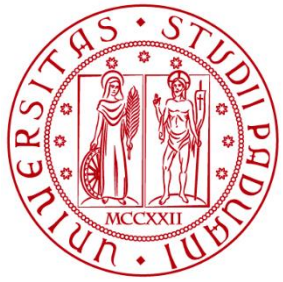
DEI
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sistemi Digitali

Circuiti Sequenziali in VHDL: Esempio del riconoscitore di sequenza

Marta Bagatin, marta.bagatin@unipd.it

Corso di Laurea in Ingegneria dell'Informazione

Anno accademico 2022-2023

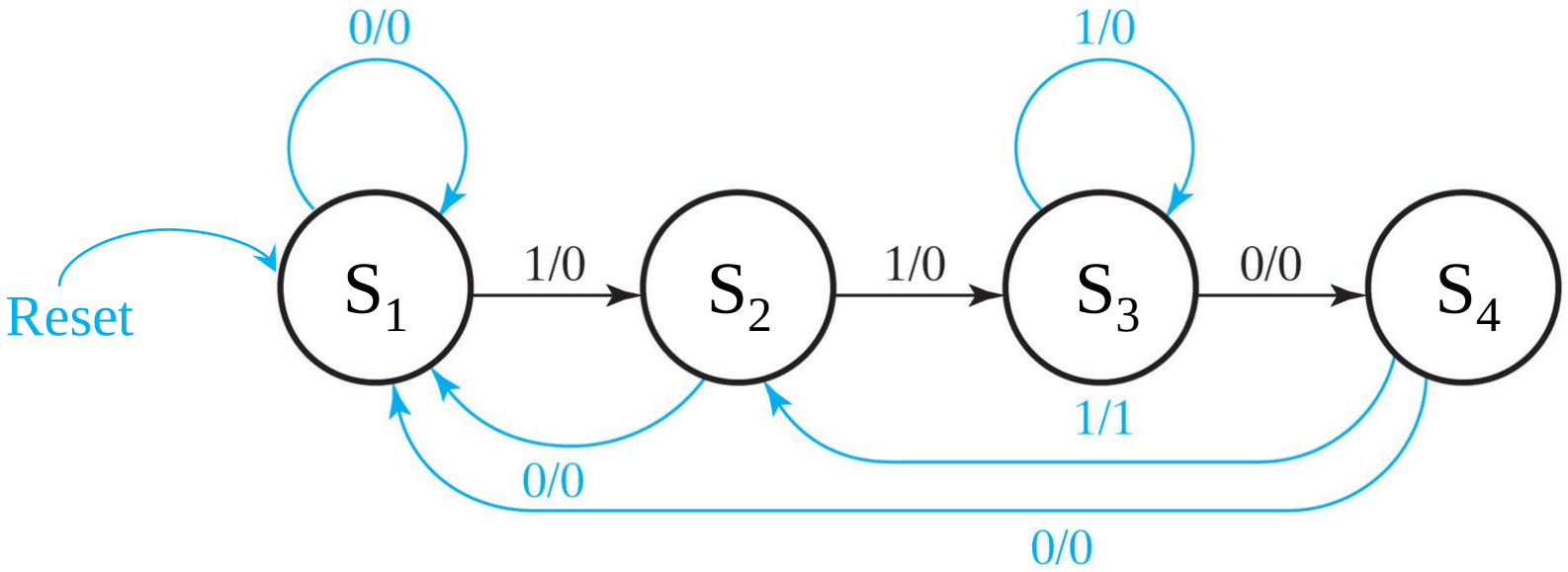
Scopo della lezione

- Descrivere in VHDL un **circuito sequenziale sincrono**
- Vedremo un possibile design e testbench del riconoscitore di sequenza (stesso esempio usato per la sintesi di logica sequenziale)

Riconoscitore di sequenza

- **Ingresso: X, Uscita: Z, riconosce la sequenza «1101»** (indipendentemente da dove si presenta all'interno di una sequenza più lunga)
  - Z = 1 quando si è presentata la sequenza 110 e X = 1
  - Z = 0 altrimenti
  - Il reset porta il circuito nello stato di nessun bit riconosciuto (stato S_1)

Diagramma degli stati:

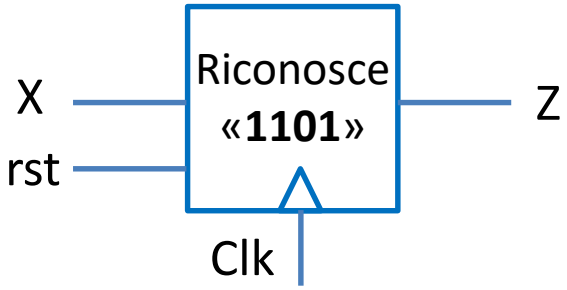


Descrizione VHDL di un circuito sequenziale

- Descriveremo il circuito con tre processi distinti
  - **Processo 1**: descrive l'**aggiornamento dello stato** del sistema
  - **Processo 2**: calcola lo **stato futuro** come funzione di stato e ingresso
  - **Processo 3** : calcola l'**uscita** come funzione di stato e ingresso
- Il processo 1 realizza la parte sequenziale del circuito. Viceversa, i processi 2 e 3 descrivono logica puramente combinatoria
- Vedremo una descrizione di tipo comportamentale del circuito

Riconoscitore di sequenza: VHDL (1/4)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity seq_rec is
   port ( clk, rst, X: in std_logic;
                    Z: out std_logic);
end seq_rec;

architecture beh of seq_rec is
   type state_type is (S1, S2, S3, S4);
   signal state, next_state: state_type;
begin
--process 1: aggiorna lo stato e implementa il reset asincrono
   state_register: process (rst, clk)
       begin
          if (rst = '1') then
             state <= S1;
          elsif (clk'event and clk = '1') then
             state <= next_state;
          end if;
   end process;
```



Type: definizione di un nuovo tipo di dato

Realizziamo la memoria con D-FF con reset asincrono.
Al reset il sistema viene portato allo stato S1

Type

Sintassi:

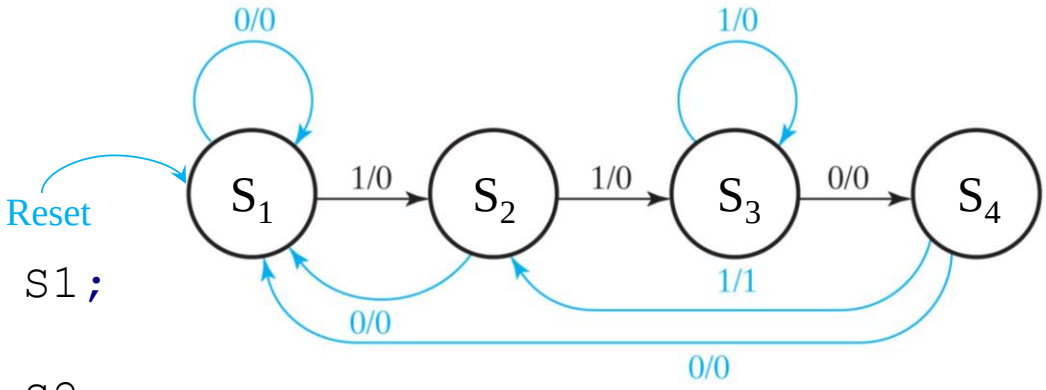
```
type nome_tipo is ( valore1, valore2, valore3, …);
```

- Con la parola chiave `type` possiamo definire nuovi tipi di dati
- Per esempio possiamo definire un nuovo tipo elencandone tutti i valori tra parentesi tonde e separati da virgole (enumeration). E' utile per definire segnali che possono assumere un numero finito di valori (identificati da nomi, non da numeri, per migliorare la leggibilità del codice)
- Nota: il tipo std_logic che usiamo per i segnali logici è un tipo enumeration, che può assumere 9 valori distinti: '0', '1', 'X', '-', 'U', 'Z', 'W', 'L', 'H'
- Una volta dichiarato un tipo all'interno di un design, si possono usare segnali di quel tipo con la usuale dichiarazione:

```
signal nome_segnale: nome_tipo;
```

Riconoscitore di sequenza: VHDL (2/4)

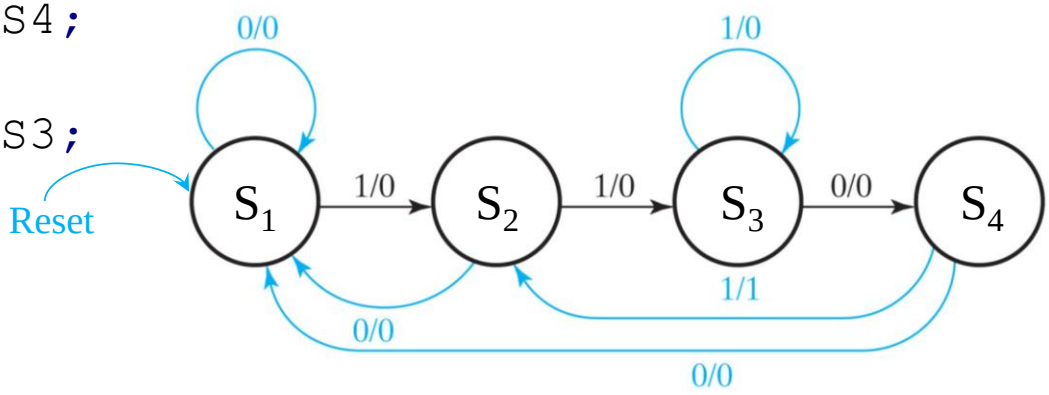
```
--process 2: calcola lo stato futuro come funzione
combinatoria di stato presente e ingresso

 next_state_comb: process (X, state)
 begin

 case state is
       when S1 =>
          if X = '0' then
              next_state <= S1;
          else
              next_state <= S2;
          end if;

       when S2 =>
          if X = '0' then
              next_state <= S1;
          else
              next_state <= S3;
          end if;
```



Stato futuro definito con un **case** (analogo sequenziale di with-select), in base ai valori dello stato e dell'ingresso

Riconoscitore di sequenza: VHDL (3/4)

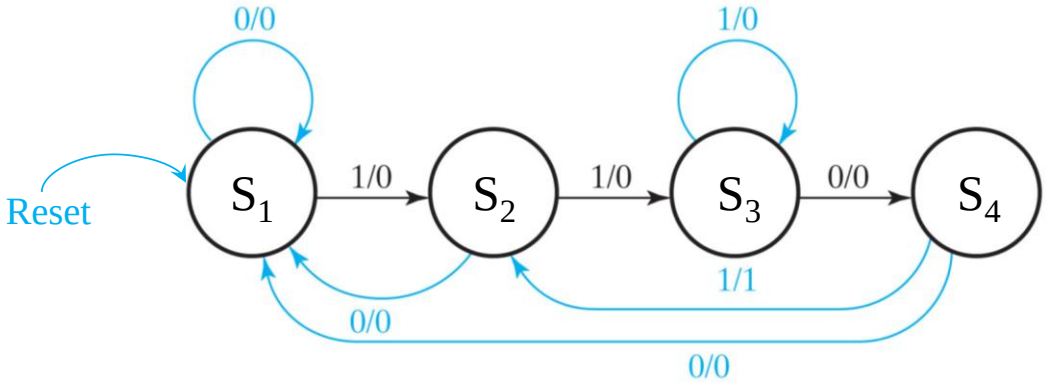
```
--process 2 (continued)

      when S3 =>
          if X = '0' then
              next_state <= S4;
          else
              next_state <= S3;
          end if;

      when S4 =>
          if X = '0' then
              next_state <= S1;
          else
              next_state <= S2;
          end if;
 end case;

 end process;
```



Non serve il when others, dato che abbiamo già enumerato tutti i casi possibili per il valore che può assumere state

Riconoscitore di sequenza: VHDL (4/4)

```
--process 3: calcola l'uscita come funzione combinatoria di
stato presente e ingresso

 output_comb: process (X, state)
 begin

 case state is
       when S1 =>
          Z <= '0';
       when S2 =>
          Z <= '0';
       when S3 =>
          Z <= '0';
       when S4 =>
          if X = '0' then
               Z <= '0';
          else
               Z <= '1';
          end if;
 end case;
 end process;
 end beh;
```



Uscita definita con un altro **case**, in base ai valori dello stato e dell'ingresso (macchina di Mealy)

Osservazioni

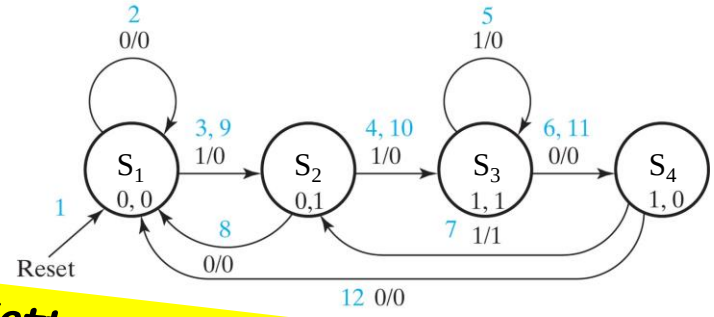
- I **tre processi** vengono **eseguiti in parallelo** e **interagiscono tra loro**, avendo segnali in comune
- In questo design abbiamo usato una **descrizione behavioral**: non abbiamo in alcun modo suggerito come realizzare il sistema (come codificare gli stati, come implementare la logica combinatoria che descrive lo stato futuro e l'uscita). Abbiamo solo fornito una descrizione del comportamento della macchina
- Con strutture semplici, i tool di sintesi automatica comunemente usati sono in grado di determinare l'implementazione ottima del sistema
- Se avessimo voluto descrivere una macchina di Moore
  - La sensitivity list del processo usato per descrivere l'uscita avrebbe contenuto solo lo stato presente e non l'ingresso
  - il case usato per l'uscita non avrebbe contenuto l'ingresso, ma solo lo stato

Simulazione di logica sequenziale sincrona

- E' necessario **cambiare gli input non troppo a ridosso del fronte di clock attivo** (per es. durante il fronte di discesa, nell'ipotesi di positive-edge-triggered FF). In questo modo, la parte di logica combinatoria ha tempo di calcolare uscita e stato futuro fino al successivo fronte attivo del clock
- Realizzeremo il testbench con **due processi**, che funzionano in parallelo:
  1) il primo **genera il clock**. Questo processo non viene mai fermato esplicitamente e riprende da capo ogni volta che ha esaurito le istruzioni contenute al suo interno, garantendo che il clock continui ad oscillare
  2) il secondo **pilota i valori dei segnali in ingresso**, incluso il reset, del sistema (sincronizzando le transizioni opportunamente, per es. ai fronti di discesa del clock). Questo processo viene fermato con il comando "`std.env.stop`", che termina la simulazione

Riconoscitore di sequenza: testbench (1/2)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test_seq_rec is
end test_seq_rec;

architecture test of test_seq_rec is

signal clock, reset, X, Z: std_logic;
signal test_sequence: std_logic_vector(0 to 10):="01110101100";

constant PERIOD: time := 100 ns;

component seq_rec is
   port (clk, rst, X: in std_logic;
                   Z: out std_logic);
end component;

begin
DUT: seq_rec port map(clock, reset, X, Z);
```



Vettore di test scelto in modo da passare in ordine attraverso tutti gli stati del diagramma

Definizione di una **costante** di tipo time

Istanziazione del circuito da simulare

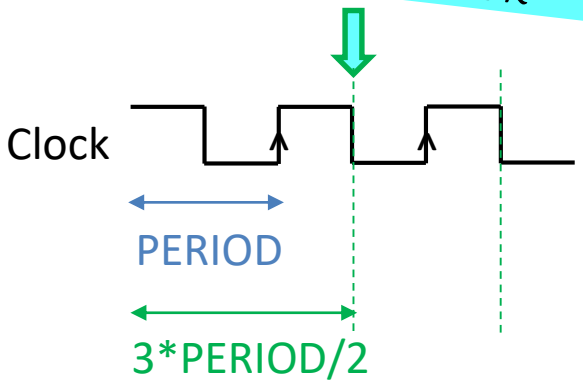
Riconoscitore di sequenza: testbench (2/2)

```
generate_clock: process begin
    clock <= '1';
    wait for PERIOD/2;
    clock <= '0';
    wait for PERIOD/2;
    --report "X = " & to_string (X) & " Z = " & to_string (Z);
end process;

apply_inputs: process begin
    reset <= '1';
    X <= '0';
    wait for 3*PERIOD/2;
    reset <= '0';
    for i in 0 to 10 loop
        X <= test_sequence(i);
        wait for PERIOD;
    end loop;
    std.env.stop;
end process;
end test;
```

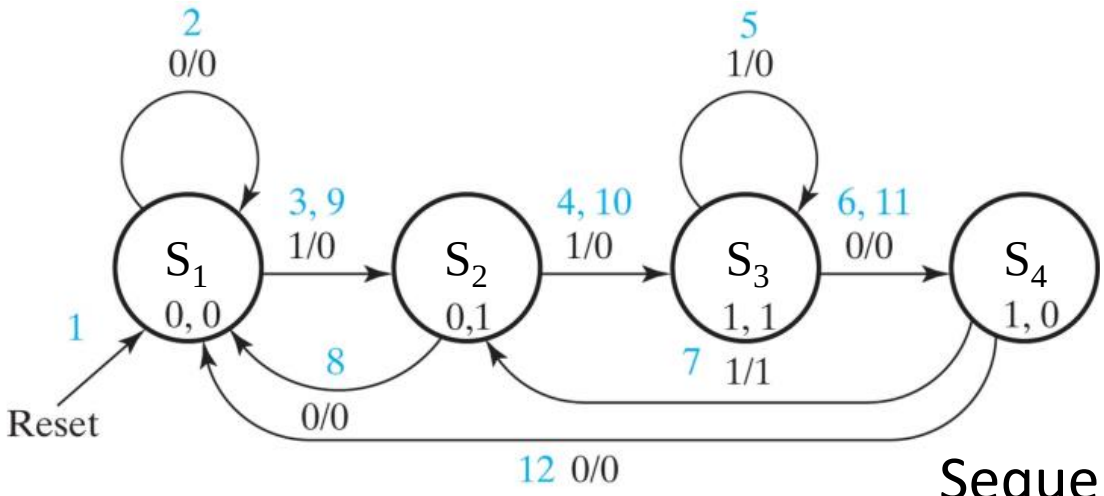
Processo che genera un **clock** con periodo pari a PERIOD

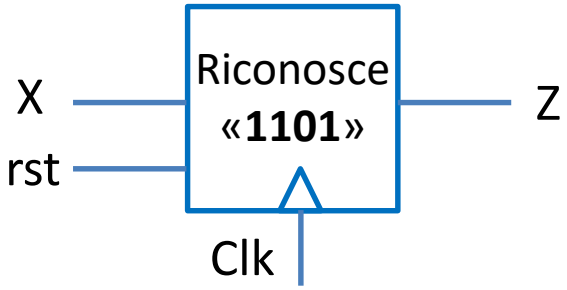
Processo che applica il reset e poi la sequenza di ingressi contenuti in test_sequence

Gli ingressi sono applicati ai fronti di discesa del clock

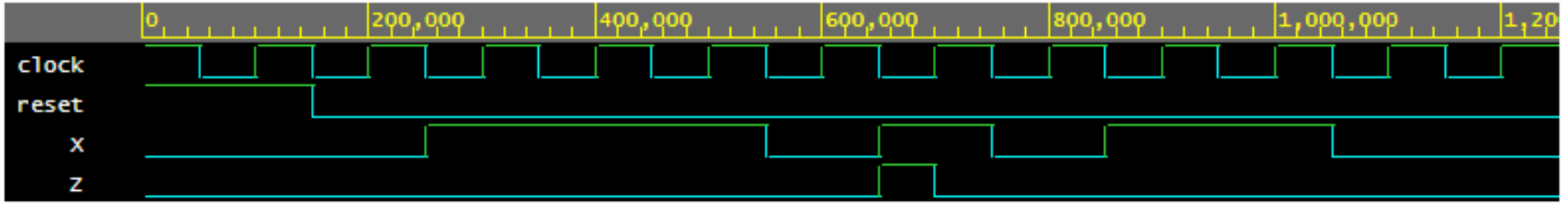


Riconoscitore di sequenza: forma d'onda





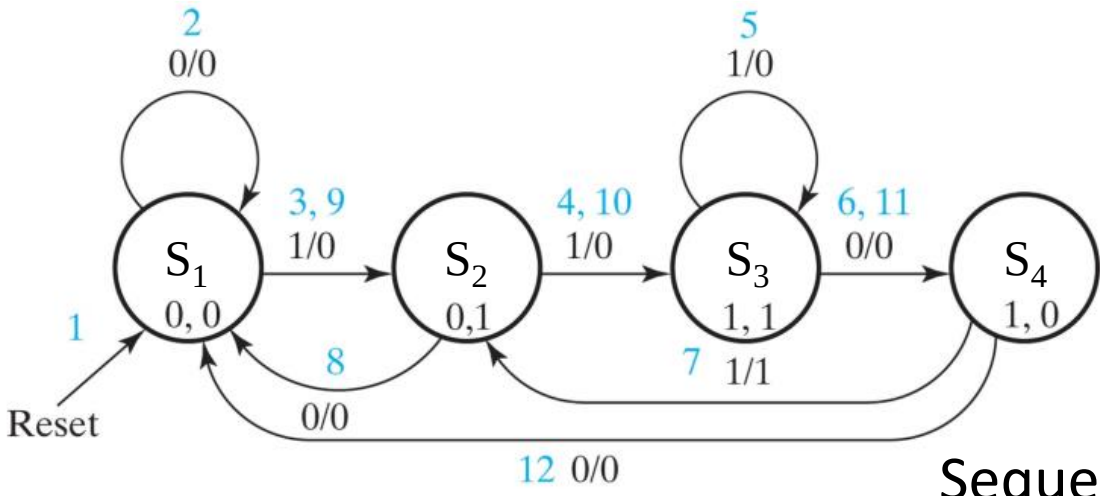
Sequenza di test: 01110101100

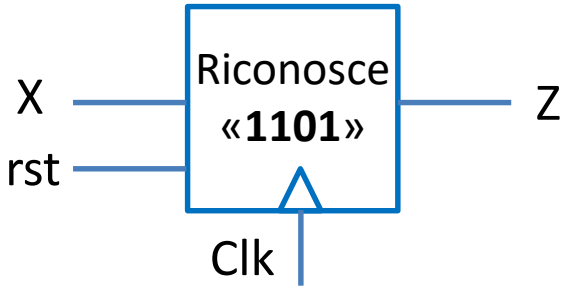


Reset iniziale: il sistema parte da uno stato noto

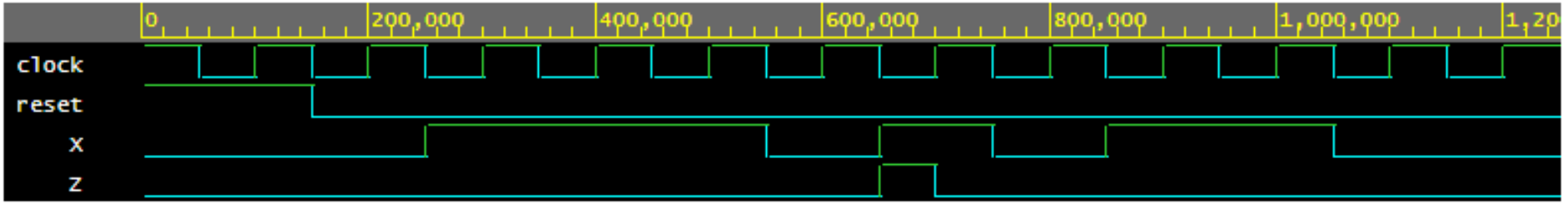
L'uscita diventa 1 all'arrivo della sequenza corretta (macchina di Mealy: uscita reagisce immediatamente al cambio l'ingresso)

Riconoscitore di sequenza: forma d'onda





Sequenza di test: 01110101100



NB: In una **macchina di Mealy**, l'uscita può cambiare anche non in corrispondenza al fronte attivo del clock (in seguito ad un cambiamento dell'ingresso). Ciò non può accadere in una **macchina di Moore**, dove l'uscita dipende solo dallo stato, che è memorizzato in un FF, la cui uscita può cambiare solo al fronte attivo del clock

Generazione di memoria in VHDL

- Se si vuole realizzare un circuito combinatorio e ci si dimentica di specificare una (o più) condizioni all'interno di uno statement if-then-else, si realizza un elemento di memoria non voluto!
- Ciò è intenzionale quando si intende realizzare un elemento di memoria (latch o flip-flop)
  - Esempio (PET D-FF):

```
...
process (CLK)
  begin
    if (CLK'event and CLK = '1') then
        if (RESET = '1') then
            Q <= '0';
        else
            Q <= D;
        end if;
    end if;
end process;
```

Volutamente è specificata **solo** la condizione del fronte attivo del clock. In tutte le altre condizioni, non è specificata alcuna azione. Ciò implicitamente significa Q <= Q (memoria)

Generazione di memoria in VHDL

Inputs			Action
RESET = 1	CLK = 1	CLK' event	
FALSE	FALSE	FALSE	Unspecified
FALSE	FALSE	TRUE	Unspecified
FALSE	TRUE	FALSE	Unspecified
FALSE	TRUE	TRUE	Q <= D
TRUE	—	—	Q <= '0'

Memoria
Q <= Q

Elementi di Memoria non voluti

- Consideriamo un circuito combinatorio che identifica i numeri primi a 3 bit. L'output vale '1' se l'input è un numero primo. Supponiamo di usare un process con un «if» statement per descrivere il circuito
  - Circuito con 1 input (3 bit) e 1 output (1 bit)

```
architecture implIf of Prime is
begin process (input)
begin
   if (input = 3d"1" OR input = 3d"2" OR input = 3d"3" OR input = 3d"5" OR input = 3d"7")
then isPrime <= '1';
   else isPrime <= '0';
   end if;
end process;
end implIf;
```

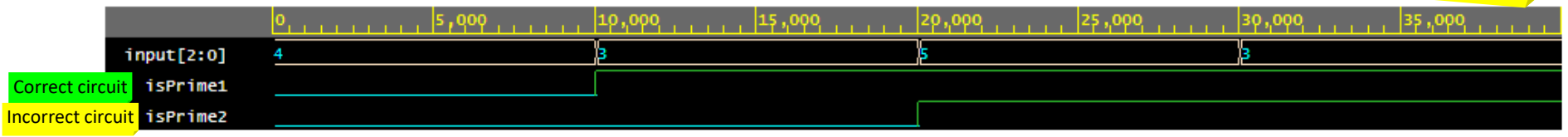
Corretto: il comportamento è specificato in tutti i casi (con la riga «else» ...)!

Elementi di Memoria non voluti

- Se ci dimentichiamo una condizione «else», creiamo un elemento di memoria in modo non intenzionale
  - Esempio:

```
architecture implIf of Prime is
begin process (input)
begin
    if (input = 3d"1" OR input = 3d"2" OR input = 3d"5" OR input = 3d"7")
then isPrime <= '1';
   elsif (input = 3d"4" OR input = 3d"6") then isPrime <= '0'
   end if;
end process;
end implIf;
```

Se ci dimentichiamo di specificare un caso (3d"3"), creiamo un latch: quando fa la transizione all'input 3, il sistema tiene l'output precedente: isPrime <= isPrime



Se '3' (output non specificato) è preceduto da '4' (isPrime <= '0'): output tiene il valore '0'
Se '3' (output non specificato) è preceduto da '5' (isPrime <= '1'): output tiene il valore '1'

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano | Kime | Martin