Descrizione di macchine a stati tramite VHDL

M. Favalli

Engineering Department in Ferrara

Motivazioni

- Introdurre la descrizione di FSM in VHDL
- Introdurre il modello di macchine a stati estese
  - largamente utilizzato a livello di design-entry
  - puó anche essere il prodotto della sintesi ad alto livello
  - esempio di modello VHDL di una macchina a stati estesa
- Formalismi alternativi

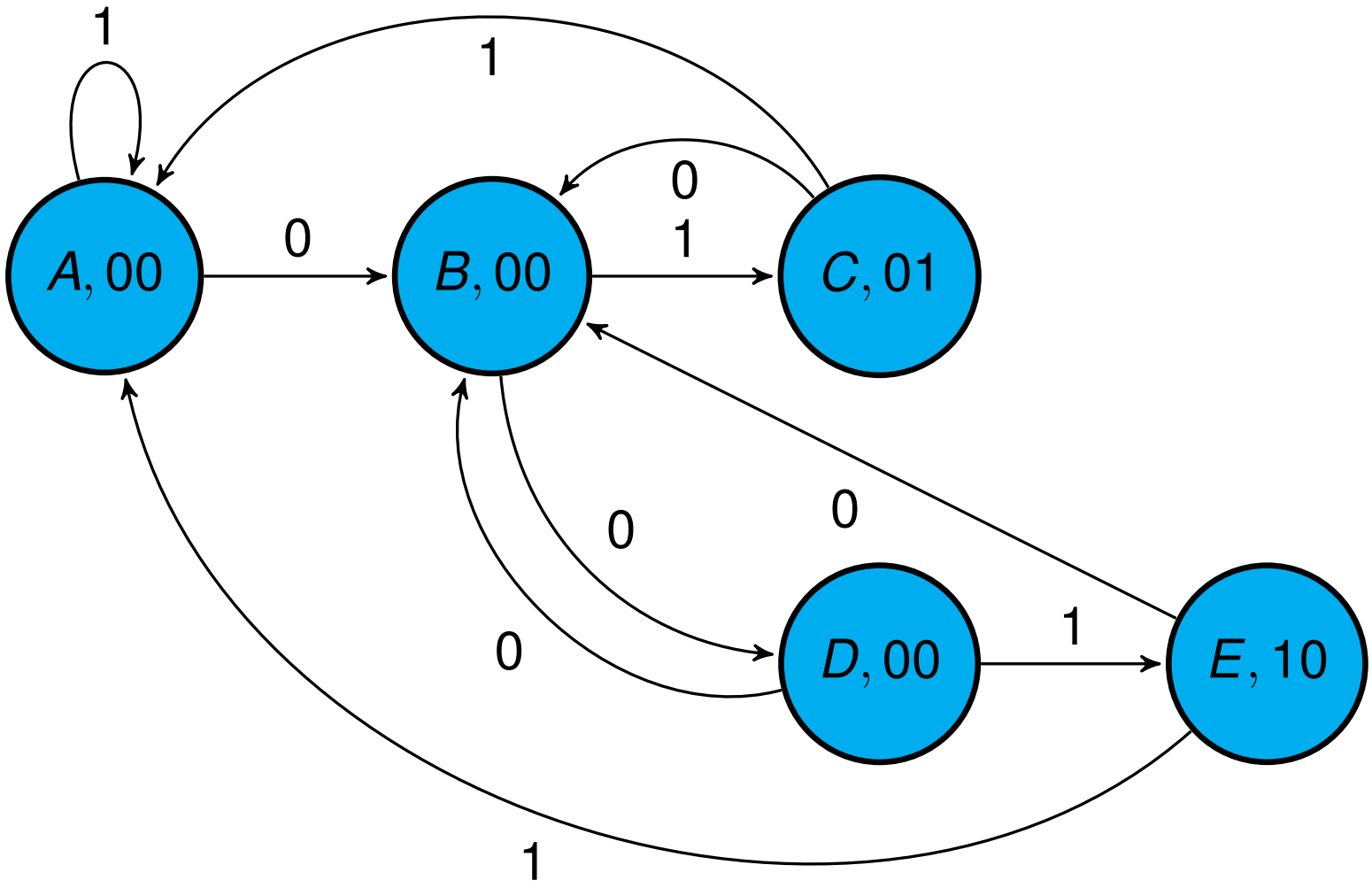
Riepilogo su FSM

- 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: STG e State Table
- Modello per l'implementazione: Huffman
- Modello per le transizioni di stato: sincrono

Esempio di FSM (Moore)

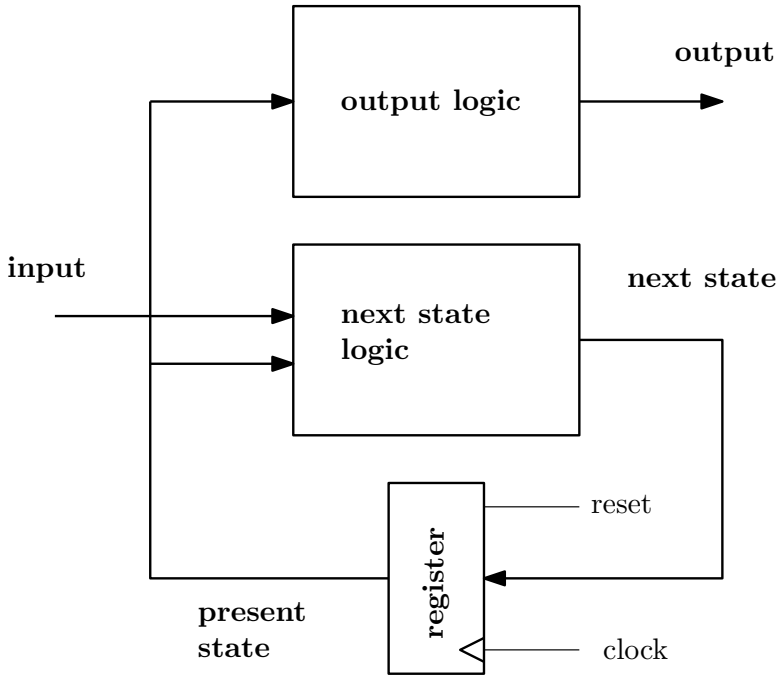
- Automa con un ingresso x e due uscite y, w
- Riconosce le sequenze di ingresso 01 e 001 (non sovrapponibili) producendo le uscite 01 e 10
- Quando non viene riconosciuto alcun simbolo, le uscite valgono 00

Esempio di FSM (Moore)



Descrizione VHDL

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



VHDL (I)

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm is
  port(x: 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;

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 (II)

```
-- next-state logic
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 (III)

```
-- 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

- La sintesi e ottimizzazione della FSM prevede i seguenti passi:
  1. minimizzazione del numero di stati
  2. codifica dello stato e verifica che la FSM sia safe
  3. sintesi e ottimizzazione della parte combinatoria della rete
  4. prima stima della frequenza di clock
- A questi passi segue poi la sintesi fisica e la valutazione accurata della frequenza di clock

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

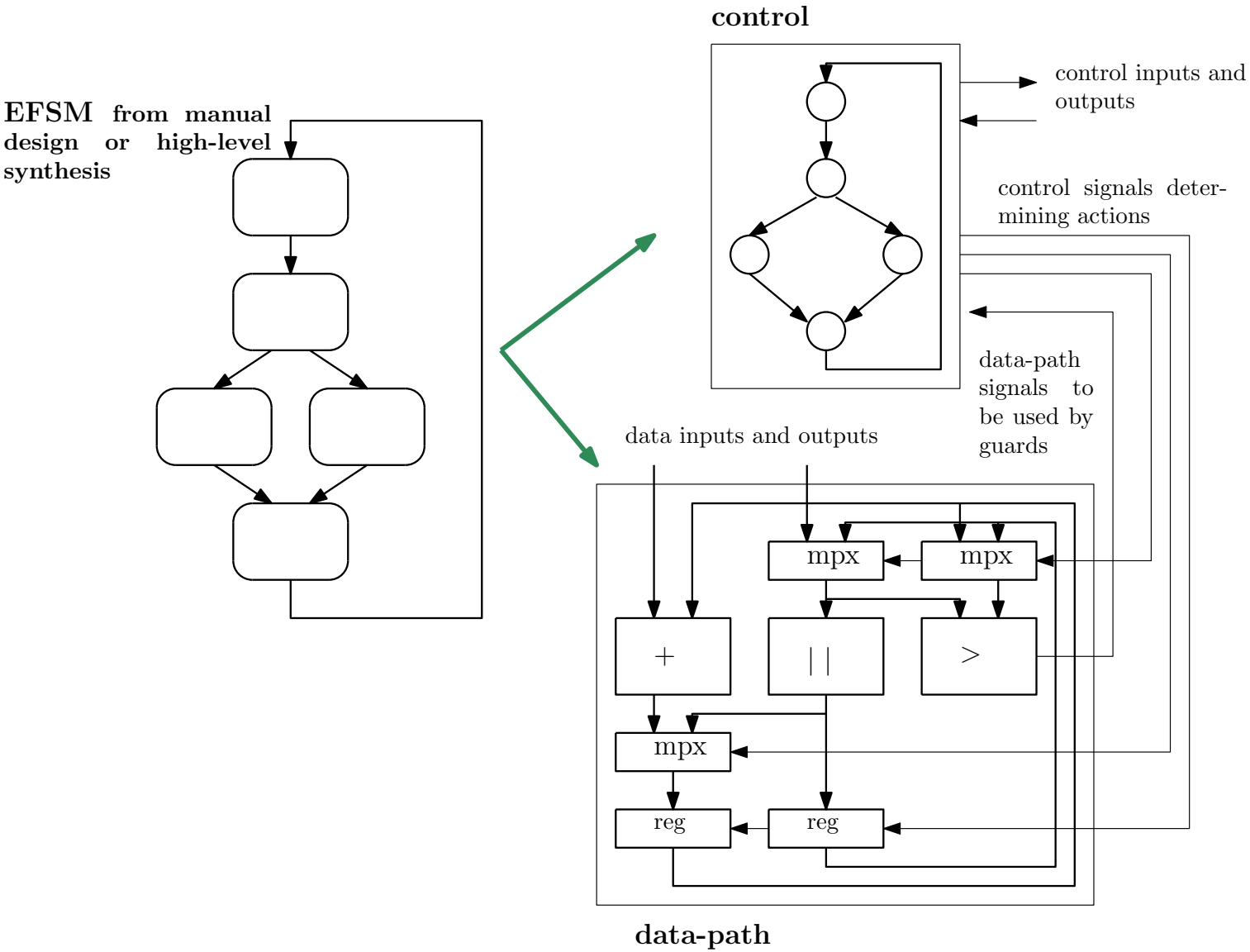
Extended Finite State Machines - EFSM

- 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 di sintesi da EFSM a controllo e data-path



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, description
-- 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

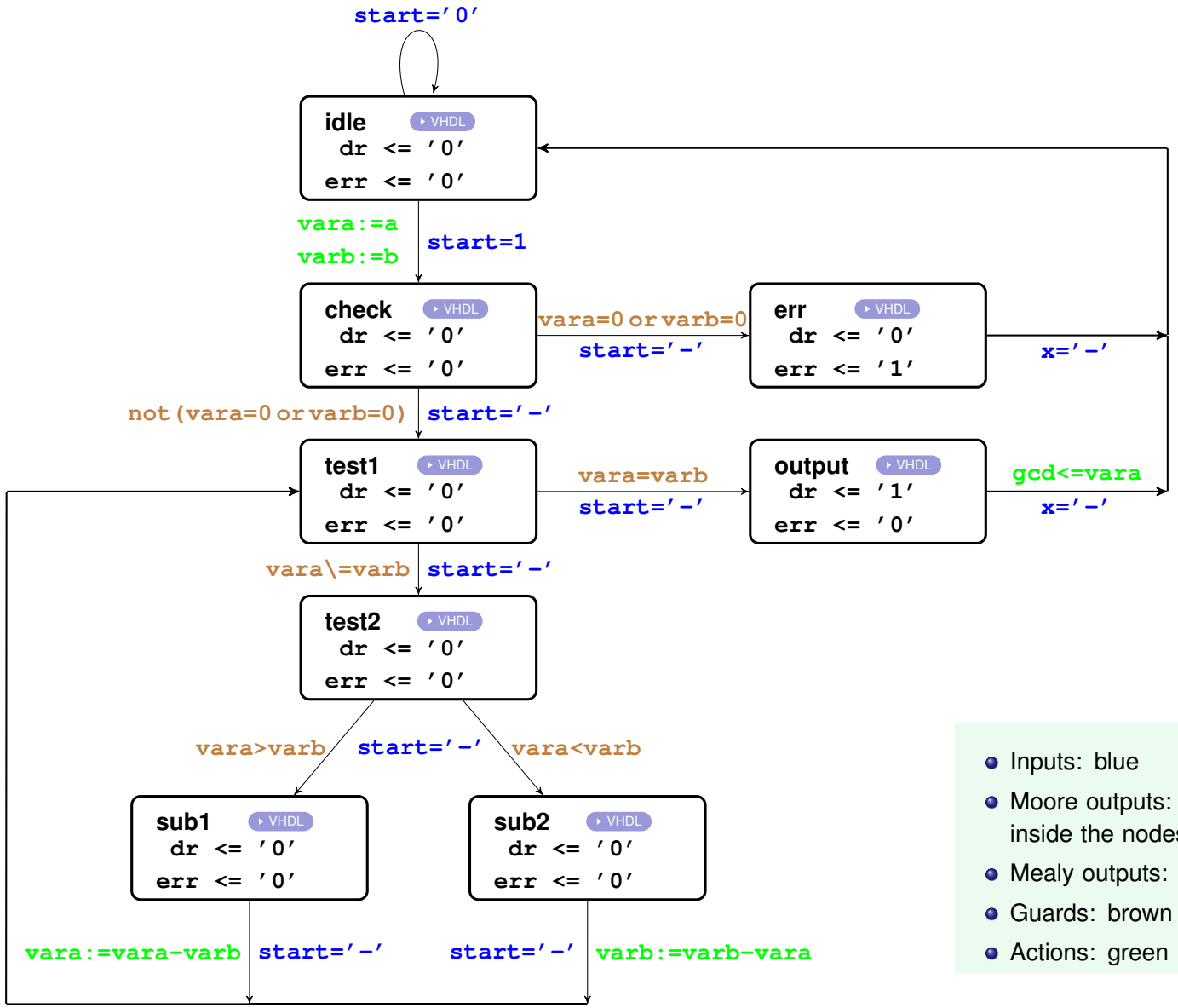
```
  process(a,b)
  variable vara,varb: unsigned(7 downto 0);
  constant zero: std_logic_vector(7 downto 0):=(others=>'0');
  begin
    if (a=zero) or (b=zero) or (is_x(a)) or (is_x(b)) then
      gcd <= (others=>'X');
    else
      vara:=unsigned(a);
      varb:=unsigned(b);
      while (vara/=varb)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

▸ VHDL - entity ▸ VHDL - architecture



- 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, any change after start is ignored
-- the error and the data redy 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

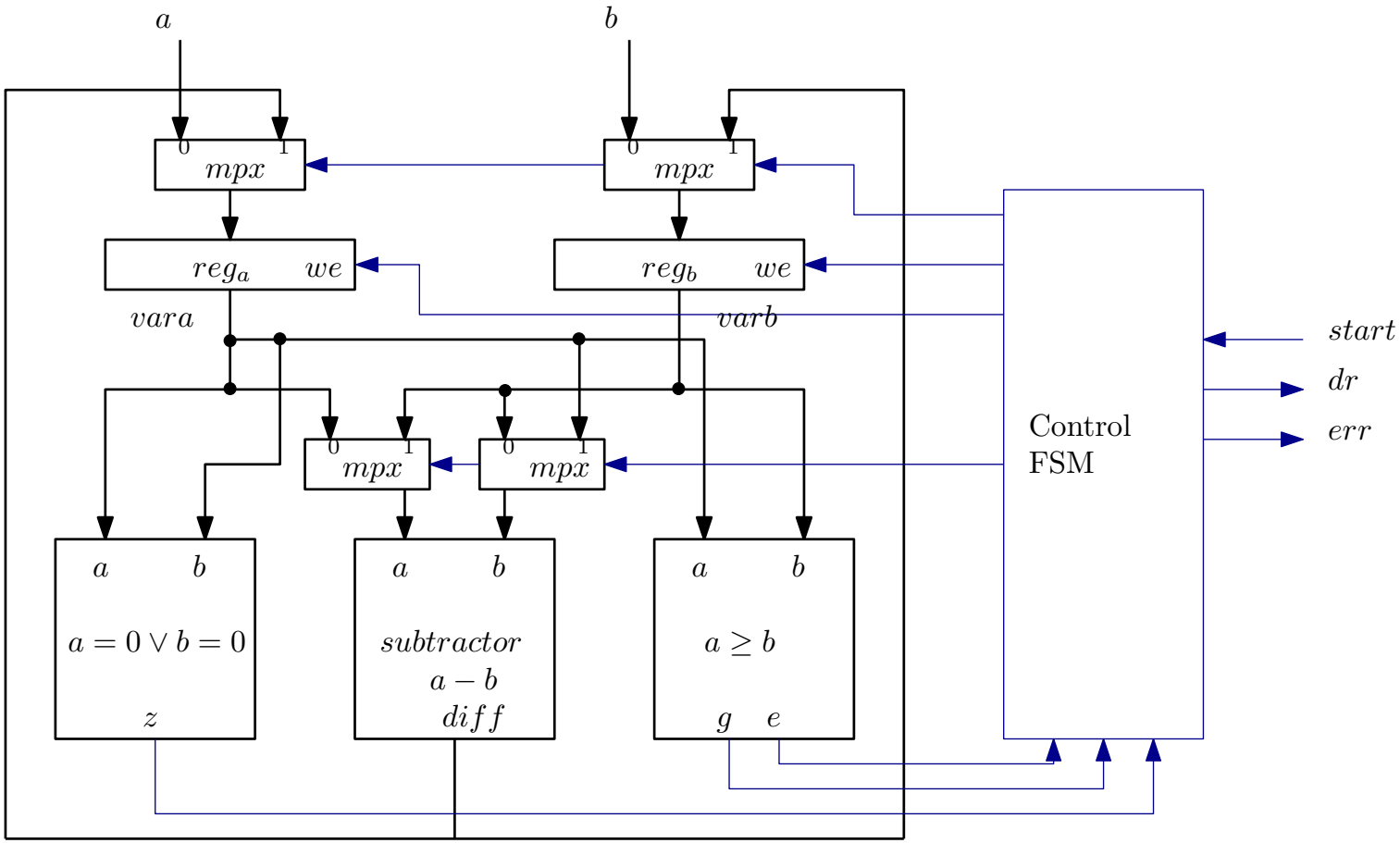

```
      when output =>
          next_state <= idle;
          dr <= '1' after 1 ns;
          err <= '0' after 1 ns;
          gcd <= std_logic_vector(vara);
      when err =>
          next_state <= idle;
          dr <= '0';
          err <= '1';
      -- useful when encoding
      when others =>
          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 (nel flusso della sintesi ad alto livello la EFSM viene speficicata dopo lo scheduling e prima del binding)
- Ci sono margini per l'ottimizzazione sia a partire dalla descrizione behavioral che dalla EFSM
- Esistono algoritmi di sintesi in grado di trasformare l'EFSM (RTL comportamentale) in un RTL strutturale estraendo data-path e controllo

gcd - livello RTL strutturale



Esercizio

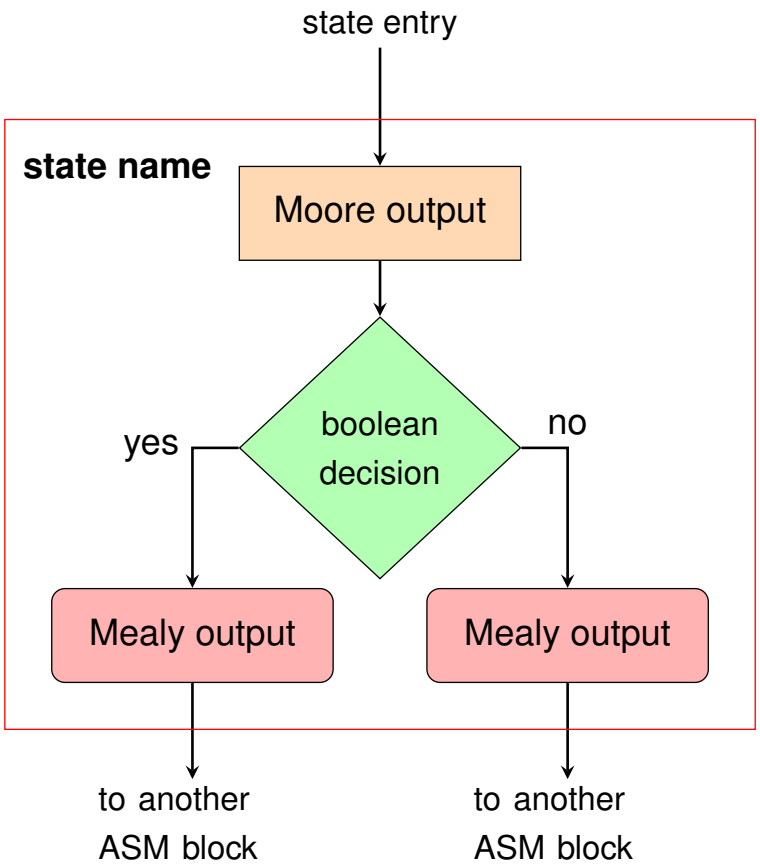
Si descriva tramite FSM e poi come EFSM un contatore binario sincrono avente come ingressi un segnale di `start` e una parola `a` che rappresenta un numero binario senza segno. Non appena ricevuto il segnale di `start`, il contatore conta da 0 a `a` e poi si arresta. L'uscita z si porta a 0 a inizio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

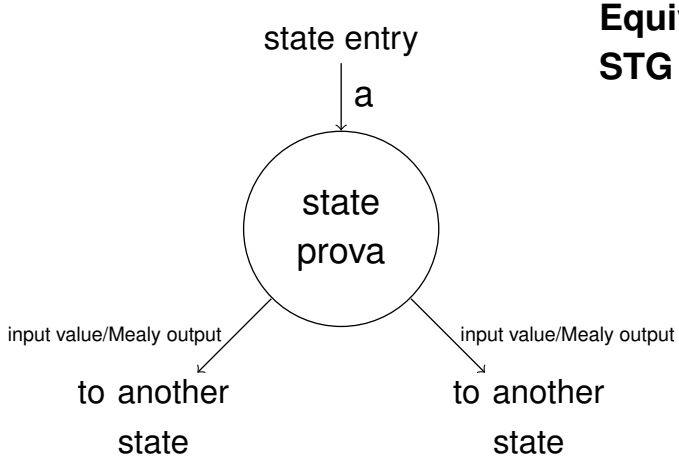
Sommario

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
- Utile dal punto di vista dell'implementazione del codice VHDL

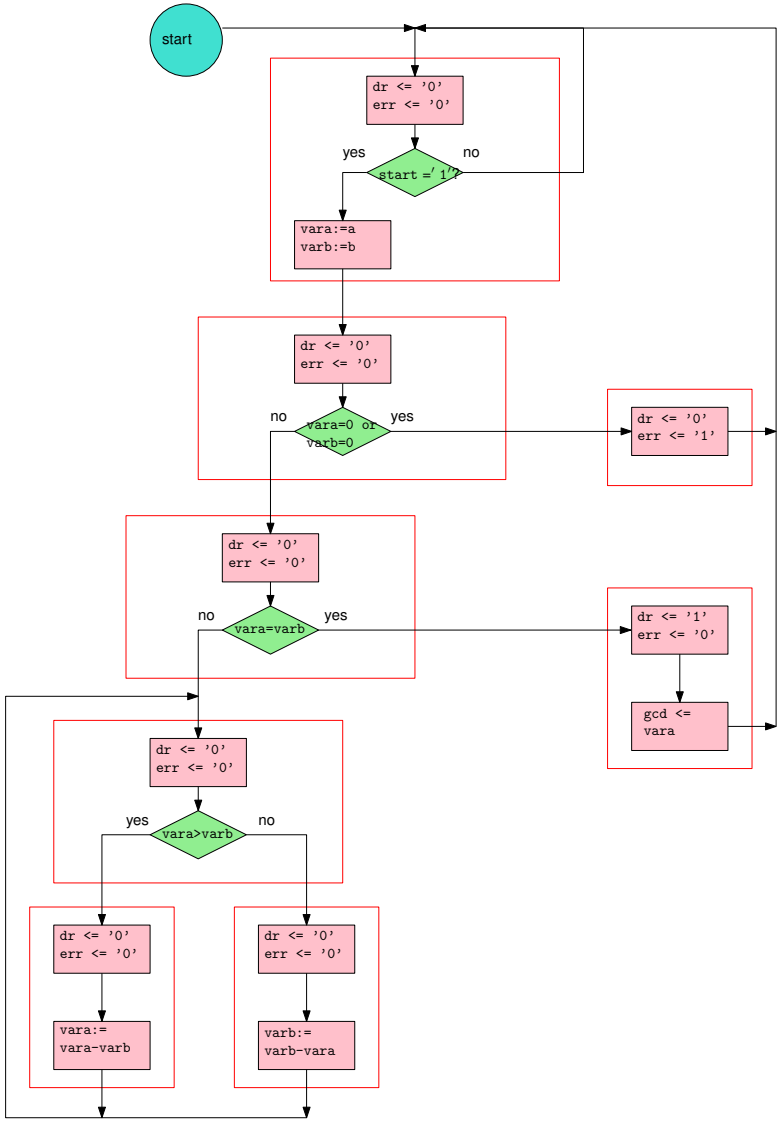
ASM - blocco (stato con 2 archi uscenti)



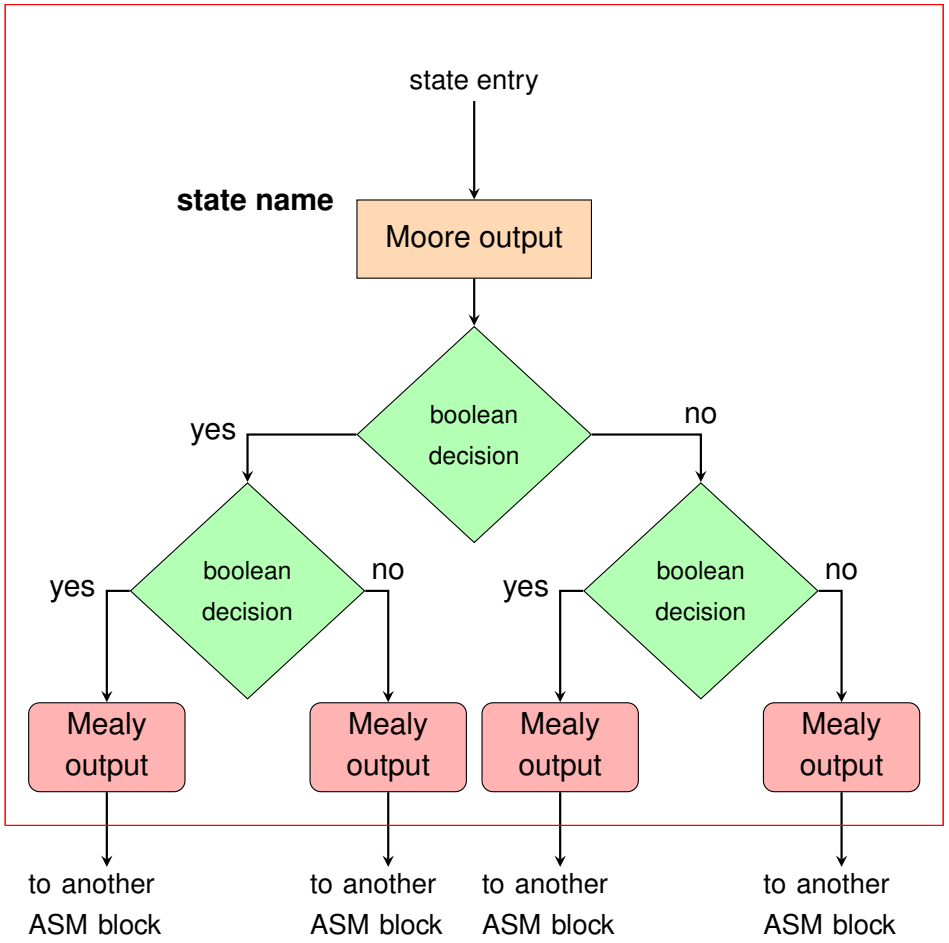


Equivalent STG notation

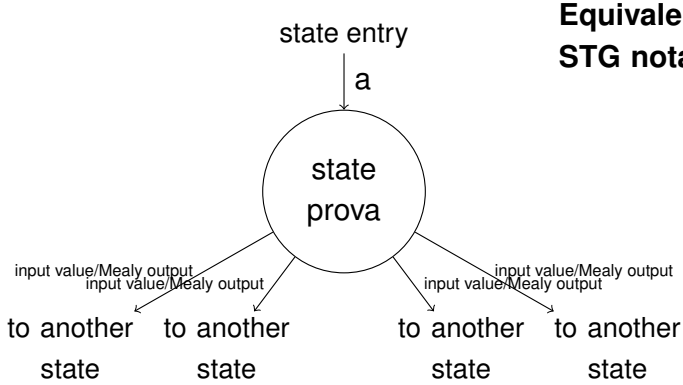
ASM - esempio gcd



ASM - blocco (stato con 4 archi uscenti)



Equivalent STG notation



Confronto fra HLS e utilizzo di EFSM

- La sintesi ad alto livello deve essere supportata da tool di EDA
- Il CDFG dopo lo scheduling corrisponde a una EFSM
- La EFSM (o ASM) assume implicitamente uno scheduling delle operazioni
- EFSM (o ASM) sono piú adatte a uno stile di progetto manuale

ASM - Esempio

ASM - VHDL code