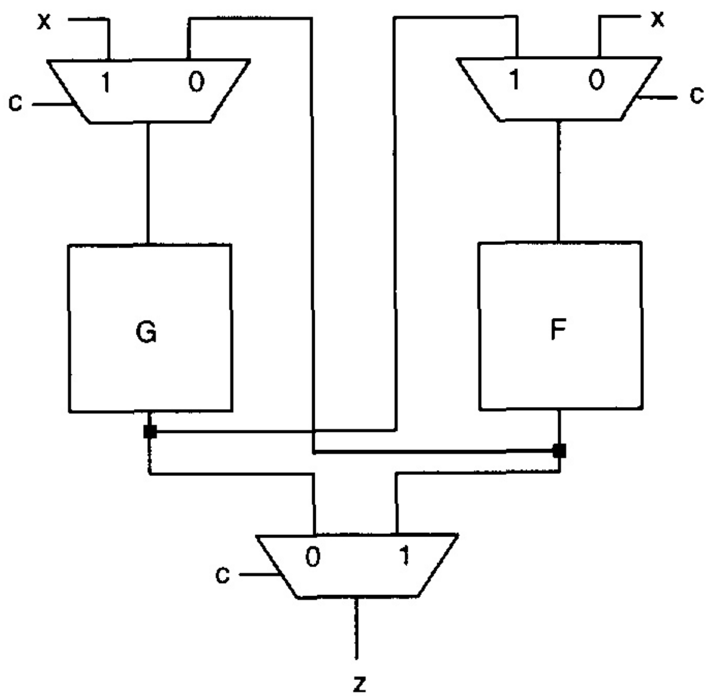
- Same as Lustre:



Rejected, unless A or B is Moore machine

What about this?

```
z = if c then
      F(G(x))
    else
      G(F(x))
```



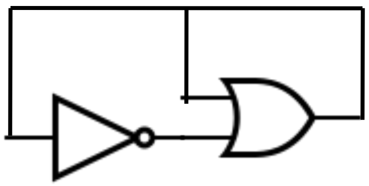
Cyclic combinational circuit.

Useful: equivalent acyclic circuit is almost 2x larger

[Malik'94]

Can we give meaning to cyclic synchronous models?

- Think of them as fix-point equations:
  - `x = F(x)`
- What is the meaning of these:
  - `x = not x`
  - `x = x`
- Is unique solution enough?
  - `x = x or not x`

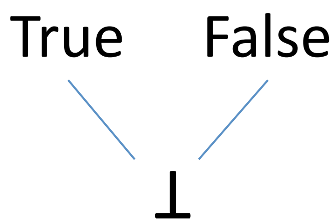


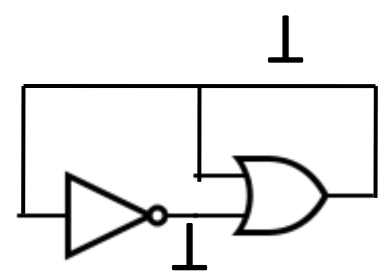
Constructive semantics

- Reason in constructive logic instead of classical logic
- “x or not x” not an axiom
- Then we cannot prove x=1 from:
  - `x = x or not x`

Constructive semantics

- Fix-point analysis in a flat CPO:
  - Start with “bottom” (undefined), iterate until fix-point is reached:
    - Guaranteed in finite number of iterations, because no. signals and no. values are both finite
  - If solution contains no undefined values, then circuit is constructive
- In our example:
  - `x = x or not x`
  - Bottom is the fix-point
  - Circuit not constructive





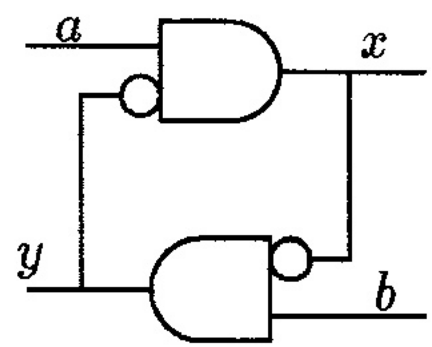
Constructive semantics: theoretical basis

- Kleene fixed point theorem:
  - *Let L be a CPO and f : L → L be a continuous (and therefore monotone) function. Then f has a least fixed point equal to* sup { bot, f(bot), f(f(bot)), … }

- In our flat CPO, continuous = monotone:
  - Non-monotone: *f(bot) > f(a),* where *a* is not *bot*
  - Not a realistic function
- In out flat CPO, termination is guaranteed.

Constructive semantics

- Another example:
  - `x = a and not y`
  - `y = b and not x`
- Here we have external inputs, must try for all possible input combinations
- Exercise!



Summary

- Synchronous model of computation:
  - Widespread, many languages, many applications
  - Easier to understand, easier to verify (than asynchronous interleaving)
  - Interesting semantically

- To go further:
  - Interesting implementation problems: how to preserve the properties that the synchronous abstraction provides (determinism, values, …) during implementation?

Questions?

References

- State machines (Moore, Mealy, …):
  - Switching and Finite Automata Theory. Zvi Kohavi, McGraw-Hill, 1978.
- Synchronous block diagrams:
  - Lublinerman and Tripakis papers on modular code generation: available from http://www-verimag.imag.fr/~tripakis/publis.html
- Synchronous languages:
  - "The synchronous languages 12 years later", Proc. IEEE, Jan 2003, and references therein.
- Constructive semantics:
  - Sharad Malik. **Analysis of cyclic combinational circuits.** ICCAD 1993.
  - Gerard Berry. **The Constructive Semantics of Pure Esterel**. Draft book, 1996, downloadable, google it.
- General, overview:
  - P. Caspi, P. Raymond and S. Tripakis. **Synchronous Programming**. In I. Lee, J. Leung, and S. Son, editors, *Handbook of Real-Time and Embedded Systems*. Chapman & Hall, 2007. Available from site above.