EECE-4740/5740 Advanced VHDL and FPGA Design
Lecture 4

FSM, ASM, FSMD, ASMD

Cristinel Ababei
Dept. of Electrical and Computer Engr.
Marquette University

1

Overview

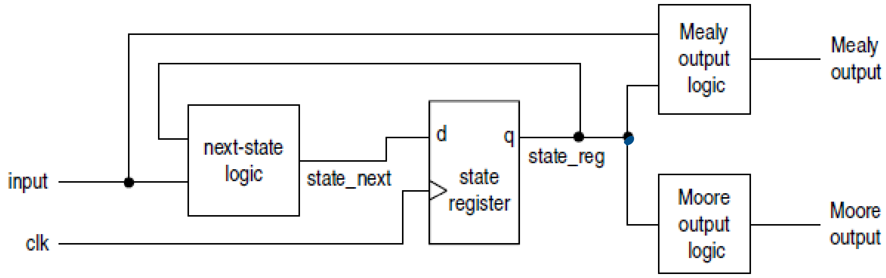
- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

2

FSM – general form

- In practice, main application of an FSM is to act as controller of a large digital system
- It examines external commands and status and activates proper control signals to control operation of a data path (composed of regular sequential and combinational components)



State Graph ←→ ASM Chart

- State graph or 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
  - More descriptive for applications with complex transition conditions and actions!
- A state diagram can be converted to an ASM chart and vice-versa

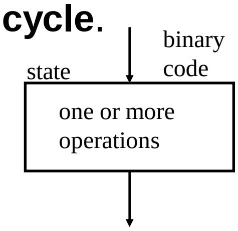
ASM Charts

- Algorithmic State Machine (ASM) Chart is a popular high-level flowchart-like graphical model (or notation) to specify the (hardware) algorithms in digital systems.
- Major differences from flowcharts are:
  - uses 3 types of boxes: state box (similar to operation box), decision box, and conditional box
  - contains exact (or precise) timing information; flowcharts impose a relative timing order for the operations.
- From the ASM chart it is possible to obtain
  - the control
  - the architecture (data processor)

5

Components of ASM Charts

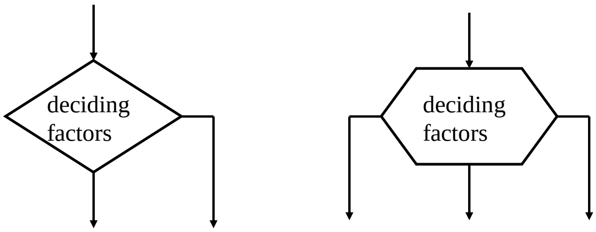
- The state box is rectangular in shape. It has at most one entry point and one exit point and is used to specify one or more operations which could be simultaneously completed in one **clock cycle**.



6

Components of ASM Charts

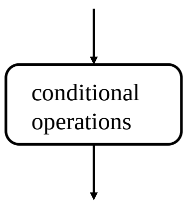
- The decision box is diamond in shape. It has one entry point but multiple exit points and is used to specify a number of alternative paths that can be followed.



7

Components of ASM Charts

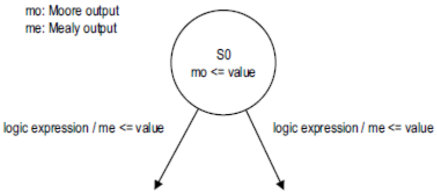
- The conditional box is represented by a rectangle with rounded corners. It always follows a decision box and contains one or more *conditional operations* that are only invoked when the path containing the conditional box is selected by the decision box.



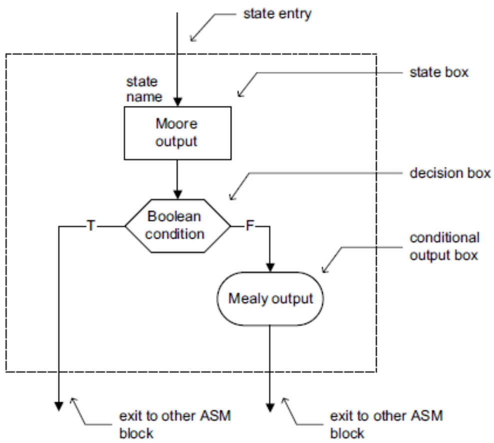
8

State Graph ←→ ASM Chart

State of State Graph



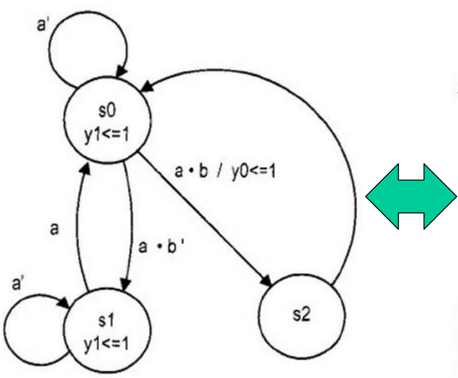
ASM Block

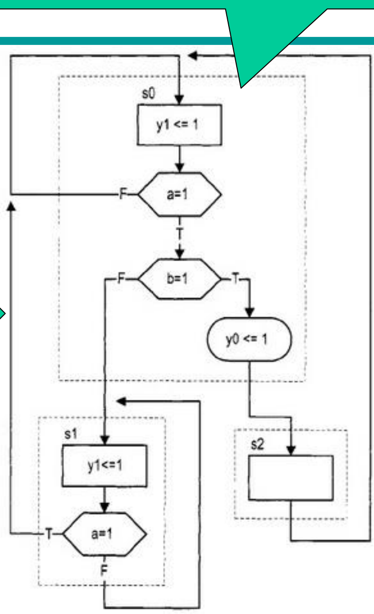


9

Example

Easier to write VHDL!





10

VHDL code

```
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;
```

11

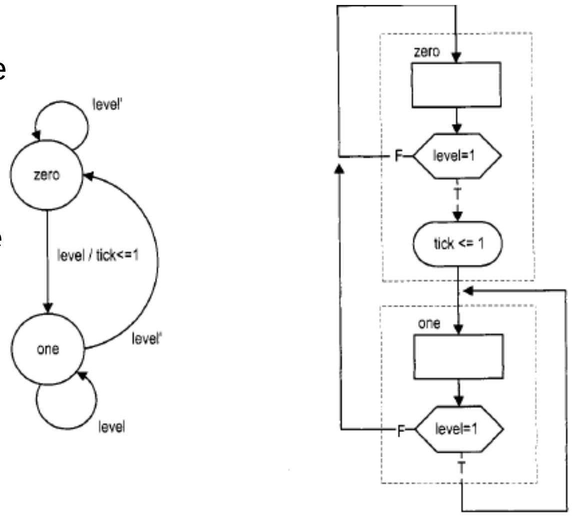
VHDL code

```
   -- 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;
```

12

Example: Rising-edge detector

- Generates a short, one-clock-cycle pulse (called a tick) when input signal changes from ‘0’ to ‘1’
- Here: Mealy machine
- Assignment: Moore machine
- See more examples in Ch.5 of P.Chu’s book (e.g., debouncing circuit)



(a) State diagram (b) ASM chart

13

VHDL code

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

entity edge_detect is
  port(
    clk, reset: in std_logic;
    level: in std_logic;
    tick: out std_logic
  );
end edge_detect;

architecture MEALY_ARCHITECTURE of edge_detect is

type state_type is (S0, S1);
signal state_current, state_next : state_type;

begin

        -- state register; process #1
        process (clk , reset)
        begin
                if (reset = '1') then
                        state_current <= S0;
                elsif (clk' event and clk = '1') then
                        state_current <= state_next;
                end if;
        end process;
```

14

VHDL code

```
-- next state and output logic; process #2
process (state_current, level)
begin
        state_next <= state_current;
        tick <= '0';
        case state_current is
                when S0 =>
                        if level = '1' then
                                state_next <= S1;
                                tick <= '1';
                        end if;
                when S1 =>
                        if level = '0' then
                                state_next <= S0;
                        end if;
        end case;
end process;

end MEALY_ARCHITECTURE;
```

15

Overview

- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

16

Finite State Machine with Data-path (FSMD): Enabler of RT design methodology

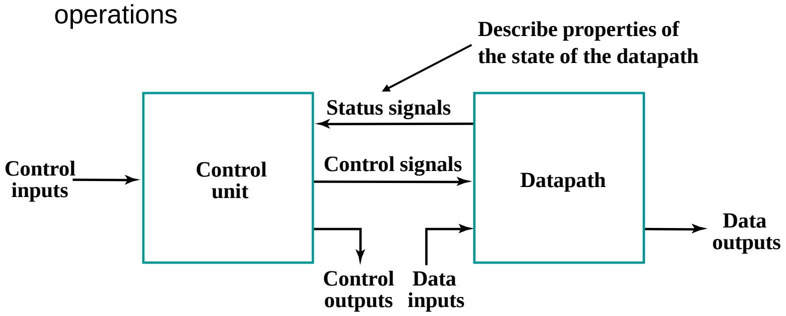
- **FSMD = *FSM* +**
  ***Regular Sequential Circuits* +**
  Combinational Circuits

- The FSM is called control-path (control logic); the rest is called data-path
- FSMD used to implement systems described by **register transfer (RT) methodology**

17

Conceptual block diagram of FSMD

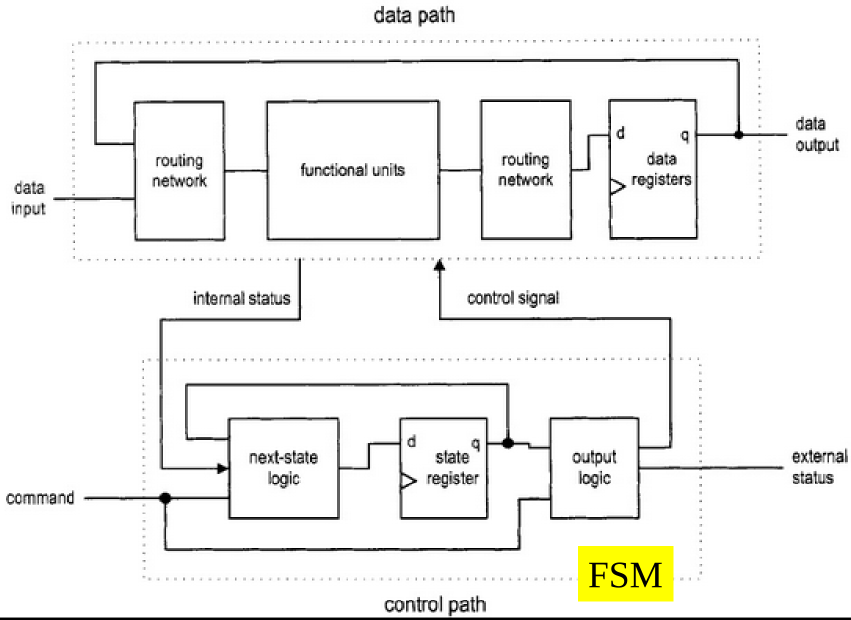
- **Datapath** - performs data transfer and processing operations
- **Control Unit** - Determines the enabling and sequencing of the operations



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

18

Block diagram of FSMD (detailed)



RT Design Methodology

- RT operations are specified as data manipulation and transfer among a collection of registers
- A circuit based on RT methodology specifies which RT operations should be executed in each step
- RT operations are done in a clock-by-clock basis; so, timing is similar to a state transition of a FSM
- Hence, FSM is natural choice to specify the sequencing of an RT algorithm
- **Extend ASM chart to incorporate RT operations and call it ASMD (ASM with datapath) chart**

RT Operations (8 slides; can be skipped)

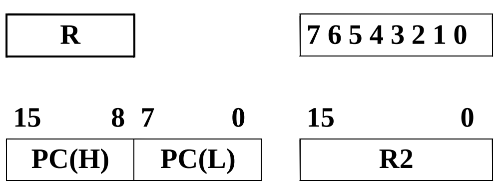
- Register Transfer Operations - the movement and processing of data stored in registers
- Three basic components:
  - A set of registers (operands)
  - Transfer operations
  - Control of operations
- Elementary operations - called microoperations
  - load, count, shift, add, bitwise "OR", etc.

- Notation: $r_{dest} \leftarrow f(r_{src1}, r_{src2},\ldots,r_{srcn})$

21

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))

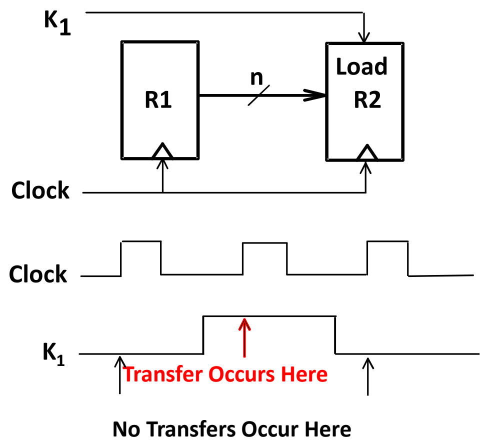


- Arrow (←) – data transfer (ex. R1 ← R2, PC(L) ← R0)
- Brackets [ ] – Specifies a memory address (ex. R0 ← M[AR], R3 ← M[PC])
- Comma – separates parallel operations

22

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.



23

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
∨ Logical OR
∧ Logical AND
⊕ Logical Exclusive OR
¯ Not

24

Example Microoperations

- $R1 \leftarrow R1 + R2$
  - Add the content of R1 to the content of R2 and place the result in R1.
- $PC \leftarrow R1 * R6$
- $R1 \leftarrow R1 \oplus R2$
- $(K1 + K2)$: $R1 \leftarrow R1 \vee R3$
  - 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 $R1 \leftarrow R1 + R2$, + means "plus".

25

Arithmetic Microoperations

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

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

26

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)

27

Shift Microoperations

- Let R2 = 11001001

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

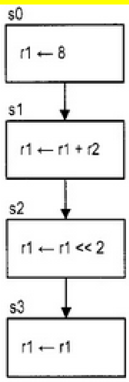
- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to “catch” the data shifted out.
- Other shifts are possible (rotates, arithmetic)

28

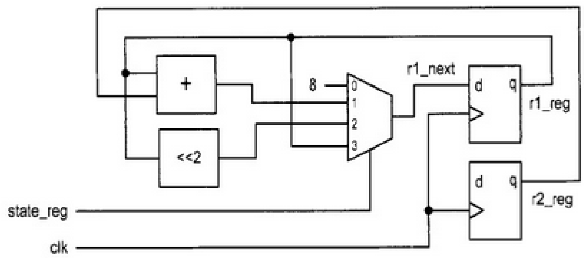
Algorithmic State Machine with Data-path (ASMD)

- **Extend ASM chart to incorporate RT operations** → ASMD (ASM with datapath) chart

ASMD segment

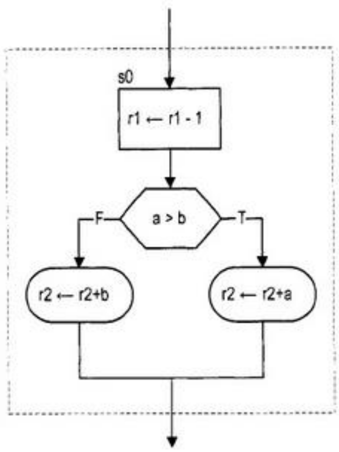


Block diagram: implementation of RT operations (likely synthesized by CAD tool for you or directly specified as structural description)

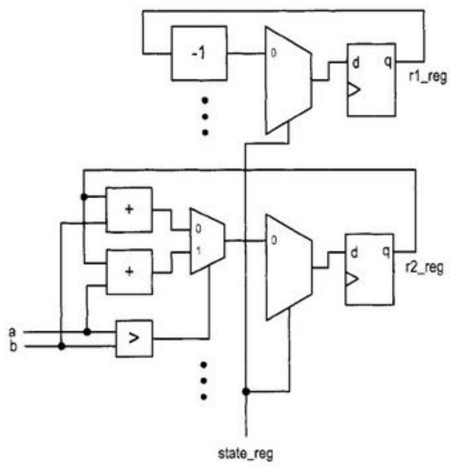


Location of RT operation inside ASM block

ASM block



Block diagram: implementation of RT operations (likely synthesized by CAD tool for you or directly specified as structural description)



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: $r \leftarrow r - 1$ means:
  - r_next <= r_reg – 1;
  - r_reg <= r_next at the rising edge of the clock (when machine exits current block)

31

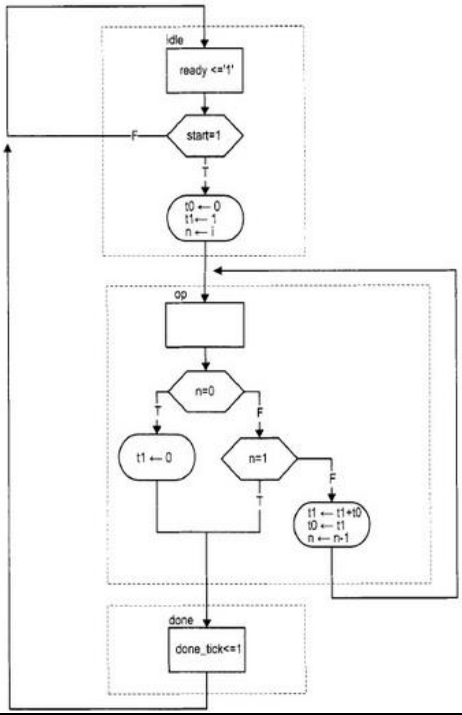
Example of ASMD description

- Fibonacci number circuit
  - Page 140 in Textbook
- A sequence of integers
- fib(i) =

$$fib(i) = \begin{cases} 0, & \text{if } i = 0 \\ 1, & \text{if } i = 1 \\ fib(i-1) + fib(i-2), & \text{if } i > 1 \end{cases}$$

32

ASMD chart



33

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;
```

34

```
   -- 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;
```

35

Overview

- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

36

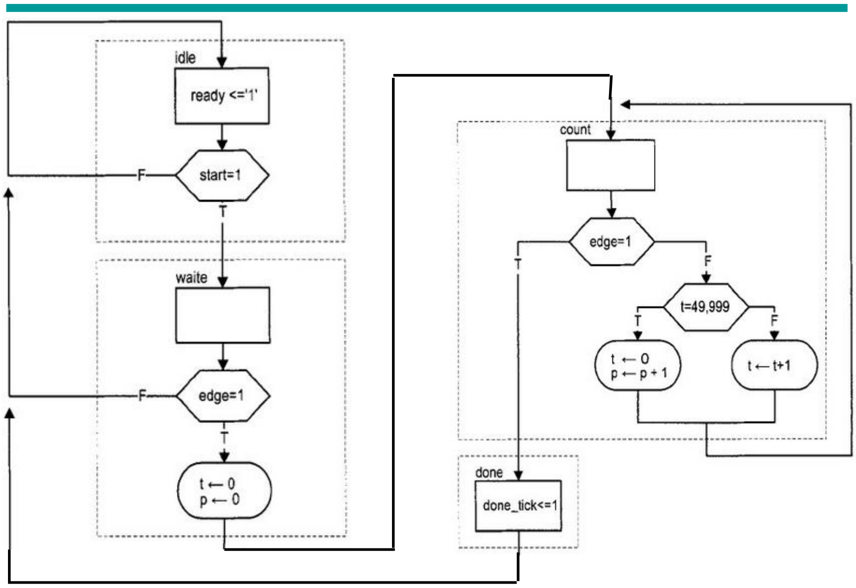
Example 1: Period Counter

- Measure the period of a periodic input waveform

- Solution:
  - Count the number of clock cycles between two rising edges of the input signal
  - Use a rising-edge detection circuit (discussed earlier)
  - Frequency of clock signal is known → easy to find the period of input signal: $N*1/f_{CLK}$
  - Assume: $T_{CLK} = (1/f_{CLK}) = 20$ ns
  - Register t counts for 50,000 clock cycles from 0 to 49,999 then wraps around; it takes 1ms to circulate through 50,000 cycles
  - Register p counts in terms of milliseconds

37

ASMD chart



38

VHDL code

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

entity period_counter is
   port(
      clk, reset: in std_logic;
      start, si: in std_logic;
      ready, done_tick: out std_logic;
      prd: out std_logic_vector(9 downto 0)
   );
end period_counter;

architecture arch of period_counter is

   constant CLK_MS_COUNT: integer := 50000; -- 1 ms tick
   type state_type is (idle, waite, count, done);
   signal state_reg, state_next: state_type;
   signal t_reg, t_next: unsigned(15 downto 0); -- up to 50000
   signal p_reg, p_next: unsigned(9 downto 0); -- up to 1 sec
   signal delay_reg: std_logic;
   signal edge: std_logic;
```

39

```
begin

   -- state and data register
   process(clk,reset)
   begin
      if reset='1' then
         state_reg <= idle;
         t_reg <= (others=>'0');
         p_reg <= (others=>'0');
         delay_reg <= '0';
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         t_reg <= t_next;
         p_reg <= p_next;
         delay_reg <= si;
      end if;
   end process;

   -- edge detection circuit
   edge <= (not delay_reg) and si;
```

40

```
-- FSMD next-state logic/DATAPATH operations
process(start,edge,state_reg,t_reg,t_next,p_reg)
   begin
      ready <= '0';
      done_tick <= '0';
      state_next <= state_reg;
      p_next <= p_reg;
      t_next <= t_reg;
      case state_reg is
         when idle =>
            ready <= '1';
            if (start='1') then
               state_next <= waite;
            end if;
         when waite => -- wait for the first edge
            if (edge='1') then
               state_next <= count;
               t_next <= (others=>'0');
               p_next <= (others=>'0');
            end if;
         when count =>
            if (edge='1') then   -- 2nd edge arrived
               state_next <= done;
            else -- otherwise count
               if t_reg = CLK_MS_COUNT-1 then -- 1ms tick
                  t_next <= (others=>'0');
                  p_next <= p_reg + 1;
               else
                  t_next <= t_reg + 1;
               end if;
            end if;
         when done =>
            done_tick <= '1';
            state_next <= idle;
      end case;
   end process;
   prd <= std_logic_vector(p_reg);
end arch;
```

41

Overview

- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

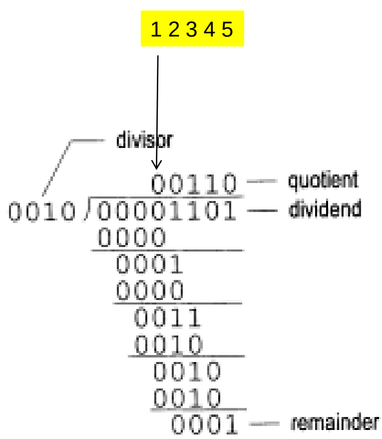
42

Example 2: Division circuit (more complex)

- Division algorithm of 4-bit unsigned integers:
  - (1) Double the dividend width - by appending 0's in front and align the divisor to leftmost bit of extended dividend
  - (2) If corresponding dividend bits are >= to divisor, subtract divisor from dividend and make corresponding quotient bit 1. Otherwise, keep original dividend bits and make quotient bit 0.
  - (3) Append one additional dividend bit to previous result and shift divisor to right 1 position
  - (4) Repeat (2) and (3) until all dividend bits are used

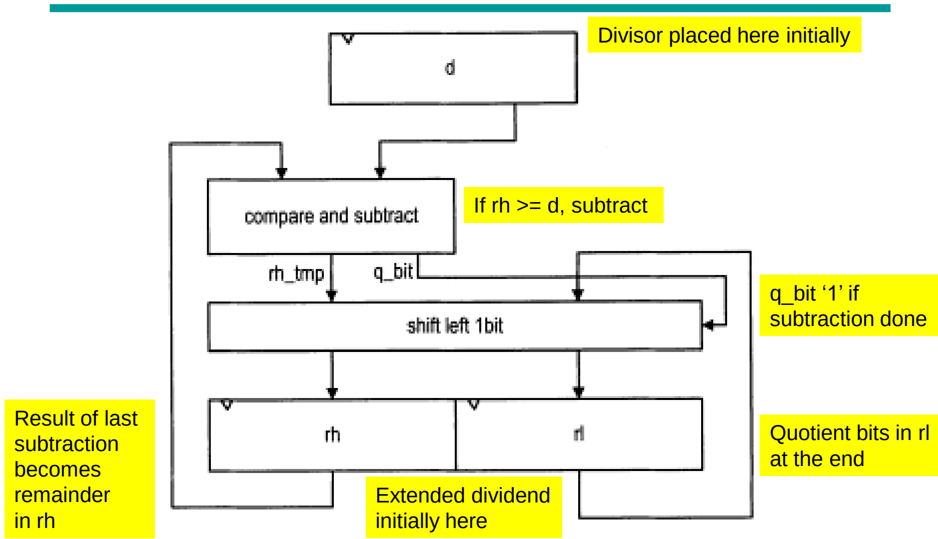
43

Division of two 4-bit unsigned integers



44

Sketch of Datapath of Division circuit



- **Assignment: draw the ASMD chart with four blocks. See Ch.6 of P.Chu's book.**

45

VHDL code

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

entity div is
    generic(
       W: integer:=8;
       CBIT: integer:=4    -- CBIT=log2(W)+1
    );
    port(
        clk, reset: in std_logic;
        start: in std_logic;
        dvsr, dvnd: in std_logic_vector(W-1 downto 0);
        ready, done_tick: out std_logic;
        quo, rmd: out std_logic_vector(W-1 downto 0)
    );
end div;

architecture arch of div is

   type state_type is (idle,op,last,done);
   signal state_reg, state_next: state_type;
   signal rh_reg, rh_next: unsigned(W-1 downto 0);
   signal rl_reg, rl_next: std_logic_vector(W-1 downto 0);
   signal rh_tmp: unsigned(W-1 downto 0);
   signal d_reg, d_next: unsigned(W-1 downto 0);
   signal n_reg, n_next: unsigned(CBIT-1 downto 0);
   signal q_bit: std_logic;
```

46

```
begin

   -- fsmd state and data registers
   process(clk,reset)
   begin
      if reset='1' then
         state_reg <= idle;
         rh_reg <= (others=>'0');
         rl_reg <= (others=>'0');
         d_reg <= (others=>'0');
         n_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         rh_reg <= rh_next;
         rl_reg <= rl_next;
         d_reg <= d_next;
         n_reg <= n_next;
      end if;
   end process;
```

47

```
-- fsmd next-state logic and data path logic
   process(state_reg,n_reg,rh_reg,rl_reg,d_reg,
           start,dvsr,dvnd,q_bit,rh_tmp,n_next)
   begin
      ready <='0';
      done_tick <= '0';
      state_next <= state_reg;
      rh_next <= rh_reg;
      rl_next <= rl_reg;
      d_next <= d_reg;
      n_next <= n_reg;
      case state_reg is
         when idle =>
            ready <= '1';
            if start='1' then
               rh_next <= (others=>'0');
               rl_next <= dvnd;                -- dividend
               d_next <= unsigned(dvsr);       -- divisor
               n_next <= to_unsigned(W+1, CBIT); -- index
               state_next <= op;
            end if;
         when op =>
            -- shift rh and rl left
            rl_next <= rl_reg(W-2 downto 0) & q_bit;
            rh_next <= rh_tmp(W-2 downto 0) & rl_reg(W-1);
            --decrease index
            n_next <= n_reg - 1;
            if (n_next=1) then
               state_next <= last;
            end if;
         when last =>  -- last iteration
            rl_next <= rl_reg(W-2 downto 0) & q_bit;
            rh_next <= rh_tmp;
            state_next <= done;
         when done =>
            state_next <= idle;
            done_tick <= '1';
      end case;
   end process;
```

48

```
   -- compare and subtract
   process(rh_reg, d_reg)
   begin
      if rh_reg >= d_reg then
         rh_tmp <= rh_reg - d_reg;
         q_bit <= '1';
      else
         rh_tmp <= rh_reg;
         q_bit <= '0';
      end if;
   end process;

   -- output
   quo <= rl_reg;
   rmd <= std_logic_vector(rh_reg);

end arch;
```

49

Overview

- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

50

Example 3: Binary-to-BCD converter

- A decimal number is represented as sequence of 4-bit BCD digits
- Conversion can be processed by a special BCD register, which is divided into 4-bit groups internally
- Shifting a BCD sequence to left requires adjustment of if a BCD digit is > 9_{10} after shifting
  - Example: If a BCD sequence is “0001 0111” (i.e., 17_{10}), it should become “0011 0100” (i.e., 34_{10}) rather than “0010 1110”
- Example:
  - Binary: 001000000000
  - BCD: 0101 0001 0010
  - Decimal: 5 1 2
- **Read section 6.3.3 in P.Chu’s book for details on algorithm**
- **Assignment: Draw a sketch of the datapath. Draw the ASMD chart.**

VHDL code

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

entity bin2bcd is
   port(
      clk: in std_logic;
      reset: in std_logic;
      start: in std_logic;
      bin: in std_logic_vector(12 downto 0);
      ready, done_tick: out std_logic;
      bcd3,bcd2,bcd1,bcd0: out std_logic_vector(3 downto 0)
   );
end bin2bcd ;

architecture arch of bin2bcd is
   type state_type is (idle, op, done);
   signal state_reg, state_next: state_type;
   signal p2s_reg, p2s_next: std_logic_vector(12 downto 0);
   signal n_reg, n_next: unsigned(3 downto 0);
   signal bcd3_reg,bcd2_reg,bcd1_reg,bcd0_reg: unsigned(3 downto 0);
   signal bcd3_next,bcd2_next,bcd1_next,bcd0_next: unsigned(3 downto 0);
   signal bcd3_tmp,bcd2_tmp,bcd1_tmp,bcd0_tmp: unsigned(3 downto 0);
```

```
begin

   -- state and data registers
   process (clk,reset)
   begin
      if reset='1' then
         state_reg <= idle;
         p2s_reg <= (others=>'0');
         n_reg <= (others=>'0');
         bcd3_reg <= (others=>'0');
         bcd2_reg <= (others=>'0');
         bcd1_reg <= (others=>'0');
         bcd0_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
         p2s_reg <= p2s_next;
         n_reg <= n_next;
         bcd3_reg <= bcd3_next;
         bcd2_reg <= bcd2_next;
         bcd1_reg <= bcd1_next;
         bcd0_reg <= bcd0_next;
      end if;
   end process;
```

53

```
-- fsmd next-state logic / data path operations
  process(state_reg,start,p2s_reg,n_reg,n_next,bin,
          bcd0_reg,bcd1_reg,bcd2_reg,bcd3_reg,
          bcd0_tmp,bcd1_tmp,bcd2_tmp,bcd3_tmp)
  begin
     state_next <= state_reg;
     ready <= '0';
     done_tick <= '0';
     p2s_next <= p2s_reg;
     bcd0_next <= bcd0_reg;
     bcd1_next <= bcd1_reg;
     bcd2_next <= bcd2_reg;
     bcd3_next <= bcd3_reg;
     n_next <= n_reg;
     case state_reg is
        when idle =>
           ready <= '1';
           if start='1' then
              state_next <= op;
              bcd3_next <= (others=>'0');
              bcd2_next <= (others=>'0');
              bcd1_next <= (others=>'0');
              bcd0_next <= (others=>'0');
              n_next <="1101";  -- index
              p2s_next <= bin;  -- input shift register
              state_next <= op;
           end if;
        when op =>
           -- shift in binary bit
           p2s_next <= p2s_reg(11 downto 0) & '0';
           -- shift 4 BCD digits
           bcd0_next <= bcd0_tmp(2 downto 0) & p2s_reg(12);
           bcd1_next <= bcd1_tmp(2 downto 0) & bcd0_tmp(3);
           bcd2_next <= bcd2_tmp(2 downto 0) & bcd1_tmp(3);
           bcd3_next <= bcd3_tmp(2 downto 0) & bcd2_tmp(3);
           n_next <= n_reg - 1;
           if (n_next=0) then
               state_next <= done;
           end if;
        when done =>
           state_next <= idle;
           done_tick <= '1';
    end case;
  end process;
```

54

```
   -- data path function units
   bcd0_tmp <= bcd0_reg + 3 when bcd0_reg > 4 else
               bcd0_reg;
   bcd1_tmp <= bcd1_reg + 3 when bcd1_reg > 4 else
               bcd1_reg;
   bcd2_tmp <= bcd2_reg + 3 when bcd2_reg > 4 else
               bcd2_reg;
   bcd3_tmp <= bcd3_reg + 3 when bcd3_reg > 4 else
               bcd3_reg;

   -- output
   bcd0 <= std_logic_vector(bcd0_reg);
   bcd1 <= std_logic_vector(bcd1_reg);
   bcd2 <= std_logic_vector(bcd2_reg);
   bcd3 <= std_logic_vector(bcd3_reg);

end arch;
```

55

Overview

- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

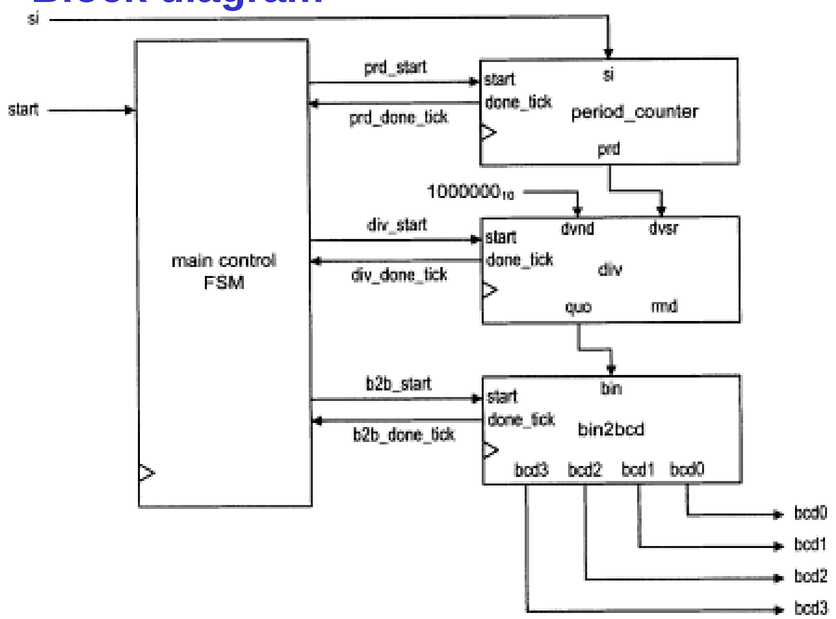
56

Example 4: Accurate low-frequency counter

- Measure frequency of a periodic input waveform
- One way:
  - Count number of input pulses in a fixed amount of time, say 1 sec
  - Not working for low-frequency signals; example 2 Hz
- Another way:
  1. Measure period of signal
  2. Take reciprocal (f=1/T)
  3. Convert binary number to BCD format
- Assume: frequency of input signal is between 1-10 Hz (T = 100...1000 ms)
- Structural description:
  - Instantiate a period counter, a division circuit, and a binary-3-BCD converter
  - Create a new ASM chart for master control

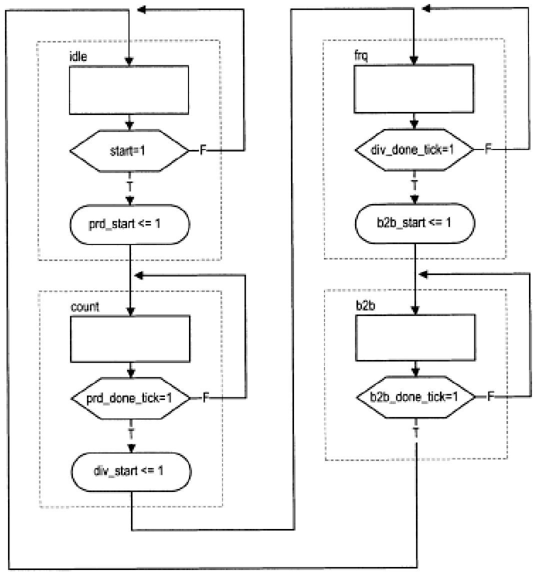
57

Block diagram



58

ASM chart of master control



VHDL code

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

entity low_freq_counter is
    port(
        clk, reset: in std_logic;
        start: in std_logic;
        si: in std_logic;
        bcd3, bcd2, bcd1, bcd0: out std_logic_vector(3 downto 0)
    );
end low_freq_counter;

architecture arch of low_freq_counter is

   type state_type is (idle, count, frq, b2b);
   signal state_reg, state_next: state_type;
   signal prd: std_logic_vector(9 downto 0);
   signal dvsr, dvnd, quo: std_logic_vector(19 downto 0);
   signal prd_start, div_start, b2b_start: std_logic;
   signal prd_done_tick, div_done_tick, b2b_done_tick: std_logic;

begin
```

```
--==============================================
-- component instantiation
--==============================================

-- instantiate period counter
prd_count_unit: entity work.period_counter
port map(clk=>clk, reset=>reset, start=>prd_start, si=>si,
         ready=>open, done_tick=>prd_done_tick, prd=>prd);

-- instantiate division circuit
div_unit: entity work.div
generic map(W=>20, CBIT=>5)
port map(clk=>clk, reset=>reset, start=>div_start,
         dvsr=>dvsr, dvnd=>dvnd, quo=>quo, rmd=>open,
         ready=>open, done_tick=>div_done_tick);

-- instantiate binary-to-BCD convertor
bin2bcd_unit: entity work.bin2bcd
port map
   (clk=>clk, reset=>reset, start=>b2b_start,
    bin=>quo(12 downto 0), ready=>open,
    done_tick=>b2b_done_tick,
    bcd3=>bcd3, bcd2=>bcd2, bcd1=>bcd1, bcd0=>bcd0);

-- signal width extension
dvnd <= std_logic_vector(to_unsigned(1000000, 20));
dvsr <= "0000000000" & prd;
```

61

```
--==============================================
-- Master FSM
--==============================================
process(clk,reset)
begin
   if reset='1' then
      state_reg <= idle;
   elsif (clk'event and clk='1') then
      state_reg <= state_next;
   end if;
end process;
```

62

```
   process(state_reg,start,
           prd_done_tick,div_done_tick,b2b_done_tick)
   begin
      state_next <= state_reg;
      prd_start <='0';
      div_start <='0';
      b2b_start <='0';
      case state_reg is
         when idle =>
            if start='1' then
               state_next <= count;
               prd_start <='1';
            end if;
         when count =>
            if (prd_done_tick='1') then
               div_start <='1';
               state_next <= frq;
            end if;
         when frq =>
            if (div_done_tick='1') then
               b2b_start <='1';
               state_next <= b2b;
            end if;
         when b2b =>
            if (b2b_done_tick='1') then
               state_next <= idle;
            end if;
       end case;
   end process;

end arch;
```

63

Overview

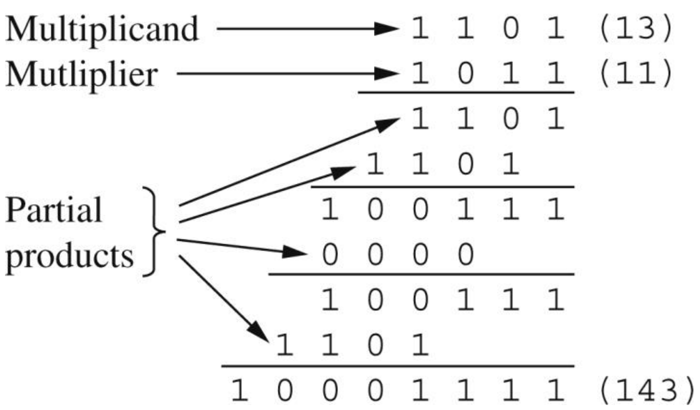
- Finite State Machine (FSM) Representations:
  1. State Graphs
  2. Algorithmic State Machine (ASM) Charts
- Finite State Machines with Datapath (FSMD)
- Algorithmic State Machine with Datapath (ASMD)

- Examples
  - Example 1 – period counter
  - Example 2 – division circuit
  - Example 3 – binary-2-BCD converter
  - Example 4 – low-frequency counter
  - Example 5 – multiplier

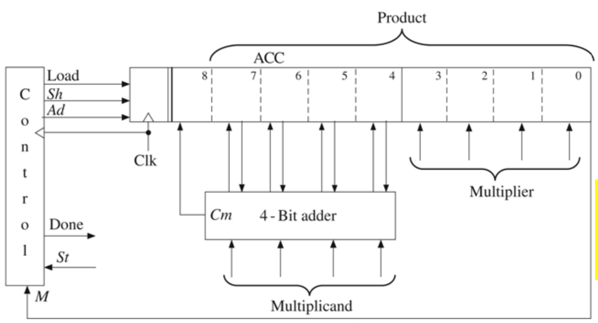
64

Example 5: Add-and-shift Multiplier

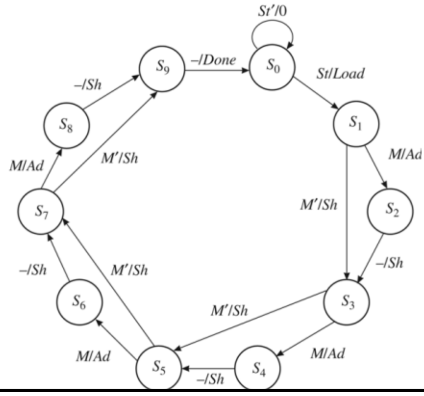
- Multiplier for unsigned numbers
- See Section 4.8 in C.H.Roth book



65



Add multiplicand with most Significant 4 bits of ACC. Put result back into ACC. Cout into bit index 8 of ACC.



State graph of control

66

Summary

- State graphs and ASM charts are graphical models for FSMs
- ASM charts are somewhat more convenient to write VHDL code from
- Finite State Machines with Datapath, FSMDs (particularly, Algorithmic State Machine with Datapath, ASMD – as a form of FSMD) are great for RT design methodology
- They are useful when we care about the internal structure of the circuit
- Try to reuse developed components to implement larger designs

67

Credits

- Chapters 5,6 of Pong P. Chu, FPGA Prototyping by VHDL, 2008
- Chapter 5 from Jr. Charles H. Roth and Lizy K. John, Digital Systems Design Using VHDL, 2007.

68