EE 459/500 – HDL Based Digital Design with Programmable Logic

Lecture 11
FSM, ASM, FSMD, ASMD

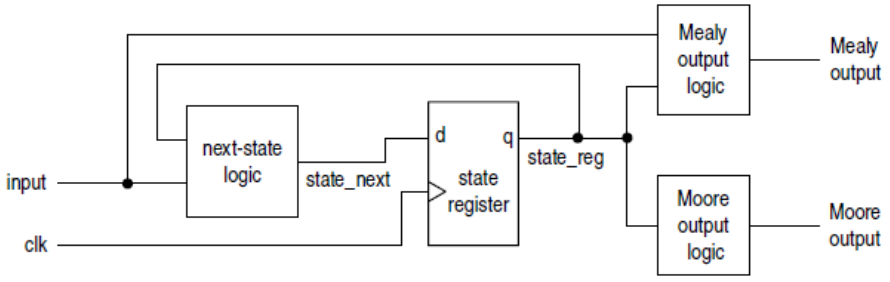
Read before class:

Chapters 4,5 from textbook

Overview

- Finite State Machines (FSMs)
  - State Graphs: general form
  - Algorithmic State Machine (ASM) charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

FSM – general form

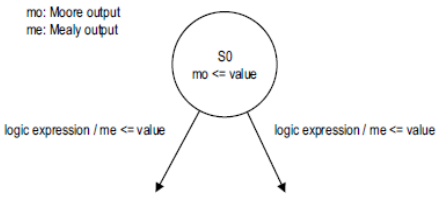


State Graph ←→ ASM chart

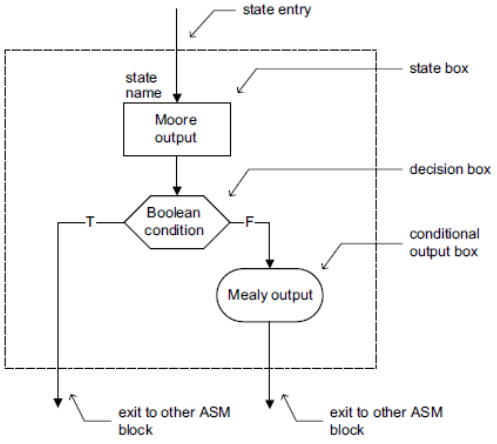
- State diagram:
  - Nodes: unique states of the FSM
  - Transitional arcs: labeled with the condition that causes the transition
- Algorithmic State Machine (ASM) chart is an alternative representation
  - Composed of a network of ASM blocks
  - ASM block:
    - State box: represents a state in the FSM
    - Optional network of decision boxes and conditional output boxes
- A state diagram can be converted to an ASM chart and vice-versa

State Graph ←→ ASM chart

State of State Graph

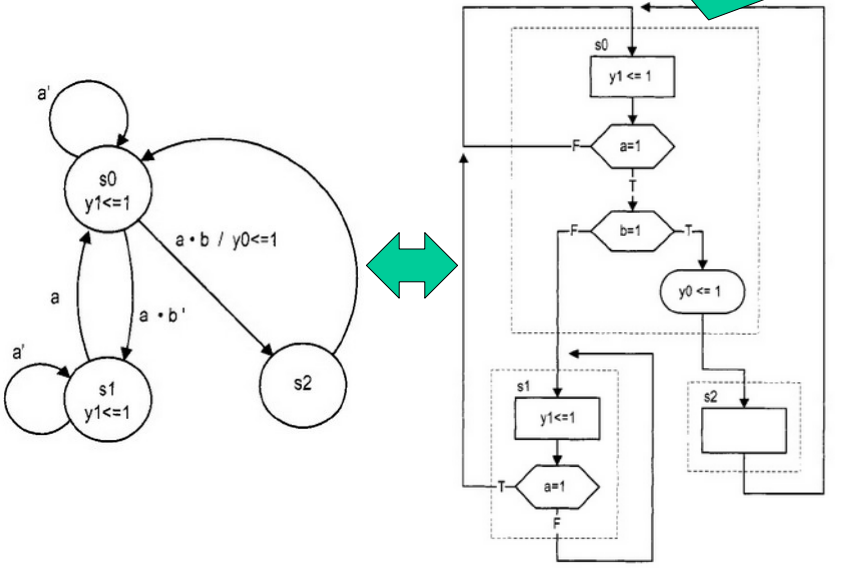


ASM Block



Example

Somewhat easier to write VHDL code for!



VHDL code of example

```
library ieee;
use ieee.std_logic_1164.all;

entity fsm_eg is
   port(
      clk, reset: in std_logic;
      a, b: in std_logic;
      y0, y1: out std_logic
   );
end fsm_eg;

architecture two_seg_arch of fsm_eg is
   type eg_state_type is (s0, s1, s2);
   signal state_reg, state_next: eg_state_type;

begin

   -- state register
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= s0;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
```

VHDL code of example

```
-- next-state/output logic
   process(state_reg,a,b)
   begin
      state_next <= state_reg;  -- default back to same state
      y0 <= '0';  -- default 0
      y1 <= '0';  -- default 0
      case state_reg is
         when s0 =>
            y1 <= '1';
            if a='1' then
               if b='1' then
                  state_next <= s2;
                  y0 <= '1';
               else
                  state_next <= s1;
               end if;
            -- no else branch
            end if;
         when s1 =>
            y1 <= '1';
            if (a='1') then
               state_next <= s0;
            -- no else branch
            end if;
         when s2 =>
            state_next <= s0;
      end case;
   end process;
end two_seg_arch;
```

Overview

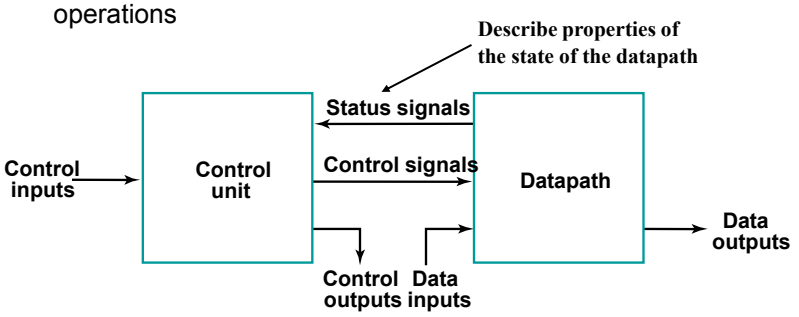
- Finite State Machines (FSMs)
  - State Graphs: general form
  - Algorithmic State Machine (ASM) charts
- Finite State Machines with Datapath (FSMD)
- ASM with Datapath (ASMD)

Finite State Machine with Data-path (FSMD)

- Combine an FSM and regular sequential circuits
- The FSM is called control-path; examines the external commands and status and generates control signals to specify operation of regular sequential circuits, which are known collectively as data-path
- FSMD used to implement systems described by **RT (register transfer) methodology**:
  - Operations are specified as data manipulation and transfer among a collection of registers

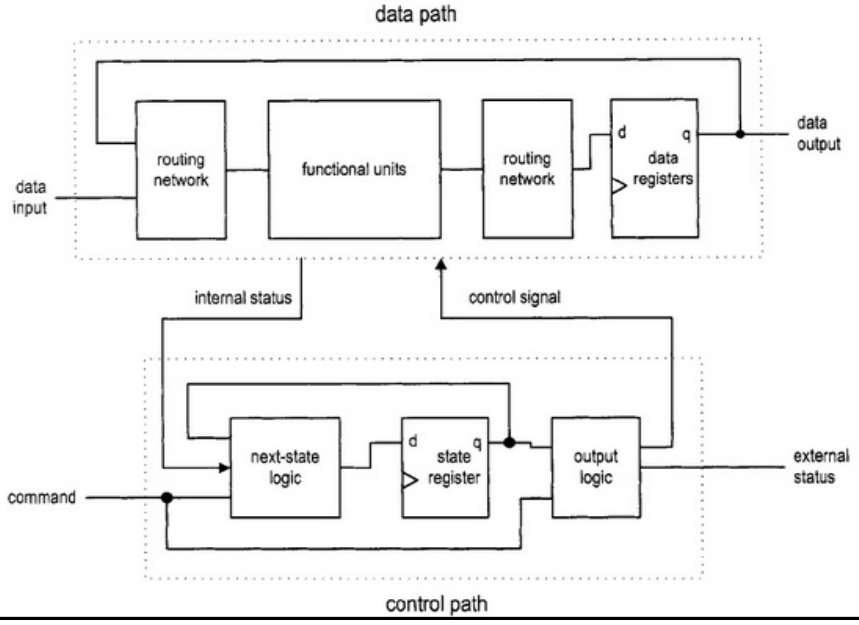
Block diagram of FSMD

- Datapath - performs data transfer and processing operations
- Control Unit - Determines the enabling and sequencing of the operations



- The control unit receives:
  - External control inputs
  - Status signals
- The control unit sends:
  - Control signals
  - Control outputs

Block diagram of FSMD (detailed)

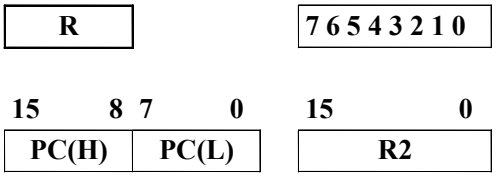


Register Transfer Operations

- Register Transfer Operations - the movement and processing of data stored in registers
- Three basic components:
  - A set of registers (operands)
  - Transfer operations
  - Control of operations
- Elementary operations - called *microoperations*
  - load, count, shift, add, bitwise "OR", etc.
- Notation: $r_{dest} \leftarrow f(r_{src1}, r_{src2},\ldots,r_{srcn})$

Register Notation

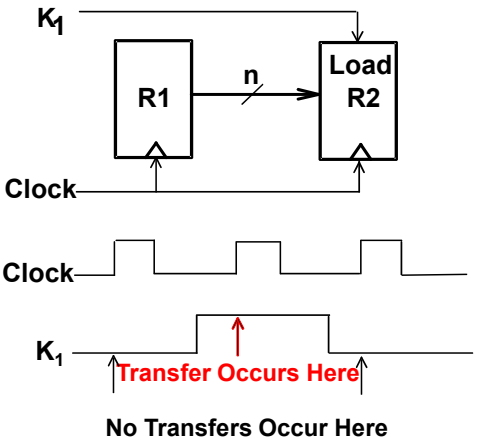
- Letters and numbers – register (e.g. R2, PC, IR)
- Parentheses ( ) – range of register bits (e.g. R1(1), PC(7:0), AR(L))



- Arrow (←) – data transfer (ex. R1 ← R2, PC(L) ← R0)
- Brackets [ ] – Specifies a memory address (ex. R0 ← M[AR], R3 ← M[PC])
- Comma – separates parallel operations

Conditional Transfer

- If (K_1 =1) then (R2 ← R1)
  ⇔ K_1: (R2 ← R1)
  where K_1 is a control expression specifying a conditional execution of the microoperation.



Microoperations

- Logical groupings:
  - Transfer - move data from one set of registers to another
  - Arithmetic - perform arithmetic on data in registers
  - Logic - manipulate data or use bitwise logical operations
  - Shift - shift data in registers

Arithmetic operations
+ Addition
– Subtraction
* Multiplication
/ Division

Logical operations
$\vee$ Logical OR
$\wedge$ Logical AND
$\oplus$ Logical Exclusive OR
$\overline{\ }$ Not

Example Microoperations

- R1← R1 + R2
  - Add the content of R1 to the content of R2 and place the result in R1.
- PC ← R1 * R6
- R1 ← R1 ⊕ R2
- (K1 + K2): R1 ← R1 ∨ R3
  - On condition K1 OR K2, the content of R1 is Logic bitwise ORed with the content of R3 and the result placed in R1.
  - NOTE: "+" (as in $K_1 + K_2$) means "OR." In R1 ← R1 + R2, + means "plus".

Arithmetic Microoperations

Symbolic Designation	Description
R0 ← R1 + R2	Addition
R0 ← $\overline{R1}$	Ones Complement
R0 ← $\overline{R1}$ + 1	Two's Complement
R0 ← R2 + $\overline{R1}$ + 1	R2 minus R1 (2's Comp)
R1 ← R1 + 1	Increment (count up)
R1 ← R1 – 1	Decrement (count down)

- Any register may be specified for source 1, source 2, or destination.
- These simple microoperations operate on the whole word

Logical Microoperations

Symbolic Designation	Description
R0 ← $\overline{R1}$	Bitwise NOT
R0 ← R1 ∨ R2	Bitwise OR (sets bits)
R0 ← R1 ∧ R2	Bitwise AND (clears bits)
R0 ← R1 ⊕ R2	Bitwise EXOR (complements bits)

Shift Microoperations

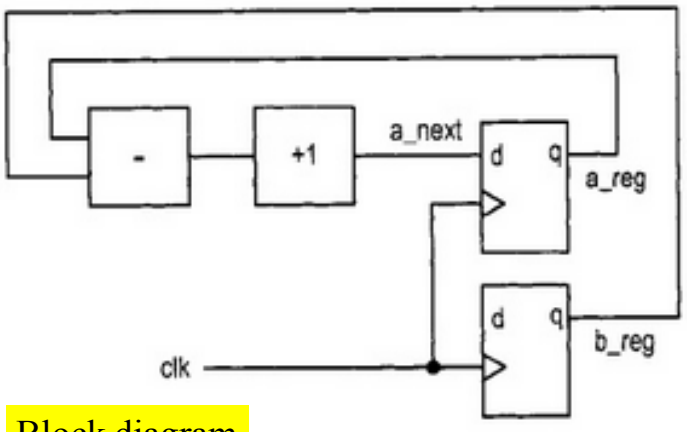
- Let R2 = 11001001

Symbolic Designation	Description	R1 content
R1 ← sl R2	Shift Left	10010010
R1 ← sr R2	Shift Right	01100100

- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to "catch" the data shifted out.
- Other shifts are possible (rotates, arithmetic)

Example

- a ← a - b + 1



Block diagram

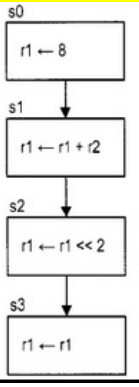
Overview

- Finite State Machines (FSMs)
  - State Graphs: general form
  - Algorithmic State Machine (ASM) charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

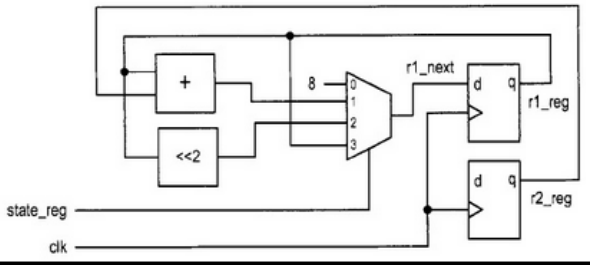
Algorithmic State Machine with Data-path (ASMD)

- Extend ASM chart to incorporate RT operations and call it ASMD (ASM with data-path):

ASMD segment

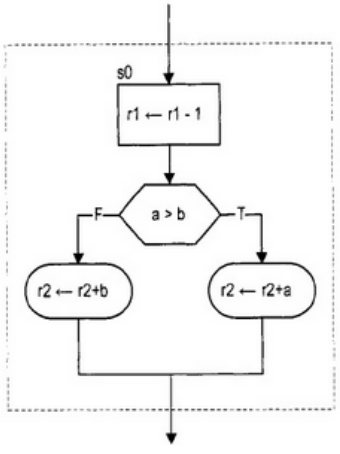


Block diagram:
Implementing the RT operations

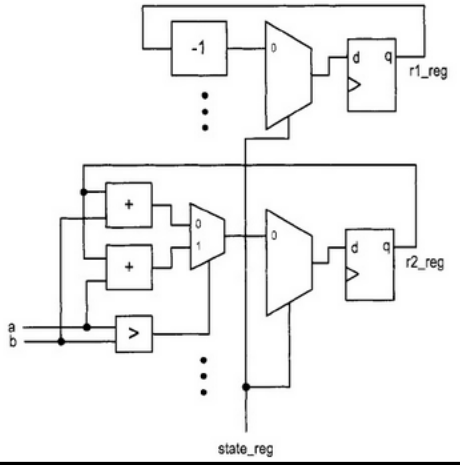


Location of RT operation inside ASM block

ASM block



Block diagram



Decision box with a register

- RT operation in an ASMD chart is controlled by an embedded clock signal
- Destination register is updated when the machine exits the current ASMD block, but not within the block!
- Example: r ← r – 1 means
  - r_next <= r_reg – 1;
  - r_reg <= r_next at the rising edge of the clock (when machine exits current block)

Example

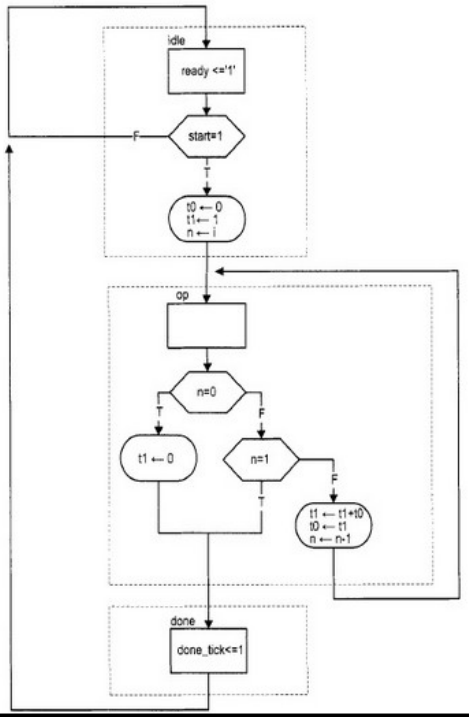
- Fibonacci number circuit
- A sequence of integers
- fib(i) =

  0, if i = 0

  1 if i = 1

  fib(i-1) + fib(i-2), if i > 1

ASMD chart



VHDL code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fib is
    port(
        clk, reset: in std_logic;
        start: in std_logic;
        i: in std_logic_vector(4 downto 0);
        ready, done_tick: out std_logic;
        f: out std_logic_vector(19 downto 0)
    );
end fib;

architecture arch of fib is
   type state_type is (idle,op,done);
   signal state_reg, state_next: state_type;
   signal t0_reg, t0_next, t1_reg, t1_next: unsigned(19 downto 0);
   signal n_reg, n_next: unsigned(4 downto 0);

begin

   -- fsmd state and data registers
   process(clk,reset)
   begin
      if reset='1' then
         state_reg <= idle;
         t0_reg <= (others=>'0');
         t1_reg <= (others=>'0');
         n_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         t0_reg <= t0_next;
         t1_reg <= t1_next;
         n_reg <= n_next;
      end if;
   end process;
```

```
  -- fsmd next-state logic
    process(state_reg,n_reg,t0_reg,t1_reg,start,i,n_next)
    begin
       ready <='0';
       done_tick <= '0';
       state_next <= state_reg;
       t0_next <= t0_reg;
       t1_next <= t1_reg;
       n_next <= n_reg;
       case state_reg is
          when idle =>
             ready <= '1';
             if start='1' then
                t0_next <= (others=>'0');
                t1_next <= (0=>'1', others=>'0');
                n_next <= unsigned(i);
                state_next <= op;
             end if;
          when op =>
             if n_reg=0 then
                t1_next <= (others=>'0');
                state_next <= done;
             elsif n_reg=1 then
                state_next <= done;
             else
                t1_next <= t1_reg + t0_reg;
                t0_next <= t1_reg;
                n_next <= n_reg - 1;
             end if;
          when done =>
             done_tick <= '1';
             state_next <= idle;
       end case;
    end process;

    -- output
    f <= std_logic_vector(t1_reg);

end arch;
```

Summary

- Algorithmic State Machine charts are somewhat more convenient to use to write behavioral VHDL code
- Finite State Machines with Datapath (FSMD) and Algorithmic State Machine with Datapath (ASMD) are useful when we care about the internal structure of the circuit (e.g., we want the synthesis tool to preserve the pipeline structure)