Descrizione di macchine a stati finiti (FSM) tramite VHDL

M. Favalli

DE: Department of Engineering Ferrara

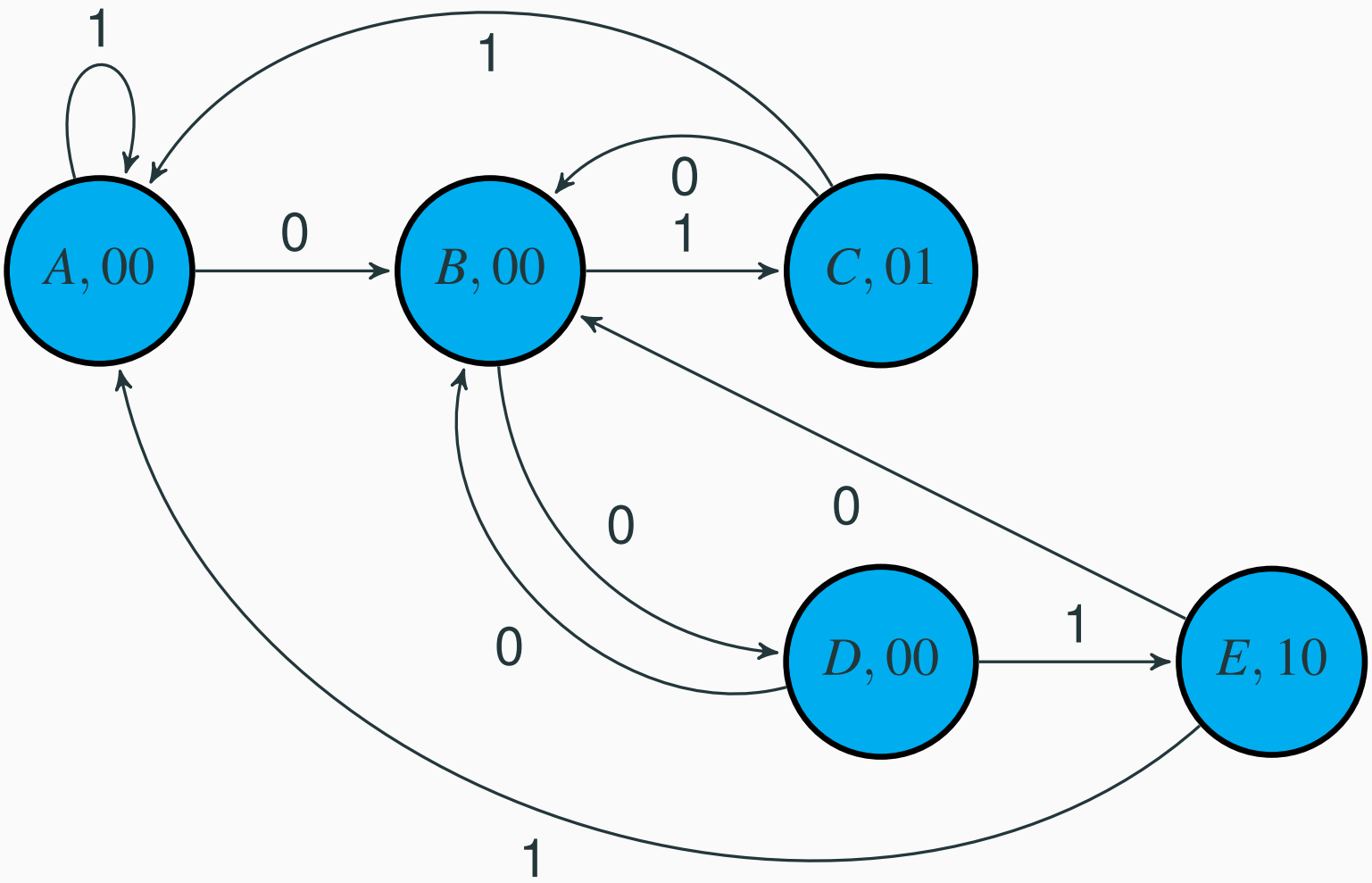
Riepilogo su FSM

- FSM: i) insieme finito di simboli di ingresso; ii) insieme finito di simboli di uscita; iii) un insieme finito di stati; iv) funzione di stato futuro; v) funzione di uscita; vi) stato iniziale;
- Formalismi per la progettazione: grafo di transizione dello stato STG e State Table
- Modello per l'implementazione: Huffman
- Modello per le transizioni di stato: sincrono
- Le FSM possono essere descritte direttamente dal progettista o essere estratte da descrizioni ad alto livello

Esempio di FSM (Moore)

- Automa con un ingresso x e due uscite y, w
- Riconosce le sequenze di ingresso 01 e 001 (non sovrapponibili) producendo le uscite 01 e 10
- Quando non viene riconosciuto alcun simbolo, le uscite valgono 00

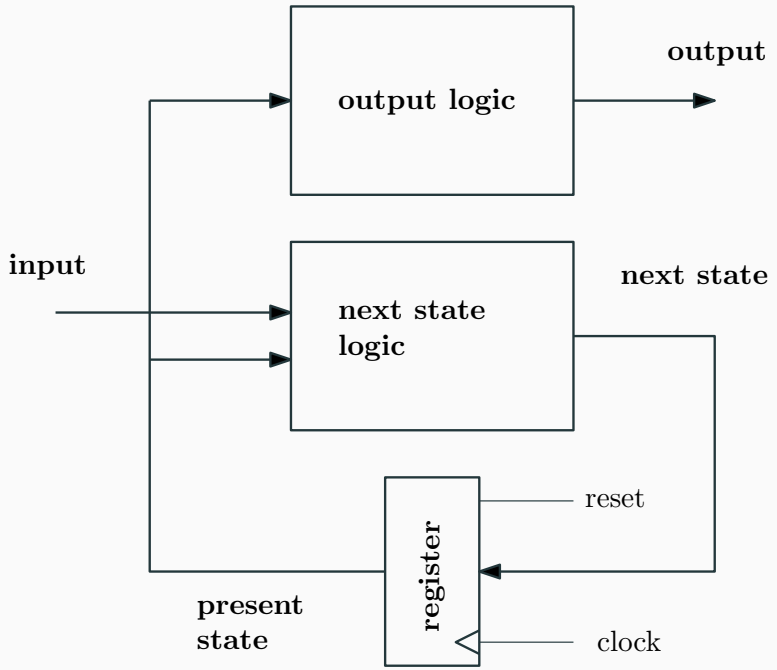
Esempio di FSM (Moore)



FSM in VHDL

Descrizione VHDL

- Non corrisponde esattamente al modello di FSM: clock e reset
- Presenta alcuni elementi del modello di Huffman (struttura)
- Modello simulabile e sintetizzabile
- Tecnica di descrizione *multi-segment*



VHDL (I): entity and symbolic state declarations

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm is
  port(x,reset: in std_logic;
       clk: in std_logic;
       y,w: out std_logic);
end entity fsm;

architecture msegmnt of fsm is
type state is (A,B,C,D,E);
signal state_curr, state_next: state;
```

VHDL (II): reset and state update process

```
begin

-- state reg. (asynchr. reset)

process(clk,reset)
begin
  if (reset='1') then
    state_curr <= A;
  elsif (rising_edge(clk)) then
    state_curr <= state_next;
  end if;
end process;
```

VHDL (III): next-state process

```
process(state_curr,x)
begin
  case state_curr is
    when A => if (x='1') then
                state_next <= A;
              elsif (x='0') then
                state_next <= B;
              end if;
    when B => if (x='1') then
                state_next <= C;
              elsif (x='0') then
                state_next <= D;
              end if;
    when C => if (x='1') then
                state_next <= A;
              elsif (x='0') then
                state_next <= B;
              end if;
    when D => if (x='1') then
                state_next <= E;
              elsif (x='0') then
                state_next <= B;
              end if;
    when E => if (x='1') then
                state_next <= A;
              elsif (x='0') then
                state_next <= B;
              end if;
  end case;
end process;
```

VHDL (IV): output process

```
-- Moore output

process(state_curr)
begin
  case state_curr is
    when A => y<='0';
              w<='0';
    when B => y<='0';
              w<='0';
    when C => y<='0';
              w<='1';
    when D => y<='0';
              w<='0';
    when E => y<='1';
              w<='0';
  end case;
end process;
end architecture msegmnt;
```

Sintesi e ottimizzazione di FSM

- Una FSM puó essere sintetizzata e ottimizzata attraverso i seguenti passi:
  - minimizzazione del numero di stati
  - assegnamento di una codifica binaria allo stato
  - sintesi e ottimizzazione delle reti che realizzano stato futuro e uscita (come reti a 2 livelli o multilivello)
  - scelta della frequenza di clock
- Questi passi vengono tipicamente svolti in maniera automatica

EFSM

Extended Finite State Machines - EFSM

- Alcune FSM presentano un numero molto grande e non gestibile esplicitamente di stati
  - un semplice contatore binario realizzato con n flip-flop ha 2^n stati (é una FSM il cui stato codifica un numero binario s realizzando la relazione di stato futuro $s^{k+1} = s^k + 1$)
  - in generale non é possibile gestire esplicitamente lo stato di macchine che utilizzano registri
- Una EFSM é una generalizzazione del concetto di FSM
- Permette di elevare il livello di astrazione nella descrizione di reti sincrone, ottenendo descrizioni piú compatte di quelle basate su FSM

Extended Finite State Machines - EFSM

- Una FSM (Mealy) calcola l'uscita e lo stato futuro sulla base di ingresso e stato presente
- In una EFSM, a ingresso e uscita vengono aggiunte condizioni (guard) e azioni (action) relative a un ambiente costituito da un numero finito di registri (data)
- Tali registri rappresentano implicitamente variabili di stato della EFSM
- Le guard sono tipicamente costruite applicando operatori relazionali sui dati o comunque operatori che ritornano una condizione booleana
- Le action consistono spesso in operazioni aritmetiche o logiche sui dati

Sintesi di EFSM

- Il modello di EFSM corrisponde naturalmente al paradigma di progetto basato su data-path e controllo
- Il data-path é costituito da registri, multiplexer, blocchi logici e aritmetici
- Il controllo é una FSM convenzionale che interagisce con l'ambiente esterno alla EFSM tramite gli ingressi e le uscite della EFSM, e con il data-path tramite:
  - segnali di uscita che controllano il data-path (determinati dalle action)
  - segnali di ingresso dal data-path che forniscono le condizioni individuate dalle guard

Esempio

- Implementazione via hardware di un semplice algoritmo algebrico (Euclide) che calcola il massimo comun divisore di due interi senza segno
- Prima specifica al livello behavioral in VHDL
  - simulabile
  - non sintetizzabile direttamente (ciclo unbounded)
  - nessuna informazione sul timing e sull'interfaccia con l'esterno
  - nessuna informazione sul tipo di realizzazione
  - nessuna informazione sulla sincronizzazione dei dati (sincrono/asincrono)

gcd - descrizione ad alto livello

```
-- high-level description of a gcd evaluator, no timing, descr
-- non directly synthesizable
library ieee;
use ieee.std_logic_1164.all,ieee.numeric_std.all;
entity gcd is
   port(
     a : in std_logic_vector(7 downto 0);
     b : in std_logic_vector(7 downto 0);
     gcd : out std-logic_vector(7 downto 0)
       );
end gcd;
architecture behav of gcd is
begin
```

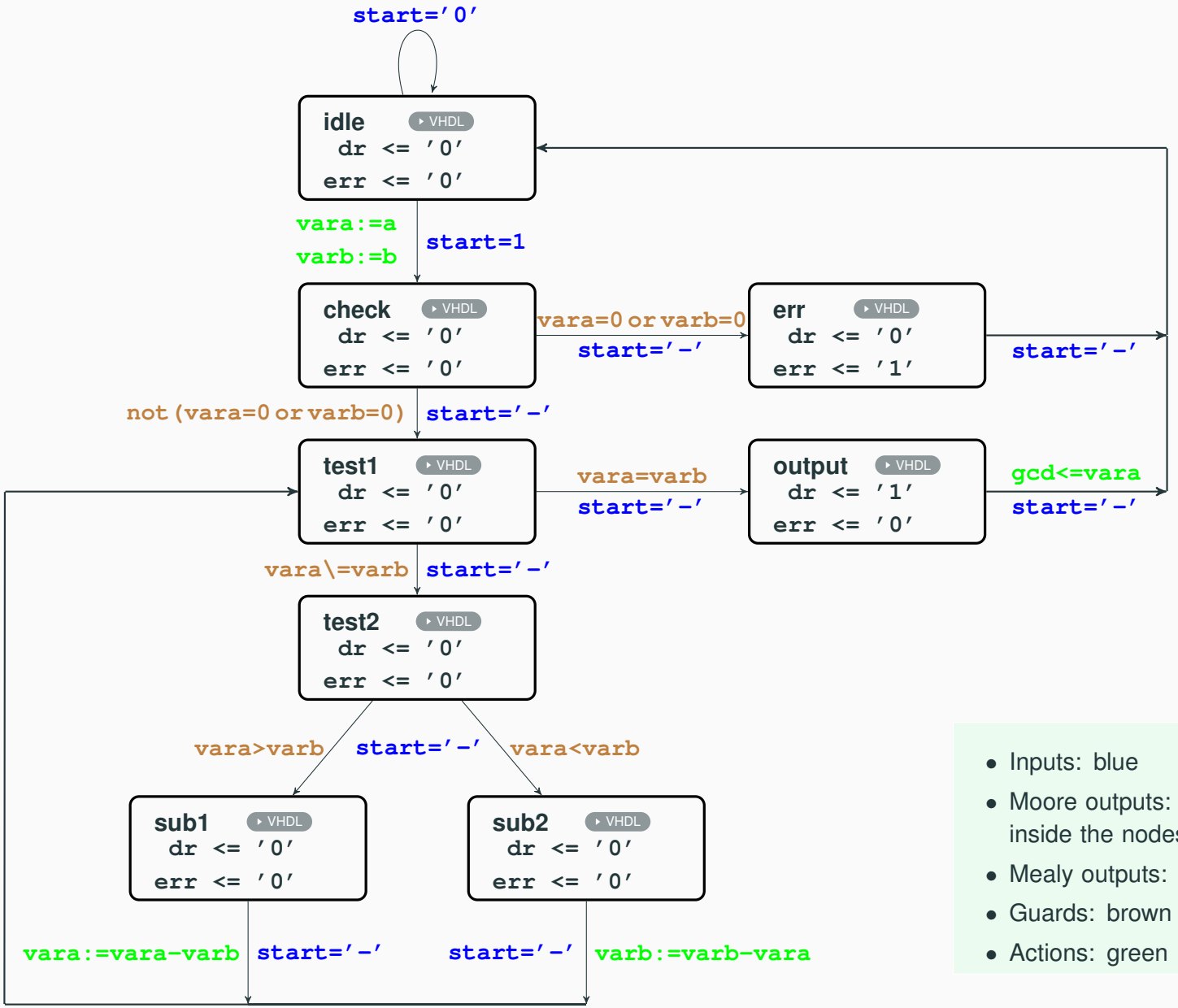
gcd - descrizione ad alto livello

```
process(a,b)
variable vara,varb: unsigned(7 downto 0);
constant zero: std_logic_vector(7 downto 0):=(others=>'0');
begin
  if (a=zero) or (b=zero) or (is_x(a)) or (is_x(b)) then
    gcd <= (others=>'X');
  else
    vara:=unsigned(a);
    varb:=unsigned(b);
    while (vara/=varb) loop
      if (vara<varb) then
        varb:=varb-vara;
      else
        vara:=vara-varb;
      end if;
    end loop;
    gcd <= std_logic_vector(vara);
  end if;
end process;                                    end architecture behav;
```

Descrizione come EFSM

- L'algoritmo puó essere descritto come EFSM
- Si sceglie l'implementazione sincrona
- Si definisce un protocollo di comunicazione con l'esterno basato su segnali di start, data-ready e di error
- Le operazioni da svolgere vengono assegnate agli stati di una EFSM definendo un controllo e un data-path

gcd - EFSM



- Inputs: blue
- Moore outputs: black text inside the nodes
- Mealy outputs: not used
- Guards: brown
- Actions: green

gcd - codice VHDL della EFSM


```
-- extended FSM version of the gcd evaluator,
-- synthesizable with a few adjustments
library ieee;
use ieee.std_logic_1164.all,ieee.numeric_std.all;

entity gcd_efsm is
   port(
        start : in std_logic;
        clk : in std_logic;
        a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        err : out std_logic;
        dr : out std_logic;
        gcd : out std_logic_vector(7 downto 0)
        );
end gcd_efsm;
```

gcd - codice VHDL della EFSM


```
-- Mealy machine
-- a,b should be steady at start, further changes are ignored
-- error and data ready signal are provided for one clock cycle
architecture behav of gcd_efsm is
type states is (idle,check,test1,test2,sub1,sub2,output,err_state);
signal curr_state,next_state: states;
begin
-- next state and output logic
  p0: process(curr_state,start,a,b)
  constant zero:unsigned(7 downto 0):=(others=>'0');
  variable vara,varb: unsigned(7 downto 0);
  begin
    case curr_state is
      when idle =>
          if (start='1') then
            next_state <= check;
            vara:=unsigned(a);
            varb:=unsigned(b);
        end if;
        dr <= '0' after 1 ns;
        err <= '0' after 1 ns;
```

gcd - codice VHDL della EFSM

```
when check =>
    if (vara=zero) or (varb=zero) or
       (is_x(a)) or (is_x(b)) then
      next_state <= err_state;
      gcd <= (others=>'0');
    else
      next_state <= test1;
    end if;
    dr <= '0' after 1 ns;
    err <= '0' after 1 ns;
when test1 =>
    if (vara = varb) then
      next_state <= output;
    else
      next_state <= test2;
    end if;
    dr <= '0' after 1 ns;
    err <= '0' after 1 ns;
```

gcd - codice VHDL della EFSM


```
when test2 =>
    if (vara > varb) then
      next_state <= sub1;
    else
      next_state <= sub2;
    end if;
    dr <= '0' after 1 ns;
    err <= '0' after 1 ns;
when sub1 =>
    next_state <= test1;
    vara:=vara-varb;
    dr <= '0' after 1 ns;
    err <= '0' after 1 ns;
when sub2 =>
    next_state <= test1;
    varb:=varb-vara;
    dr <= '0' after 1 ns;
    err <= '0' after 1 ns;
```

gcd - codice VHDL della EFSM

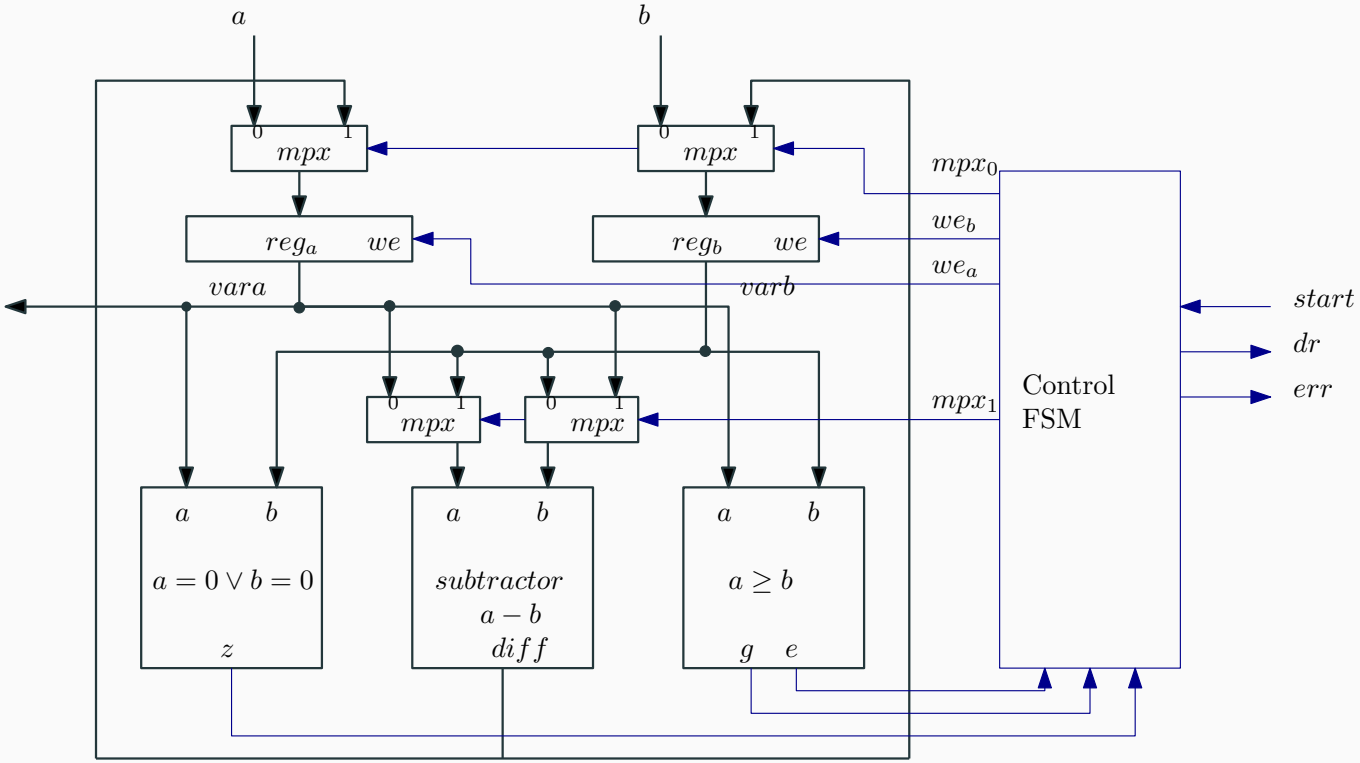

```
        when output =>
            next_state <= idle;
            dr <= '1' after 1 ns;
            err <= '0' after 1 ns;
            gcd <= std_logic_vector(vara);
        when err =>
            next_state <= idle;
            dr <= '0';
            err <= '1';
        -- useful when encoding
        when others =>
            next_state <= idle;
            dr <= '0' after 1 ns;
            err <= '1' after 1 ns;
        end case;
    end process p0;

    p1: process(clk) -- state update
        begin
        if (rising_edge(clk)) then
        curr_state <= next_state;
      end if;
    end process p1;
end architecture behav;
```

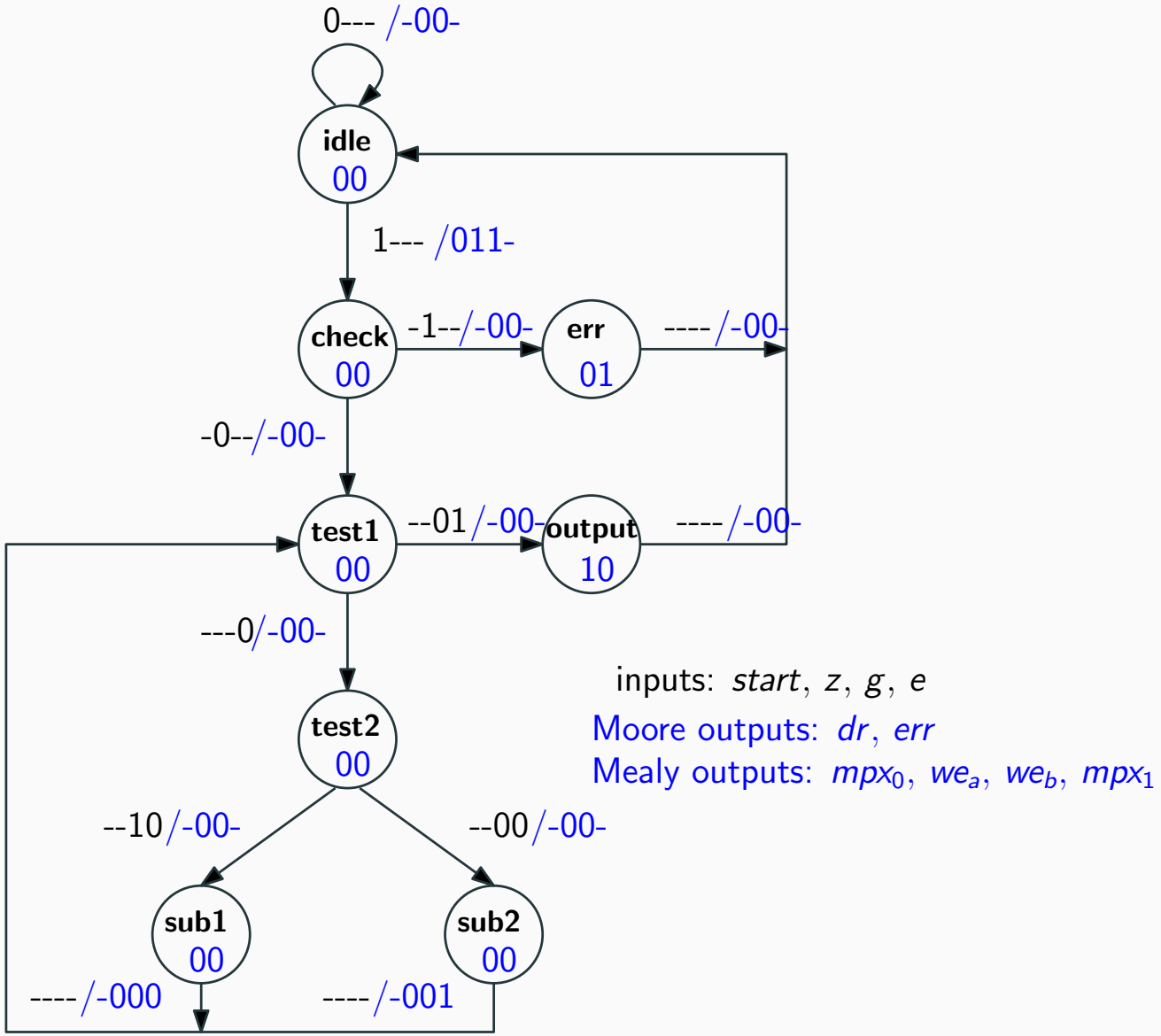
Note

- L'assegnazione delle operazioni ai diversi stati é in parte arbitraria
- Ci sono margini per l'ottimizzazione sia a partire dalla descrizione behavioral che dalla EFSM
- Vedremo in seguito algoritmi di sintesi in grado di trasformare l'EFSM (RTL comportamentale) in un RTL strutturale estraendo data-path e controllo

gcd - livello RTL strutturale



gcd - FSM che realizza il controllo



inputs: $start$, z, g, e

Moore outputs: dr, err

Mealy outputs: mpx_0, we_a, we_b, mpx_1

Contatore up-down

- Contatore binario up-down con reset sincrono (n flip-flop)
- Ingressi **`en`**, **`reset`**, **`up`** e **`clk`**
- Se **`reset='0'`** ed **`en='1'`** il contatore conta da 0 a $2^n - 1$ se **`up='1'`** e in direzione opposta se **`up='0'`**
- Lo stato del contatore é implicito, ovvero corrisponde al conteggio corrente
- É un caso particolare di EFSM con un unico stato (cui corrisponde una rete di controllo combinatoria)

Contatore up-down: codice VHDL

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

entity counter is
  generic(n: natural);
  port(en,reset,up,clk: in std_logic;
       q: out std_logic_vector(n-1 downto 0));
end entity counter;

architecture behav of counter is
begin
  process(clk)
  variable s: unsigned(n-1 downto 0);
  constant n1: unsigned:=to_unsigned(1,s'length);
  constant n0: unsigned:=to_unsigned(0,s'length);
```

Contatore up-down: codice VHDL

```
  begin
    if (rising_edge(clk)) then
      if (reset='1') then
        s:=n0;
      elsif (reset='0') then
        if (en='1') then
          if (up='1') then
            s:=s+n1;
          else
            s:=s-n1;
          end if;
        end if;
      end if;
    end if;
    q<=std_logic_vector(s);
  end process;
end architecture behav;
```

Esercizio

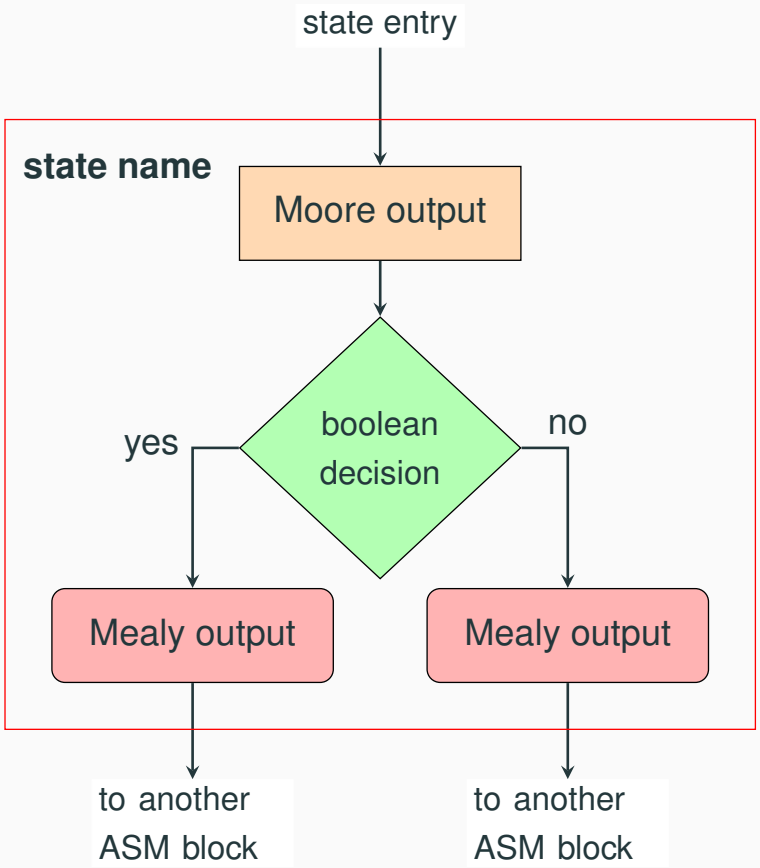
Si descriva tramite FSM e poi come EFSM un contatore binario sincrono avente come ingressi un segnale di `start` e una parola `a` che rappresenta un numero binario senza segno. Non appena ricevuto il segnale di `start`, il contatore conta da 0 a `a` e poi si arresta. L'uscita z si porta a 0 a inizio conteggio e assume il valore 1 a fine conteggio. Si consideri il caso (per la FSM) di $\texttt{a} \leq 15$.

ASM

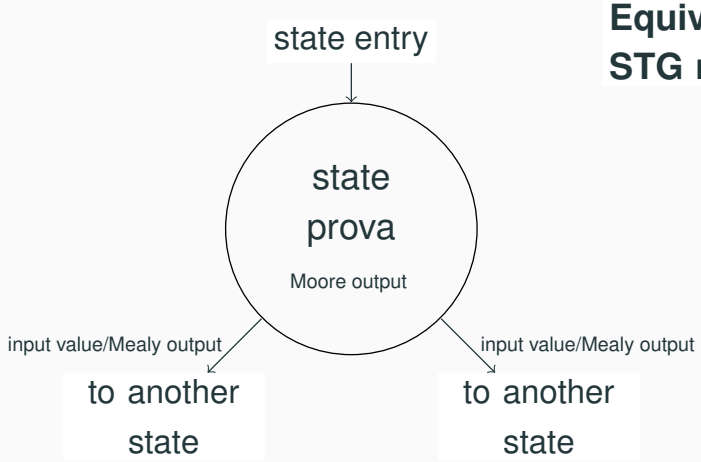
Algorithmic State Machines

- Si tratta di un formalismo alternativo a quello di FSM e EFSM
- Riprende il formalismo dei diagrammi di flusso usati nell'ambito del software
- É piú vicino all'implementazione del codice VHDL
- A ogni stato é associato un blocco che definisce dei valori per le uscite di Moore e quelle di Mealy, e un test che determina lo stato futuro
- Per evitare la proliferazione di stati, il blocco puó avere piú test e piú rami di uscita

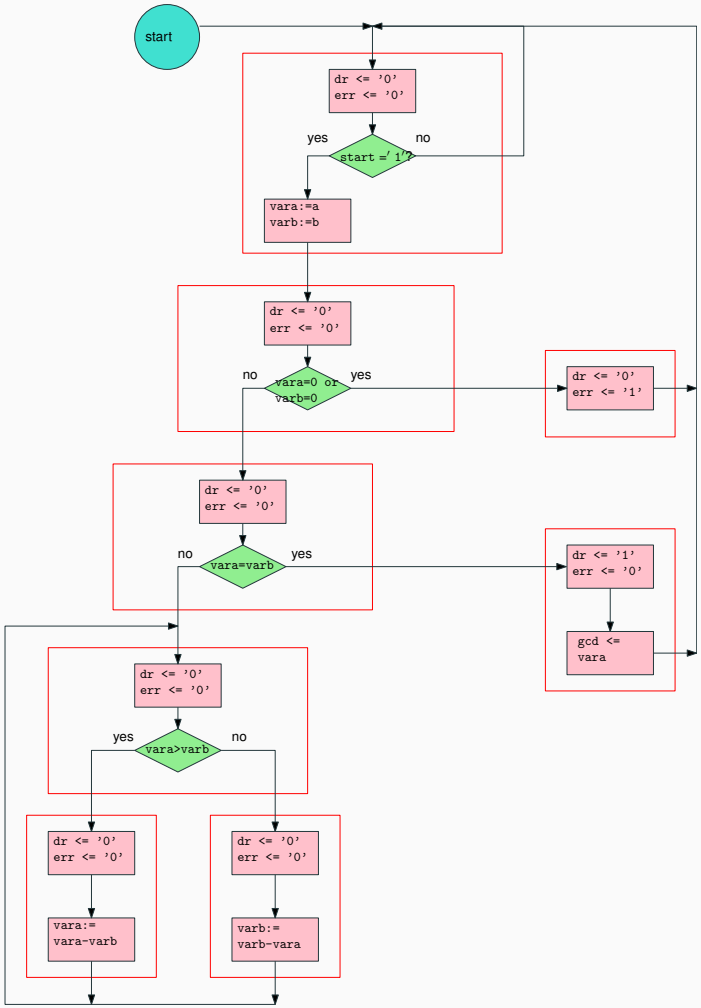
ASM - blocco (stato con 2 archi uscenti)



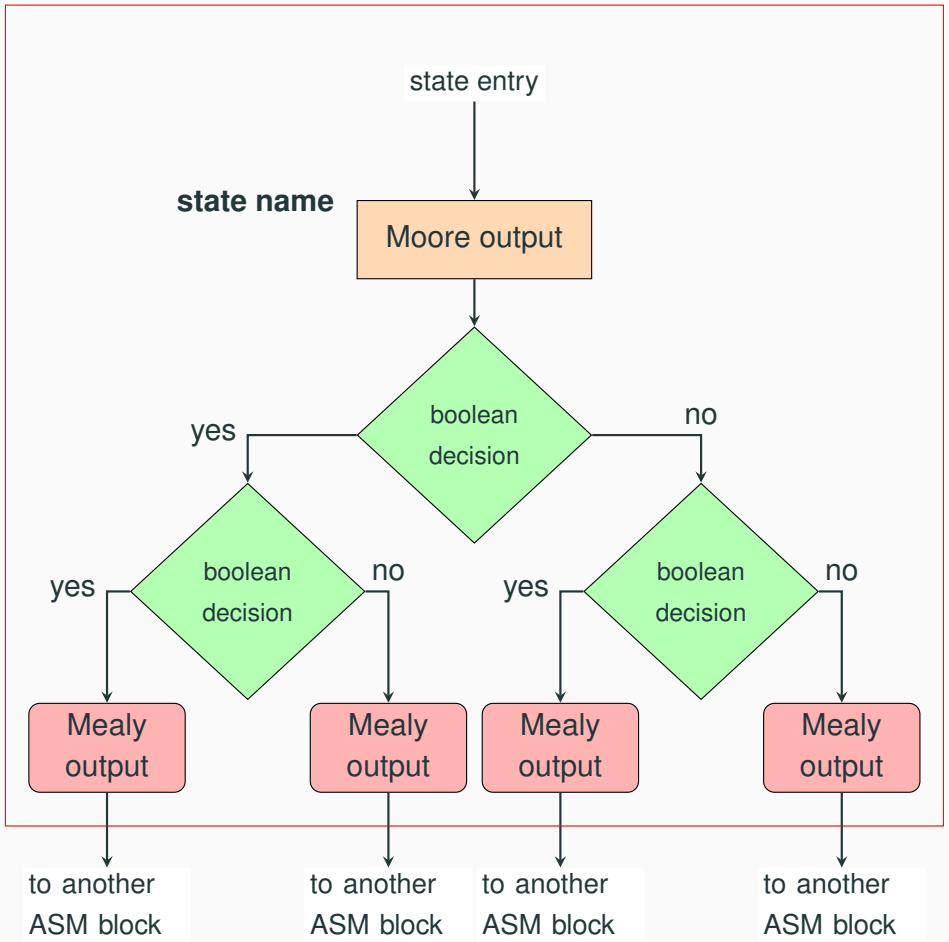
Equivalent STG notation

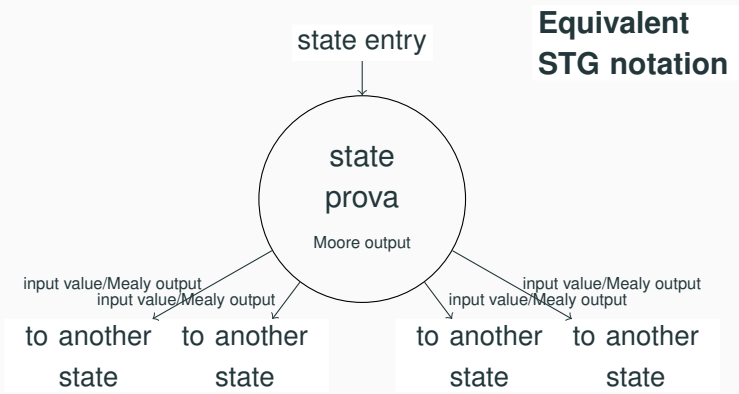


ASM - esempio gcd



ASM - blocco (stato con 4 archi uscenti)





Conclusioni

- Si é introdotta la descrizione in VHDL di una FSM simbolica
- Il formalismo delle FSM é stato esteso con le EFSM che consentono una rappresentazione compatta di controllo e data-path
- Si é illustrato un formalismo alternativo (ASM)