Synchronous Dataflow Programming

CS684: Embedded Systems

Topic 5

Paritosh Pandya

Indian Institute of Technology, Bombay

February 14, 2021

Multi-mode controller using switch

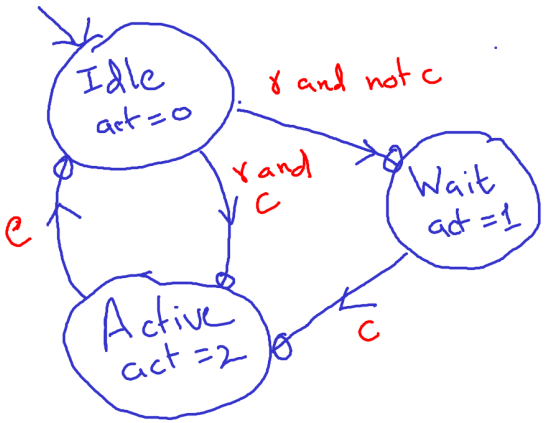
- A node can be in exactly one mode at each clock cycle.
- Equations of the currently active mode are applied.
- Each output and internal variable has exactly one equation in each mode.
- Each mode acts as a name space and clock domain.
  All pre(x) values are stored in a mode local copy. last(x) variables are global and shared between modes.
- reset blocks can be used to reset the equations under specified conditions.

Mixed language for Multi-mode Complex Control

Finite State Automata with data flow equations.

Hybrid Program

- States are modes.
- Each state has an associated set of equations.
- Transitions specify conditions for state (i.e. mode) change.
- Complex control is organized as automata with hierarchy, concurrency and sharing of flows.
- Dataflow and fsm control can be freely mixed and nested.



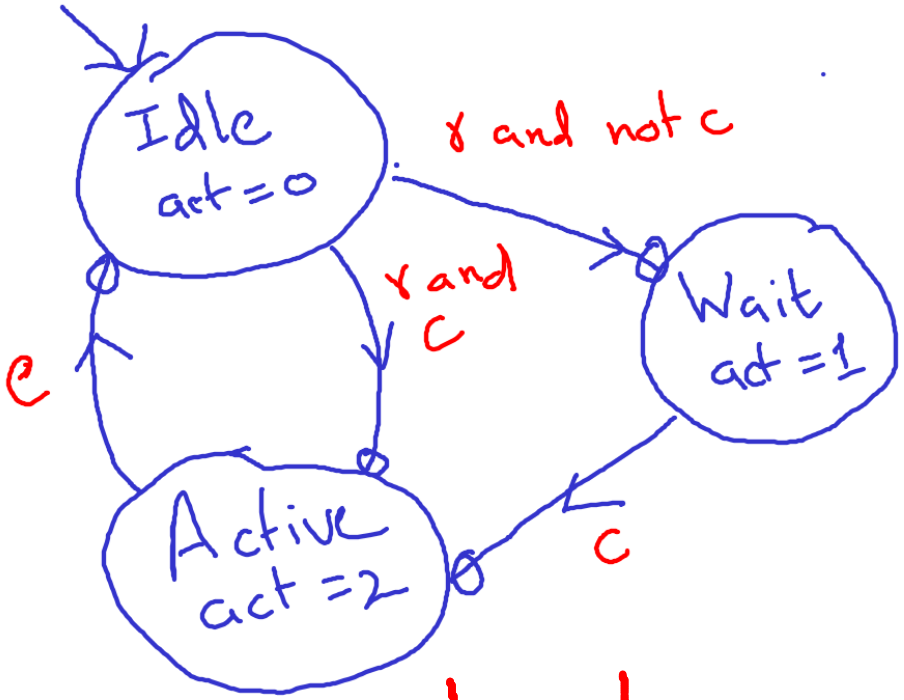
First Example

```
node myautomaton(r,c,e:bool) returns (act:int)
let
     automaton
       state Idle
                do act = 0
       until r and c then Active
           | r and not c then Wait

       state Wait
                do act = 1
       until c then Active

       state Active
                do act = 2
        until e then Idle
    end
tel
```

First Example Diagram



st	I	I	I	W	W	W	A	I	I	I	A	...
r	0	0	1	0	1	0	0	0	0	1	0	...
c	0	1	0	0	0	1	0	0	0	1	0	...
e	0	1	0	0	0	0	1	0	0	0	0	...
act	0	0	0	1	1	1	2	0	0	0	2	...
ns	I	I	W	W	W	A	I	I	I	A	A	...

Automata under Weak transitions: **until**

Structure of Automaton

- A set of states with transitions between them.
- One set of equations (i.e. mode block) for each state.
- A set of transitions going out of each state (keyword until or unless)
- Each transition has a guard giving condition under which it is taken and target state.

Execution of automaton: In each cycle

- Start State:
- With weak transitions, the start state is the Active state.
- Equations of the active state are applied.
- Guard is evaluated AFTER evaluating the active state equations.
  Guard can refer to variables defined by the equations.
  If guard true transition is taken and next state changed.
- Next State: this is the start state of the next cycle.

State as Mode Block

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       until c continue Down
       state Down
          do o = 150 -> -2 * i; stup = false;
       until c continue Up
     end
  tel
```

ST	U	U	U	U	D	D	D	D	U				...
i	4	4	3	3	3	3	3	3	3	3	3	3	...
c	0	0	0	1	0	0	0	1	0	0	1	0	0	...
o	60	5	4	4	150	−6	−6	−6	4	4	4	−6	−6	...
stup	1	1	1	1	0	0	0	0	1	1	1	0	0	...
NS	U	U	U	D	D	D	D	U						...

Same automaton with reset transitions

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       until c then Down
       state Down
          do o = 150 -> -2 * i; stup = false;
       until c then Up
     end
  tel
```

ST														...
i	4	4	3	3	3	3	3	3	3	3	3	3	3	...
c	0	0	0	1	0	0	0	1	0	0	1	0	0	...
o	60	5	4	4	150	−6	−6	−6	60	4	4	150	−6	...
stup	1	1	1	1	0	0	0	0	1	1	1	0	0	...
NS														...

Principles

- A `then` transition resets the mode block on entry to the state,
- A `continue` transition enters the states mode block WITHOUT resetting. resume.
- Each state is a mode with its own name space and clock domain.
- `pre(x)` in a mode refers to the previous value of x when the automaton was in this state.
- `last x` refers to global variable `x` shared between states. Its value is value of x in previour cycle (irrespective of the state).

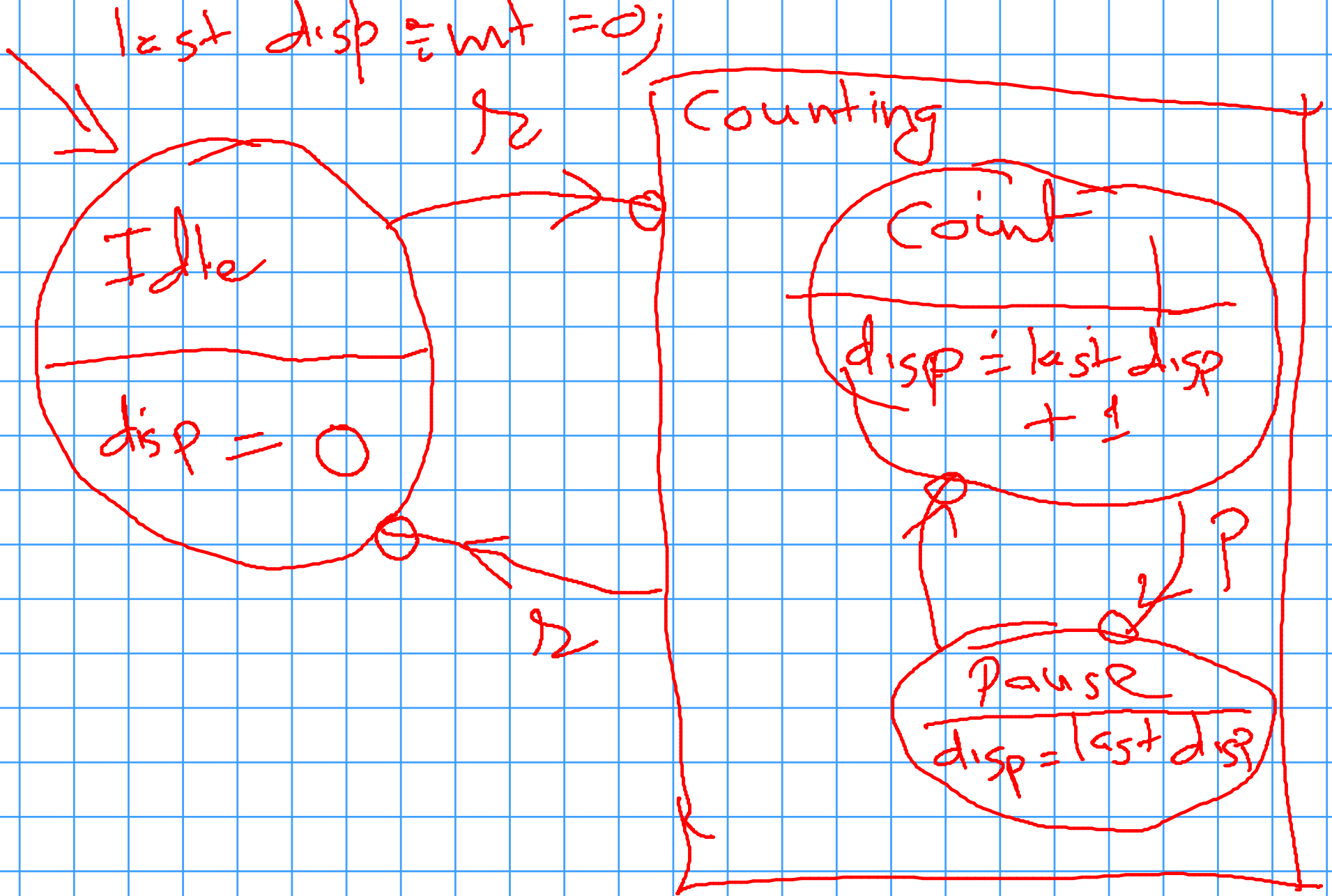
State as Mode block

```
node myautomaton() returns (y:int; stup:bool; v:int)
var last x:int = 2;
let
    y = x;
    automaton
        state Up
                var w:int;
                do x = (last x) + 1; stup = true;
                   w = 0 -> pre(w)+1; v=w;
        until  x >= 5 continue Down
        state Down
            var w:int;
            do x = (last x) - 1;  stup = false;
            w = 50 -> pre(w)-2; v=w;
        until x <= 3 continue Up
    end
tel
```

State as Mode block (cont)

```
node myautomaton() returns (y:int; stup:bool; v:int)
var last x:int = 2;
let
    y = x;
    automaton
        state Up
                var w:int;
                do x = (last x) + 1; stup = true;
                   w = 0 -> pre(w)+1; v=w;
        until  x >= 5 continue Down
        state Down
            var w:int;
            do x = (last x) - 1;  stup = false;
            w = 50 -> pre(w)-2; v=w;
        until x <= 3 continue Up
    end
tel
```

y	3	4	5	4	3	4	5	4	3	4	...
stup	1	1	1	0	0	1	1	0	0	1	...
v	0	1	2	50	48	3	4	46	44	5	...


last disp := 0;
g2
Counting
Idle
disp = 0
count
disp := last disp + 1
P
g2
Pause
disp = last disp

node stopwatch(p, r : bool) returns(disp : int)
sec : bool
automaton
state Idle do
disp = 0
until r then Counting
state Counting do
automat
state count.
until p then —
state Pause
end until p then —
until r then Idle
end

Concurrent Automata

```
node myautomaton () returns (ping,pong : bool)
let
automaton  -- A_ping
    state S1
        do   ping = true
        until true then S2
    state S2
        do   ping = false
    until pong then S1
end;
```

```
automaton -- A_pong
    state S1
        do   pong = false
        until ping then S2
    state S2
        do   pong = true
        until  true then S1
end
tel
```

global state

ST (S_1, S_1) (S_2, S_2)

ping	1	0	1	0	1	...
pong	0	1	0	1	0	...

NS (S_2, S_2) (S_1, S_1)

Strong Transitions: Example

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       unless c then Down
       state Down
          do o = 150 -> -2 * i; stup = false;
       unless c then Up
     end
  tel
```

ST								...
AS								...
i	4	4	4	4	4	4	4	...
c	0	0	0	1	0	0	1	0 ...
o	60	5	5	150	−8	−8	60	5 ...
stup	1	1	1	0	0	0	1	1 ...
NS								...

Understanding strong transitions

In each cycle,

- Start State:
- guard is evaluated BEFORE any equations.
  Guard cannot refer to current value of equation output.
- If the guard is true, ACTIVE STATE is the target state of unless.
  otherwise it remains the start state.
- equations of ACTIVE STATE are applied.
- Now until transition of the active state (if any) is applied.
- No successive unless is applied. At most one unless followed by one until.

Weak and Strong Transitions: Example

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       unless c then Down
       state Down
          do o = 150 -> -2 * i; stup = true;
       until c then Up
     end
  tel
```

ST											...
AS											...
i	4	4	4	4	4	4	4	4	4	4	...
c	0	0	0	1	0	0	0	1	1	0	...
o	60	5	5	150	60	5	5	150	150	60	...
stup	1	1	1	0	1	1	1	0	0	1	...
NS											...