Introduction to Structured VLSI Design

- FSMD's

Joachim Rodrigues

Outline

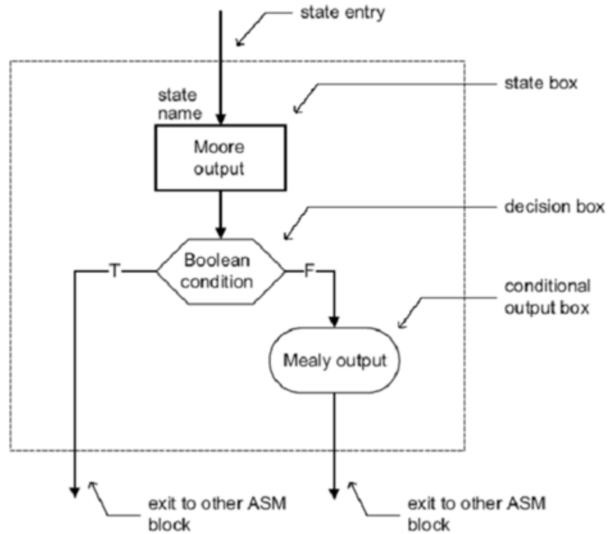
- ASM
- Overview of FSMD
- FSMD design of a repetitive-addition multiplier
- VHDL Examples

ASM (algorithmic state machine)

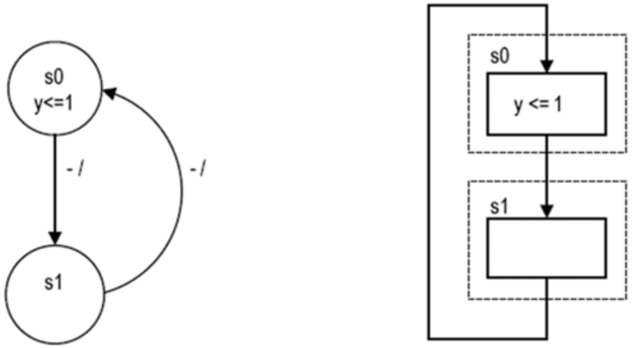
ASM (algorithmic state machine) chart

- Flowchart-like diagram
- Provide the same info as an FSM
- More descriptive, better for complex description
- ASM block
  - One state box
  - One ore more optional decision boxes: with T or F exit path
  - One or more conditional output boxes: for Mealy output

ASM block

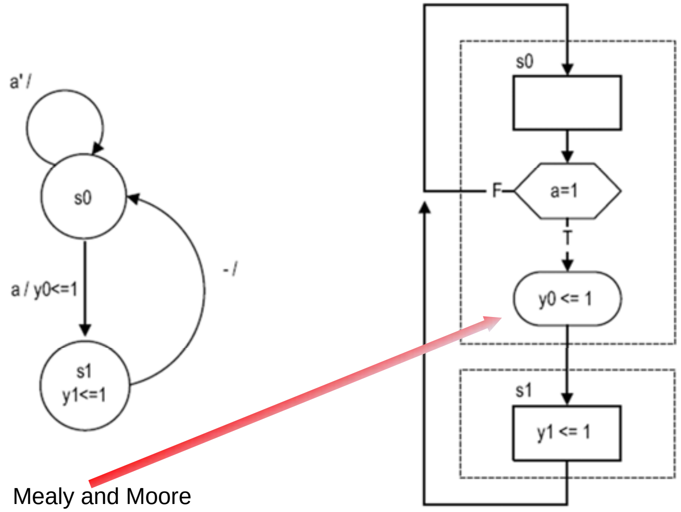


State Diagram and ASM Chart conversion



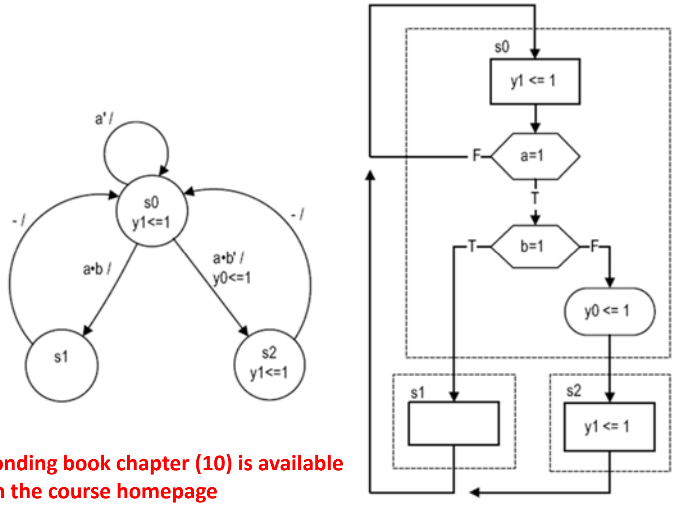
Moore

State Diagram and ASM Chart conversion



Mealy and Moore

State Diagram and ASM Chart conversion



Corresponding book chapter (10) is available as pdf on the course homepage

Introduction

- How to realize an algorithm in hardware?
- Two characteristics of an algorithm:
  - Use of variables (symbolic memory location) e.g., n = n + 1 in C
  - Sequential execution (execution order is important)

Algorithm

- Summate 4 numbers
- Divide the result by 8
- Round the result

- Pseudocode

```
size = 4
sum = 0;
for i in (0 to size-1) do {
   sum = sum + a(i);}
q = sum / 8;
r = sum rem 8;
if (r > 3) {
   q = q+1;}
outp = q;
```


HDL Implementation

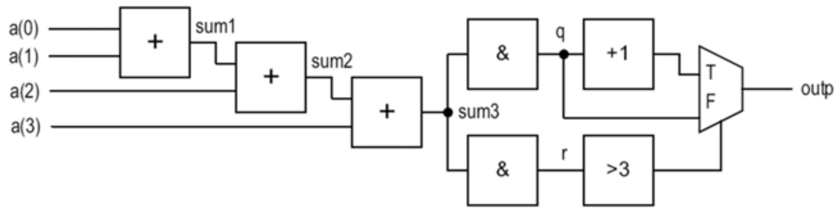
- "Dataflow" implementation in VHDL
  - Convert the algorithm in to combinational circuit
  - No memory elements
  - The sequence is embedded into the "flow of data"

```
sum  <= 0;
sum0  <= a(0);
sum1  <= sum0 + a(1);
sum2  <= sum1 + a(2);
sum3  <= sum2 + a(3);
q  <= "000" & sum3(8 downto 3);
r  <= "00000" & sum3(2 downto 0);

outp  <= q + 1 when (r > 3) else
         q;
```

The instructions are "sequential", BUT...

HW-mapping

... instructions will be executed concurrently



Dataflow implementation- drawbacks

- Problems with dataflow implementation:
  - Can only be applied to trivial algorithm
  - Not flexible
    - Can we just share one adder in a time-multiplexing fashion to save hardware resources
    - What happen if input size is not fixed

    (i.e., size is determined by an external input)

Register Transfer Methodology

- Realized algorithm in hardware
- Use register to store intermediate data and imitate variable
- Use a data path to realize all register operations
- Use a control path (FSM) to specify the order of register operation
- The system is specified as sequence of data manipulation/transfer among registers
- Realized by FSM with a data path (FSMD)

Basic RT operation

- Basic form:

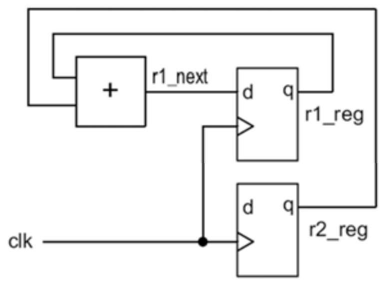
$$r_{dest} \leftarrow f(r_{src1}, r_{src2}, \ldots, r_{srcn})$$

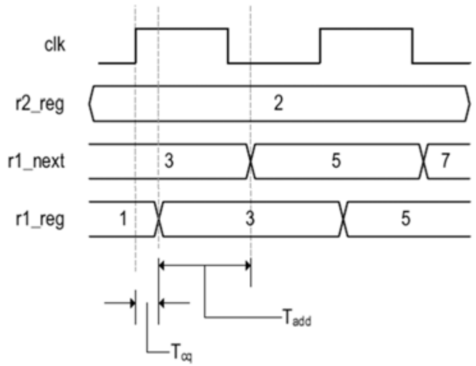
- Interpretation:
  - At the rising edge of the clock, the output of registers r_{src1} r_{src2} etc. are available.
  - The output are passed to a combinational circuit that performs *f( ).*
  - At the **next rising edge** of the clock, the result is stored into r_{dest}

$$r \leftarrow 1$$
$$r \leftarrow r$$
$$r0 \leftarrow r1$$
$$n \leftarrow n - 1$$
$$y \leftarrow a \oplus b \oplus c \oplus d$$
$$s \leftarrow a^2 + b^2$$

Implementation example

$$r1 \leftarrow r1+r2$$

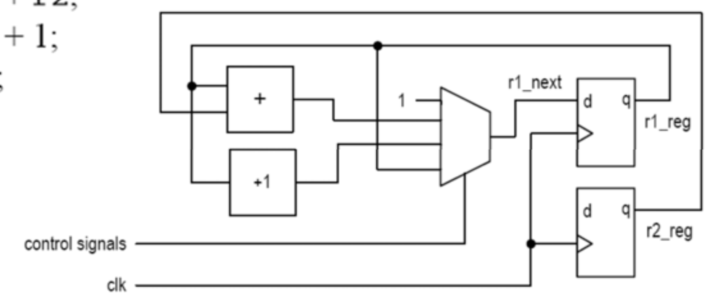




Implementation example

- Multiple RT operations

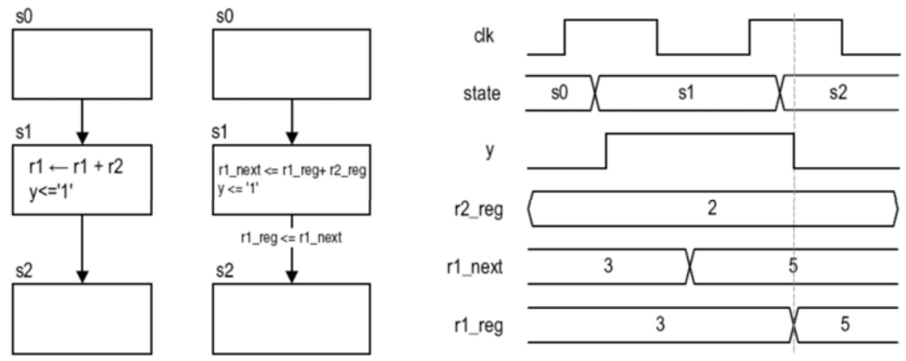
$$r1 \leftarrow 1;$$
$$r1 \leftarrow r1 + r2;$$
$$r1 \leftarrow r1 + 1;$$
$$r1 \leftarrow r1;$$



FSM as control path

- FSM is a good to control RT operation
  – State transition is on clock-by-clock basis
  – FSM can enforce order of execution
  – FSM allows branches on execution sequence
- Normally represented in an extended ASM chart known as ASMD (ASM with datapath) chart

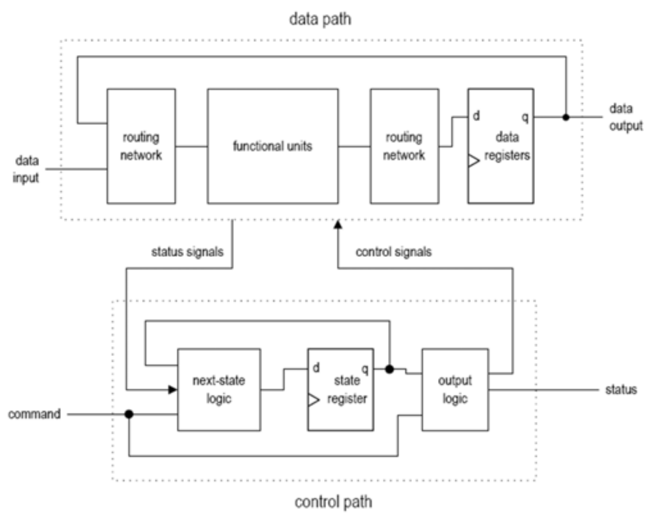
FSM timing



(a) (b) (c) Timing diagram

Note: new value of r1 is only available when the FSM enters s2 state

Basic Block Diagram of FSMD



FSMD design example

Repetitive addition multiplier

- Basic algorithm: 7*5 = 7+7+7+7+7

```
if (a_in=0 or b_in=0) then {
   r = 0;}
else{
   a = a_in;
   n = b_in;
   r = 0;
   while (n != 0 ){
      r = r + a;
      n = n-1;}
}
return(r)
```

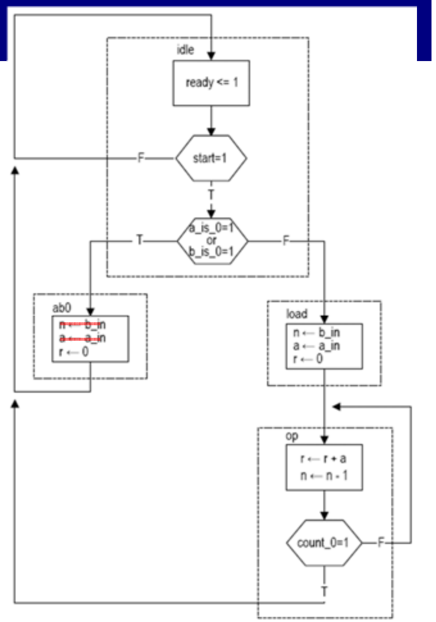
Pseudo code

```
      if (a_in=0 or b_in=0) then {
         r = 0;}
      else{
         a = a_in;
         n = b_in;
         r = 0;
op:      r = r + a;
         n = n-1;
         if (n = 0) then{
            goto stop;}
         else{
            goto op;}
      }
stop: return(r);
```

ASMD-friendly code

ASMD Chart

- Input:
  - a_in, b_in: 8-bit unsigned
  - clk, reset
  - start: command
- Output:
  - r: 16-bit unsigned
  - ready: status
- ASMD chart
  - Default RT operation: keep the previous value
  - Note the parallel execution in op state



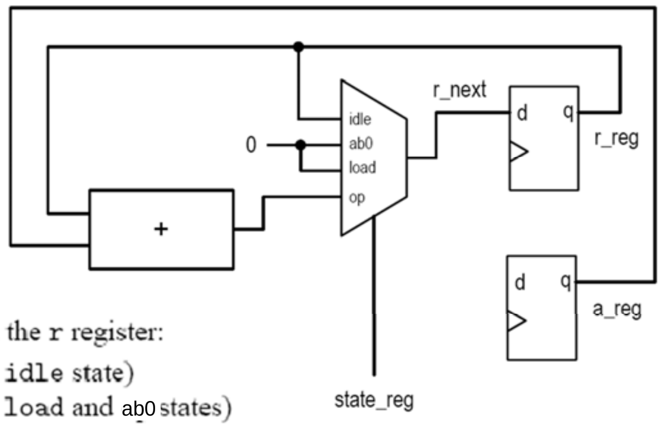
Recipe

- Construction of the data path
  - List all RT operations
  - Group RT operation according to the destination register
  - Add combinational circuit/mux
  - Add status circuits

- RT operations with the r register:
  - r ← r (in the idle state)
  - r ← 0 (in the load and op states)
  - r ← r + b (in the op state)
- RT operations with the n register:
  - n ← n (in the idle state)
  - n ← a_in (in the load and ab0 states)
  - n ← n - 1 (in the op state)
- RT operations with the b register:
  - b ← b (in the idle and op states)
  - b ← b_in (in the load and ab0 states)

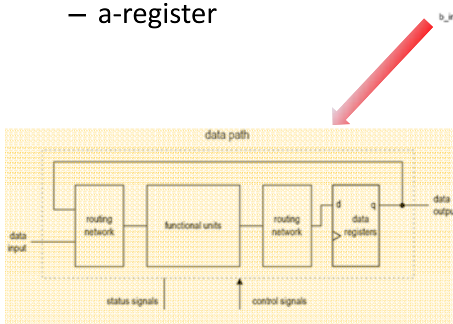
Register in datapth

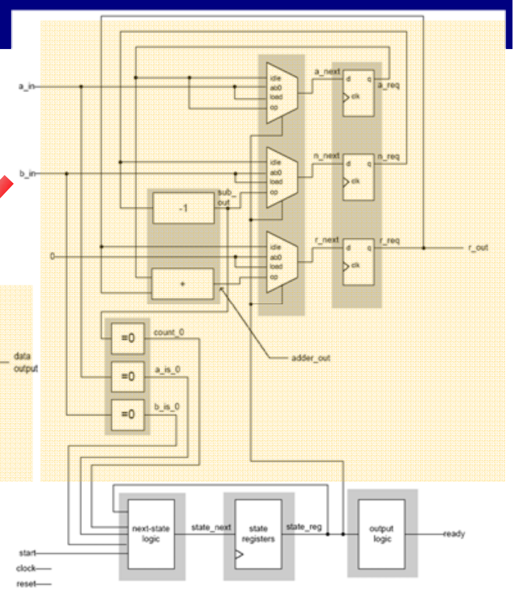
Circuit associated with **r register**



- RT operations with the r register:
  - r ← r (in the idle state)
  - r ← 0 (in the load and ab0 states)
  - r ← r + b (in the op state)

- Continue with
  - n-register
  - a-register





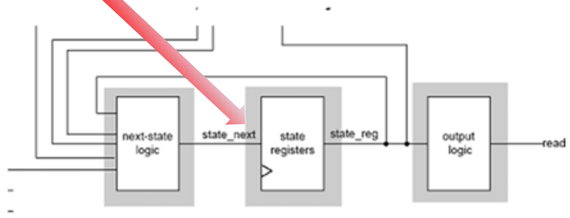
Entity

- VHDL code: follow the block diagram

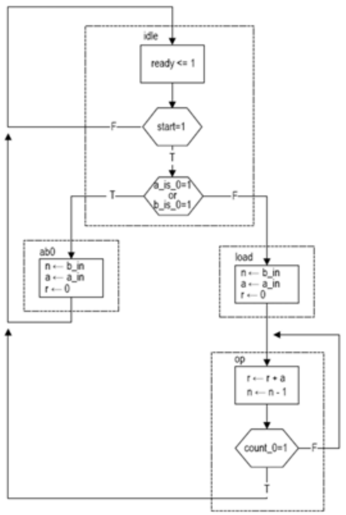
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity seq_mult is
   port(
      clk, reset: in std_logic;
      start: in std_logic;
      a_in, b_in: in std_logic_vector(7 downto 0);
      ready: out std_logic;
      r: out std_logic_vector(15 downto 0)
      );
end seq_mult;
```


Sequential Process (state registers)

```
-- control path: state register
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;
```



FSM Implementation

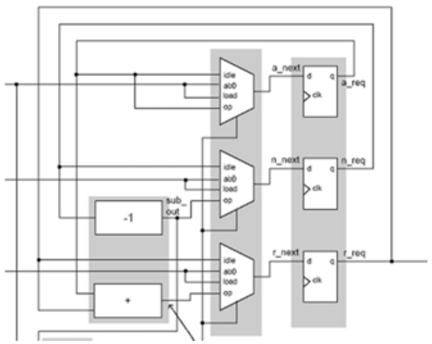



```
-- control path: next-state/output logic
process(state_reg,start,a_is_0,b_is_0,count_0
begin
   case state_reg is
      when idle =>
         if start='1' then
            if (a_is_0='1' or b_is_0='1') the
               state_next <= ab0;
            else
               state_next <= load;
            end if;
         else
            state_next <= idle;
         end if;
      when ab0 =>
         state_next <= idle;
      when load =>
         state_next <= op;
      when op =>
         if count_0='1' then
            state_next <= idle;
         else
            state_next <= op;
         end if;
   end case;
end process;
```


Sequential Process (data registers)

```
-- control path: output logic
ready <= '1' when state_reg=idle else '0';
-- data path: data register
process(clk,reset)
begin
   if reset='1' then
      a_reg <= (others=>'0');
      n_reg <= (others=>'0');
      r_reg <= (others=>'0');
   elsif (clk'event and clk='1') then
      a_reg <= a_next;
      n_reg <= n_next;
      r_reg <= r_next;
   end if;
end process;
```

Multiplexer Routing



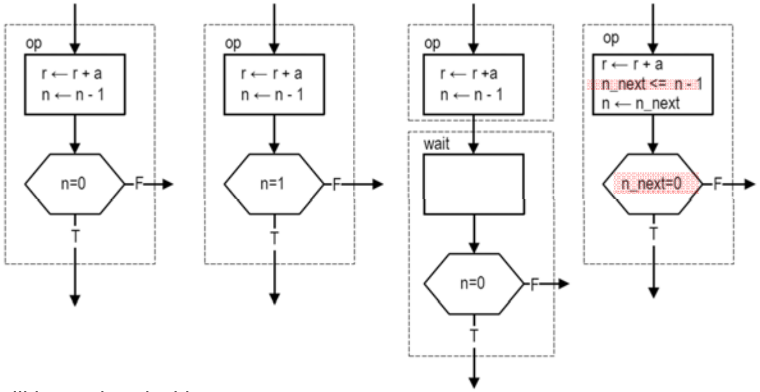

```
-- data path: routing multiplexer
process(state_reg,a_reg,n_reg,r_reg,
        a_in,b_in,adder_out,sub_out)
begin
   case state_reg is
      when idle =>
         a_next <= a_reg;
         n_next <= n_reg;
         r_next <= r_reg;
      when ab0 =>
         a_next <= unsigned(a_in);
         n_next <= unsigned(b_in);
         r_next <= (others=>'0');
      when load =>
         a_next <= unsigned(a_in);
         n_next <= unsigned(b_in);
         r_next <= (others=>'0');
      when op =>
         a_next <= a_reg;
         n_next <= sub_out;
         r_next <= adder_out;
   end case;
end process;
```


Functional Units

```
-- data path: functional units
adder_out <= ("00000000" & a_reg) + r_reg;
sub_out <= n_reg - 1;

-- data path: status
a_is_0 <= '1' when a_in="00000000" else '0';
b_is_0 <= '1' when b_in="00000000" else '0';
count_0 <= '1' when n_next="00000000" else '0';

-- data path: output
r <= std_logic_vector(r_reg);
```


Register in decision box

- Register is updated when the FSM exits current state
- How to represent count_0='1' using register?



n will be updated with the **next** clock

One extra clock required

?