Synchronous systems as finite state machines with datapath (FSMD)

Lecture 05 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2019-20

Table of Contents

lecture topics

outline:

- control models: finite state machines (FSM)
  - Mealy FSM, Moore FSM
  - conversion between the two types of FSM
- dataflow control models: FSM with datapath (FSMD)
- example: FSMD model of up-down counter
- example: FSMD model of Euclid's GCD algorithm
- FSMD model as two-stacked FSM
- datapath representation of FSMD
- proper FSMD formation rules

finite state machines (FSM)

FSM, a common control model for hardware description (and more):

- a set of states, with a distinguished initial state
- a set of inputs and a set of outputs
- a state transition function
- an output function

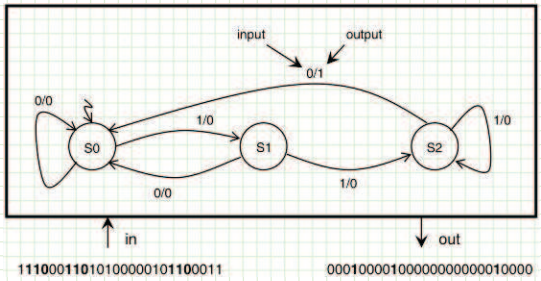
a few variations:

- with *acceptor* states, a distinguished subset of the set of states
- with an output function that depends
  - on current state and input: *Mealy* FSM
  - on the current state only: *Moore* FSM

FSM models are often presented as labelled transition graphs

example: a pattern recognizer FSM

here is an example of Mealy FSM that recognizes a binary pattern



Schaumont, Figure 5.5 - Mealy FSM of a recognizer for the pattern '110'

the output '1' signals the recognition of an occurrence of the pattern in the binary input stream

- recognition takes place through an incremental process: recognition of initial, longer and longer prefixes of the pattern

  states are the (finite) *memory* of an FSM

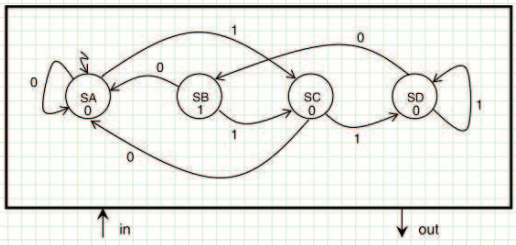
it is possible to build an equivalent Moore FSM, by a general method

equivalent Moore FSM of a given Mealy FSM

a simple procedure to convert a Mealy FSM into an equivalent Moore FSM:

- Moore states: bijection with the distinct pairs (s, o) where s is a Mealy state and there exists a transition to s labelled with output o
- for each (s, o) ↔ S assign output o to Moore state S
- for each Mealy transition s → s' labelled i/o define a transition labelled i to Moore state S' ↔ (s', o) from Moore states S ↔ (s, x), for all outputs x
- resolve any possible ambiguity about the choice of the initial state for the Moore FSM

this procedure, together with the correspondence (s0,0) ↔ SA, (s0,1) ↔ SB, (s1,0) ↔ SC, (s2,0) ↔ SD, yields an equivalent Moore FSM for the recognizer of the pattern '110', presented in the figure:



Schaumont, Figure 5.6 - Moore FSM of a recognizer for the pattern '110'

FSM con datapath (FSMD)

the FSMD model combines dataflow processing with control over it, described by an FSM

to this purpose, *instructions* are defined in a datapath description, that are executed under FSM control

in Gezel, `sfg` (*signal flow graph*) blocks represent such instructions

- similar to the `always` block, but hold expressions that are executed only if so prescribed by the FSM
- instructions, represented by `sfg` names, are the outputs from the FSM to the datapath
- more precisely, at every clock cycle the FSM may prescribe the (concurrent) execution of a *set* of the datapath instructions

an FSM controller is defined for a specific datapath:

- it acquires the instruction names as its output vocabulary
- its state transitions may be *conditioned* by expressions over the values of datapath registers

example: FSMD model of an up-down counter

the counter, here described in Gezel, is initialized to 0, then it alternates

a sequence of increments up to a maximum threshold, with

a sequence of decrements down to a minimum threshold

```
dp updown(out c :ns(3)) {
 reg a : ns(3);
 always { c = a; }
 sfg inc { a = a + 1; } // instruction inc
 sfg dec { a = a - 1; } // instruction dec
 sfg clr { a = 0; } // instruction clr
}
```

Schaumont, Listing 5.12 - Datapath for an up-down counter with three instructions

```
fsm ctl_updown(updown) {
 initial s0;
 state s1, s2;
 @s0 (clr) -> s1;
 @s1 if (a < 3) then (inc) -> s1;
                else (dec) -> s2;
 @s2 if (a > 0) then (dec) -> s2;
                else (inc) -> s1;
}
```

Schaumont, Listing 5.13 - Controller for the up-down counter

Cycle	FSM curr /next	DP instr	DP expr	a curr /next
0	s0/s1	clr	a = 0	0/0
1	s1/s1	inc	a = a+1	0/1
2	s1/s1	inc	a = a+1	1/2
3	s1/s1	inc	a = a+1	2/3
4	s1/s2	dec	a = a-1	3/2
5	s2/s2	dec	a = a-1	2/1
6	s2/s2	dec	a = a-1	1/0
7	s2/s1	inc	a = a+1	0/1
8	s1/s1	inc	a = a+1	1/2
9	s1/s1	inc	a = a+1	2/3

Schaumont, Table 5.4 - Behavior of the FSMD in Listing 5.13

the next table shows what happens at every clock cycle, for the first ten cycles

example: Euclid's GCD algorithm

the algorithm is slightly different from that seen in the previous lab tutorial:

termination with one of the two variables equal to zero

```
dp euclid(in m_in, n_in : ns(16);
          out gcd : ns(16)) {
 reg m, n : ns(16);
 reg done : ns(1);
 sfg init { m = m_in;
            n = n_in;
            done = 0;
            gcd = 0; }
 sfg reduce {  m = (m >= n) ? m - n : m;
               n = (n > m) ? n - m : n; }
 sfg outidle {  gcd = 0;
               done = ((m == 0) | (n == 0)); }
 sfg complete{ gcd = ((m > n) ? m : n);
               $display("gcd = ", gcd); }
}
```

```
fsm euclid_ctl(euclid) {
 initial s0;
 state s1, s2;
 @s0 (init) -> s1;
 @s1 if (done) then (complete) -> s2;
                else (reduce, outidle) -> s1;
  @s2 (outidle) -> s2;
}
```

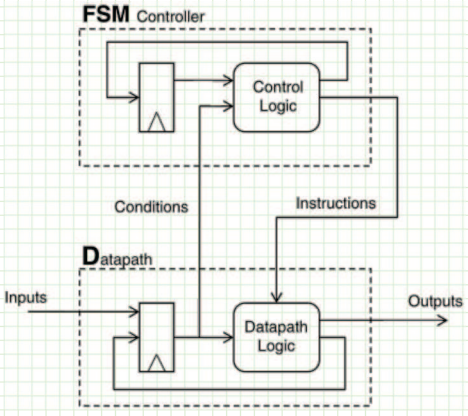
control elements introduced with the FSM: almost all those proposed in the previous lab experience

N.B.: the execution of `reduce` and `outidle` fired by the FSM in state `s1` is *concurrent*

Schaumont, Listing 5.14 - Euclid's GCD as an FSMD

FSMD model as two stacked FSMs

a dataflow model may be viewed as an FSM as well, with state space defined by its registers



FSM activities through each clock cycle:

1. both FSMs: state update (registers)
2. controller FSM: choice of next state and of instructions for the datapath FSM, determined by current state and by conditions on datapath state
3. datapath FSM: choice of next state and output determined by current state and by instructions selected by the controller FSM

Schaumont, Figure 5.7 - An FSMD consists of two stacked FSMs

in practice it is convenient to describe only the controller FSM by state transitions

not to incur in the notorious space state explosion

datapath representation of FSMD: controller

if a datapath, which can be viewed as an FSM, may be described by expressions, this should be possible for the controller FSM as well

actually in some cases there is some gain with this approach, but generally it has several shortcomings:

```
dp updown_ctl(in a_sm_3, a_gt_0 : ns(1);
              out instruction : ns(2)) {
 reg state_reg : ns(2);
 // state encoding: s0 = 0, s1 = 1, s2 = 2
 // instruction encoding: clr = 0, inc = 1, dec = 2
 always {
  state_reg  = (state_reg == 0) ? 1 :
               ((state_reg == 1) & a_sm_3) ? 1 :
               ((state_reg == 1) & ~a_sm_3) ? 2 :
               ((state_reg == 2) & a_gt_0) ? 2 : 1;
  instruction = (state_reg == 0) ? 0 :
               ((state_reg == 1) & a_sm_3) ? 1 :
               ((state_reg == 1) & ~a_sm_3) ? 2 :
               ((state_reg == 2) & a_gt_0) ? 2 : 1;
 }
}
```

Schaumont, Listing 5.15 - FSM controller for updown counter using expressions

a comparison with the example description in Listing 5.13 shows:

- a greater descriptive complexity, and
- the need to introduce a numerical encoding of states rather than a symbolic one

on the other hand, separate descriptions of controller and datapath by expressions may be combined into a single description, whereby some performance enhancement may be gained

- for example, latency may often decrease by one clock cycle, since state transition conditions, which need to be evaluated on registers in the FSMD model, in a single description may be generated and evaluated within the same clock cycle
- however, as we are going to see, besides this advantage other shortcomings show up

complete datapath representation of FSMD

```
dp updown_ctl(out c : ns(3)) {
 reg a : ns(3);
 reg state : ns(2);
 sig a_sm_3 : ns(1);
 sig a_gt_0 : ns(1);
 // state encoding: s0 = 0, s1 = 1, s2 = 2
 always {
  state = (state == 0) ? 1 :
          ((state == 1) & a_sm_3) ? 1 :
          ((state == 1) & ~a_sm_3) ? 2 :
          ((state == 2) & a_gt_0) ? 2 : 1;
  a_sm_3 = (a < 3);
  a_gt_0 = (a > 0);
  a = (state == 0) ? 0 :
      ((state == 1) & a_sm_3) ? a + 1 :
      ((state == 1) & ~a_sm_3) ? a - 1 :
      ((state == 2) & a_gt_0) ? a + 1 : a - 1;
  c = a;
 }
}
```

Schaumont, Listing 5.16 - updown counter using expressions

the comparison of the two-part FSMD description given in Listings 5.12 and 5.13 with this monolithic description shows that in the latter:

- the mingling of data processing aspects with scheduling, sequencing and control aspects complicates the *understandability* of the overall design
- the monolithic integration of these aspects also makes it harder the *reusability* of parts of the description
  - for example, reuse of the datapath under a different scheduling

on the other hand, single description offers more optimization opportunities:

- of state assignments, whereas they are automatically generated by synthesis tools in the other case
- of applications where control has a little influence, whereas much greater is the influence of high-throughput requirements

for complex, highly structured applications, FSMD description with separate FSM and datapath descriptions seems to be preferable

proper FSMD

a *proper FSMD* is one which has deterministic behaviour

> a desirable property, also enjoyed by SDF graphs, as already seen

for a hardware FSMD implementation, deterministic behaviour means that the hardware is race-free

> race conditions are a typical problem with concurrent systems, and hardware is inherently concurrent

an FSMD model is *proper* if it satisifies the following properties:

1. neither registers nor signals are assigned more than once during a clock cycle
2. no circular definition exists between signals (wires)
3. if a signal is used as an operand of an expression, it must have a known value in the same clock cycle
4. all datapath outputs must be defined (assigned) during all clock cycles

references

recommended readings:

> Schaumont, Ch. 5, Sect. 5.3-5.4.3, 5.6

for further consultation:

> Schaumont, Ch. 5, Sect. 5.7