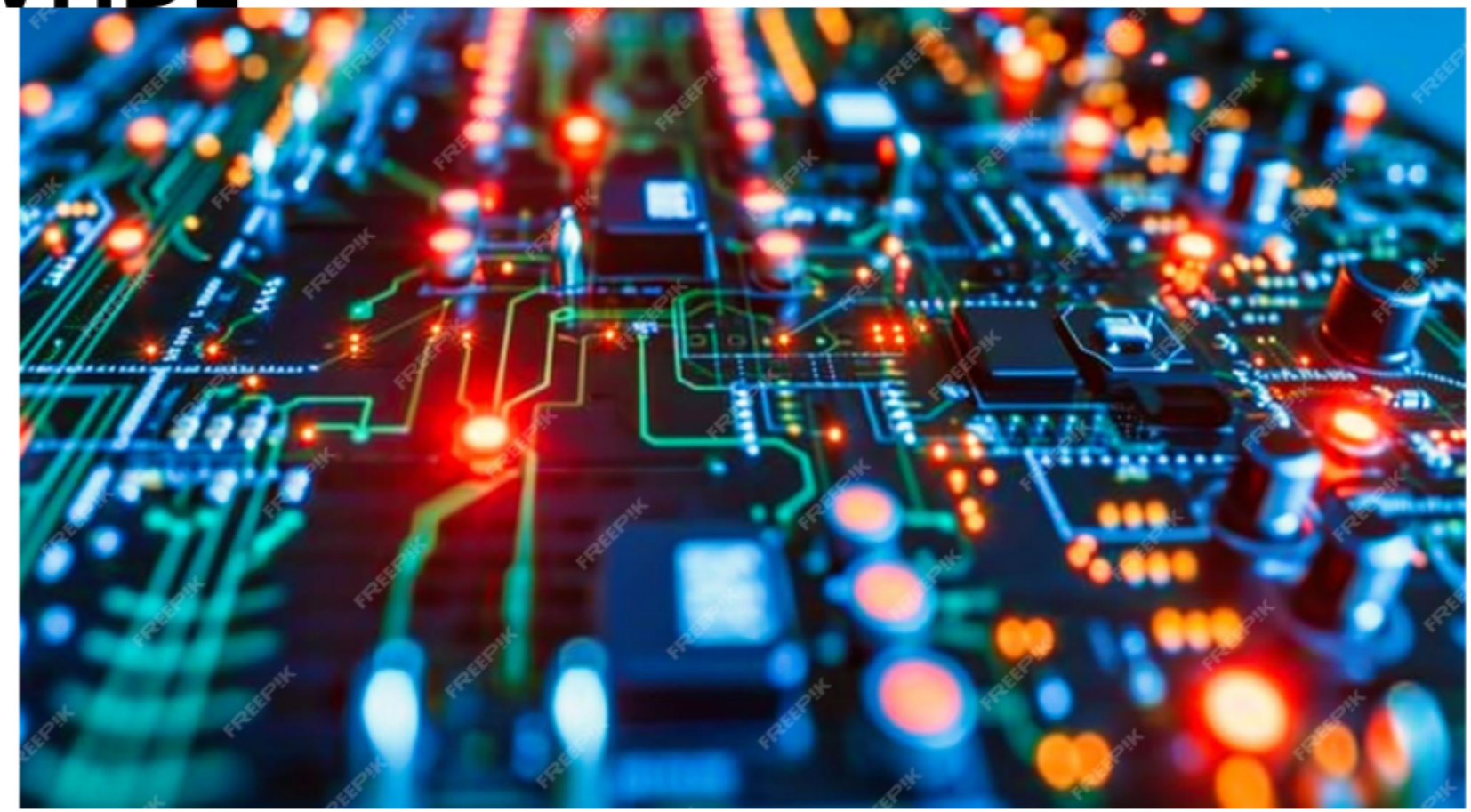
LAE1 VHDL

LAB8

Contatori



https://homes.di.unimi.it/re/Corsi/lae1_25_26/

a.a. 2025-26

matteo.re@unimi.it

Obiettivi della lezione

- Relazione tra FSM e contatori (ripasso teoria e trattamento “speciale” dei contatori in VHDL)
- Contatore modulo 2^N
- Contatore modulo 2^N up/down
- Descrizione VHDL di contatori : tecnica di descrizione mono processo (come evitare problemi di conversione tra tipi e descrivere contatori con un numero arbitrario di bit).
- Descrizione e simulazione dei tipi di contatori presentati

Esercizi

- Descrizione e simulazione di contatori espressi su un numero arbitrario di bit
- Contatore con load di tipo asincrono

Per ottenere i sorgenti di esercitazione:

wget http://homes.di.unimi.it/re/Corsi/lae1_25_26/L8.tar.gz
tar -zvxf L8.tar.gz
cd L8
(PDF slide, L8.pdf, disponibile in : http://homes.di.unimi.it/re/Corsi/lae1_25_26/)

Contatori

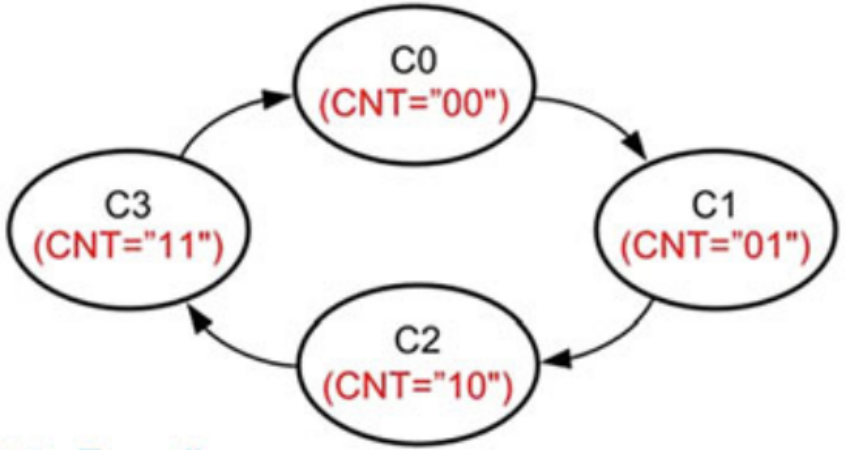
- I contatori sono una **sottoclasse** delle FSM
- La loro caratteristica principale e' quella di percorrere un diagramma delle transizioni tra gli stati con conformazione **ciclica** ed in modo **LINEARE**, nel senso che per raggiungere una seconda volta il medesimo stato sono costretti a riattraversare l'intero "anello" degli stati.
- Dato il vincolo di continuita' nell'attraversamento di STD, con una FSM di tipo contatore e' possibile adottare una **codifica degli stati** tale per cui sulla linea di output vengono semplicemente riportati **i bit della codifica degli stati**. In questo caso e' prassi comune definire la FSM del contatore come "state encoded".
- Nonostante il vincolo forte di percorrere STD in un'unica direzione e' possibile introdurre **segnali di controllo** che permettano al contatore di contare in avanti o all'indietro (questo **NON** cambia l'insieme dei numeri rappresentabili sulla linea in output … quello dipende dal numero degli stati). In questi casi ci si riferisce al contatore come contatore di tipo "**UpDown**".
- Dati i vincoli logici di questo tipo di contatore VHDL permette di descriverli **utilizzando UN UNICO PROCESSO**.

Esercizi

- Implementazione di vari tipi di contatori

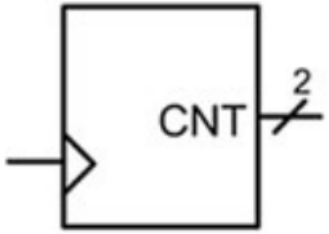
Contatore modulo 2^N con N=2

- 2 bit up counter binario. Il contatore **incrementa** di 1 ad ogni singolo fronte di **salita** del clock (00, 01, 10, 11). Quando il contatore raggiunge 11 riparte a contare da 00. La linea di output, composta da 2 bit, ha nome CNT.
- **STD** e **STT** :



(Output)

Current State	Next State	CNT
C0	C1	"00"
C1	C2	"01"
C2	C3	"10"
C3	C0	"11"



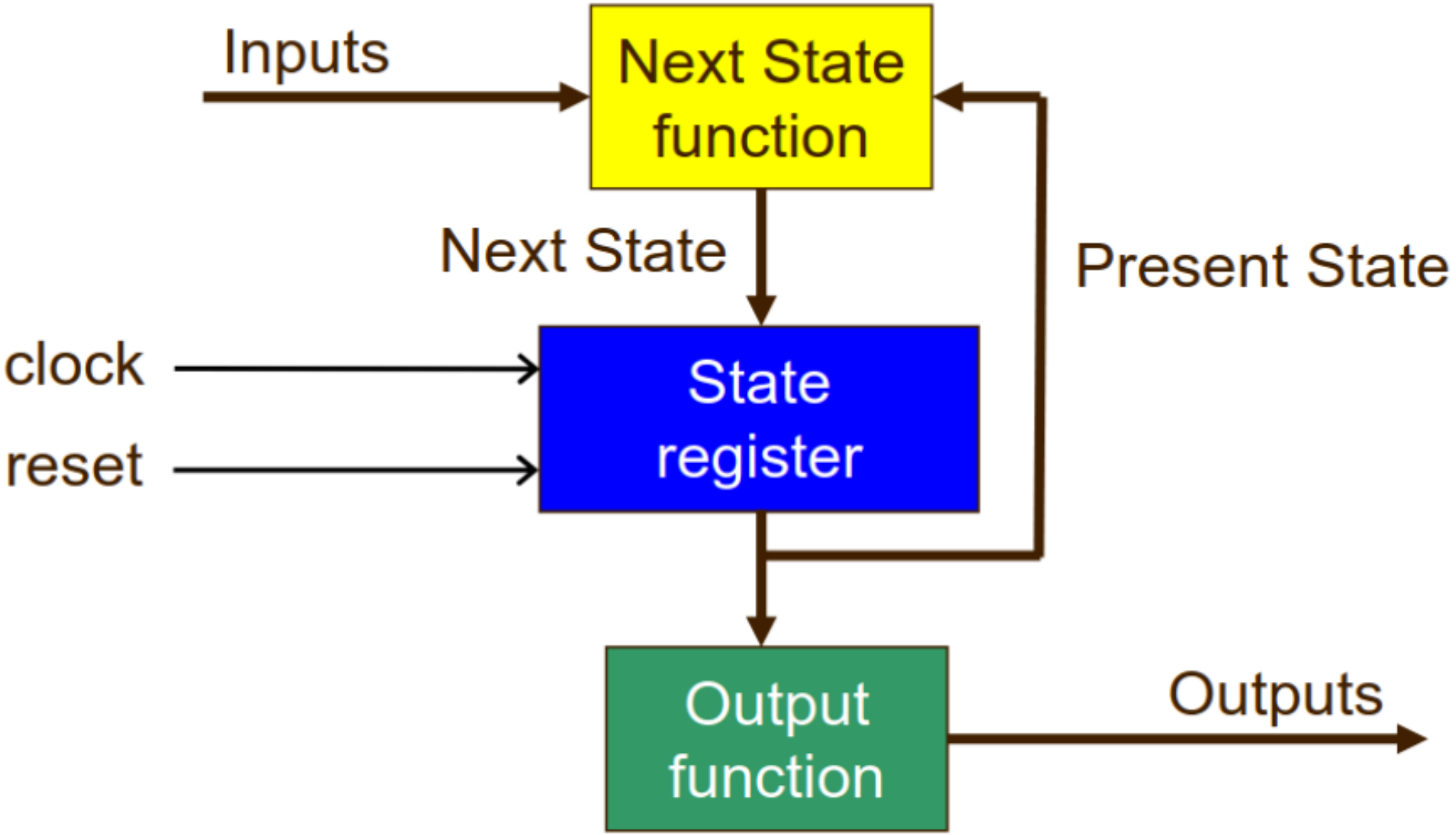
- **CODIFICA STATI**:

State	Code
C0	= "00"
C1	= "01"
C2	= "10"
C3	= "11"

Current State			Next State			Outputs
	Q1_cur	Q0_cur		Q1_nxt	Q0_nxt	CNT
C0	0	0	C1	0	1	"00"
C1	0	1	C2	1	0	"01"
C2	1	0	C3	1	1	"10"
C3	1	1	C0	0	0	"11"

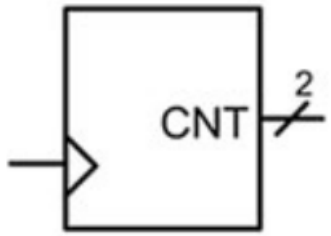
Contatore modulo 2^N con N=2

- E' una FSM di tipo **Moore** …



Contatore modulo 4 : cnt2bit.vhdl

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

-- 2 bit up counter:
entity cnt2bit is
        port( clk : in std_logic;
              cnt : out std_logic_vector(1 downto 0));
end entity;
```



Contatore modulo 4 : cnt2bit.vhdl

```
architecture arch of cnt2bit is
        type state is (C0, C1, C2, C3);
        signal current_state, next_state : state;
begin
        ---------------------------------------
        STATE_MEMORY : process(clk) is
        begin
                if(rising_edge(clk)) then
                        current_state <= next_state;
                end if;
        end process;
        ---------------------------------------
```

Contatore modulo 4 : cnt2bit.vhdl

```
-----------------------------------------
NEXT_STATE_LOGIC : process(current_state) is
begin
        case(current_state) is
                when C0 => next_state <= C1;
                when C1 => next_state <= C2;
                when C2 => next_state <= C3;
                when C3 => next_state <= C0;
                when others => next_state <= C0;
        end case;
end process;
-----------------------------------------
```

Contatore modulo 4 : cnt2bit.vhdl

```
        ------------------------------------------
        OUTPUT_LOGIC : process(current_state) is
        begin
                case(current_state) is
                        when C0 => cnt <= "00";
                        when C1 => cnt <= "01";
                        when C2 => cnt <= "10";
                        when C3 => cnt <= "11";
                        when others => cnt <= "00";
                end case;
        end process;
end architecture;
```

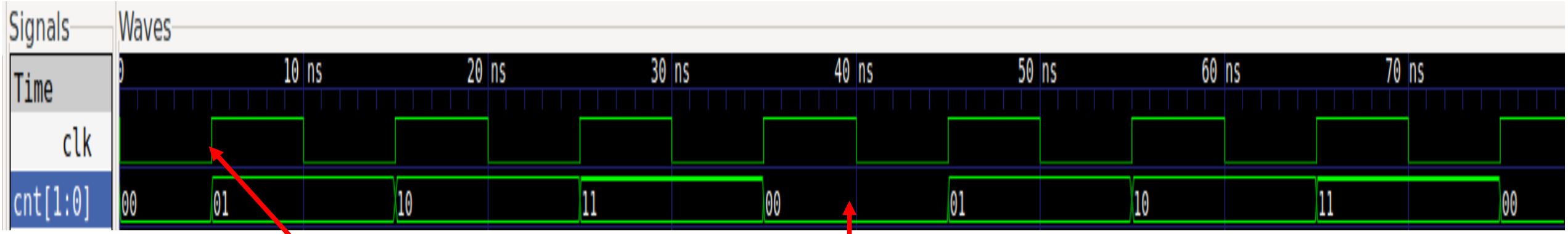
Contatore modulo 4 : cnt2bit_tb.vhdl

Impostazione valore **iniziale** segnale clk

Processo di clock

```
    -- signal to map to component
signal clk : std_logic := '1';
 signal  cnt : std_logic_vector(1 downto 0);

begin
    -- map signals
    uut: cnt2bit port map ( clk=>clk,
                            cnt=>cnt);

    process is
    begin
            clk <= not clk;
            wait for 5 ns;
    end process;

    stimulus: process is
    begin
     -- write your test here
    wait;
```

Contatore modulo 4 : analisi onda



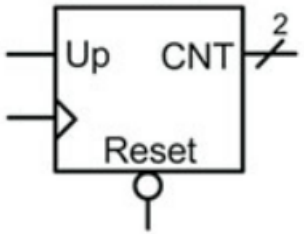
Il contatore :

1) Non ha segnali in input (se si esclude il segnale di clock)
2) Avanza ad ogni fronte di **salita** del segnale di clock
3) Raggiunto il valore massimo ricomincia a coontare da “00”

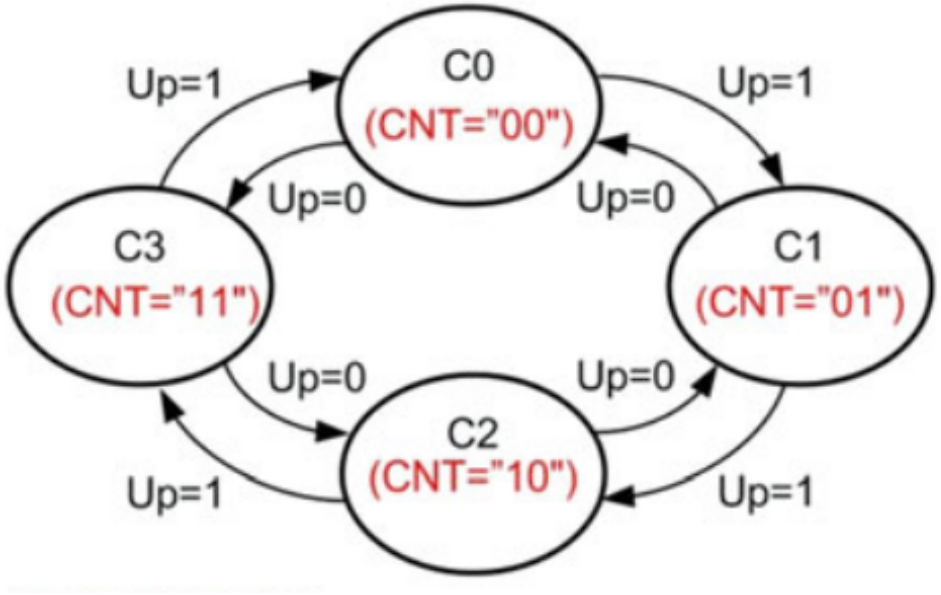
… concludiamo che analisi d’onda evidenzia il comportamento atteso.

Contatore modulo 4 Up/Down

- 2 bit up/down counter binario. Il contatore **incrementa** di 1 ad ogni singolo fronte di **salita** del clock (00, 01, 10, 11) SE Up e' alto ALTRIMENTI **decrementa**. Quando il contatore raggiunge 11 riparte a contare da 00 (direzione: incremento). Quando raggiunge 00 riparte da 11 (direzione: decremento)La linea di output, composta da 2 bit, ha nome CNT.



- **STD e STT**:



	(Input)		(Output)
Current State	Up	Next State	CNT
C0	0	C3	"00"
	1	C1	
C1	0	C0	"01"
	1	C2	
C2	0	C1	"10"
	1	C3	
C3	0	C2	"11"
	1	C0	

Contatore modulo 4 Up/Down

- 2 bit up/down counter binario. Il contatore **incrementa** di 1 ad ogni singolo fronte di **salita** del clock (00, 01, 10, 11) SE Up e' alto altrimenti **decrementa**. Quando il contatore raggiunge 11 riparte a contare da 00 (direzione: incremento). Quando raggiunge 00 riparte da 11 (direzione: decremento)La linea di output, composta da 2 bit, ha nome CNT.

- **CODIFICA STATI**:

State	Code
C0	= "00"
C1	= "01"
C2	= "10"
C3	= "11"

Current State			Input	Next State			Outputs
	Q1_cur	Q0_cur	Up		Q1_nxt	Q0_nxt	CNT
C0	0	0	0	C3	1	1	"00"
C0	0	0	1	C1	0	1	"00"
C1	0	1	0	C0	0	0	"01"
C1	0	1	1	C2	1	0	"01"
C2	1	0	0	C1	0	1	"10"
C2	1	0	1	C3	1	1	"10"
C3	1	1	0	C2	1	0	"11"
C3	1	1	1	C0	0	0	"11"

Contatore modulo 4 : cnt2bitud.vhdl

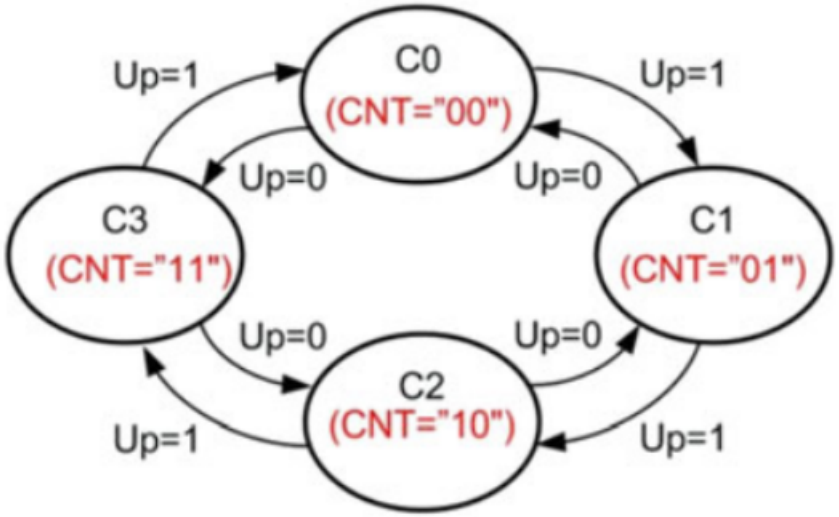
```
library ieee;
use ieee.std_logic_1164.all;

-- 2 bit up counter:
entity cnt2bitud is
        port( clk : in std_logic;
              reset : in std_logic;
              up : in std_logic;
              cnt : out std_logic_vector(1 downto 0));
end entity;

architecture arch of cnt2bitud is
        type state is (C0, C1, C2, C3);
        signal current_state, next_state : state;
begin
```

Contatore modulo 4 : cnt2bitud.vhdl

```
begin
        ------------------------------------------
        STATE_MEMORY : process(clk, reset) is
        begin
                if(reset = '0') then
                        current_state <= C0;
                elsif(rising_edge(clk)) then
                        current_state <= next_state;
                end if;
        end process;
        ------------------------------------------
```

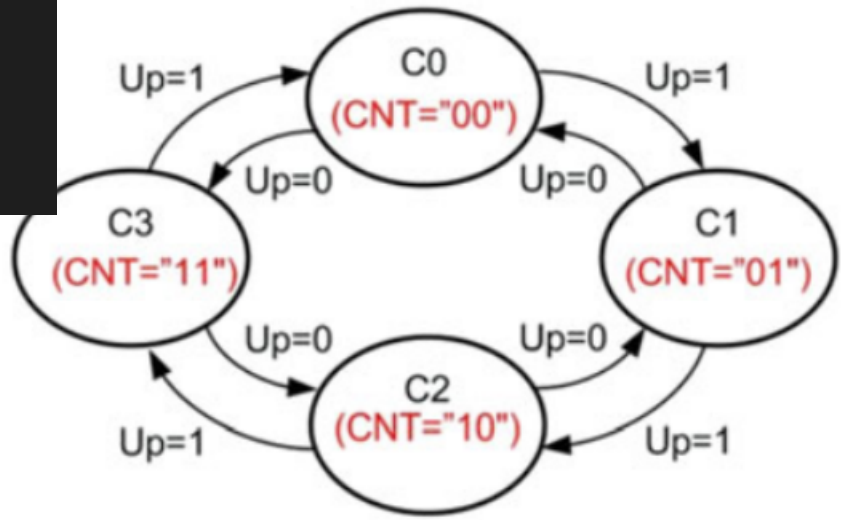
Contatore modulo 4 : cnt2bitud.vhdl

```
-----------------------------------------
NEXT_STATE_LOGIC : process(current_state, up) is
begin
        case(current_state) is
                when C0 => if(up = '1') then
                                next_state <= C1;
                           else
                                next_state <= C3;
                           end if;
                when C1 => if(up = '1') then
                                next_state <= C2;
                           else
                                next_state <= C0;
                           end if;
                when C2 => if(up = '1') then
                                next_state <= C3;
                           else
                                next_state <= C1;
                           end if;
                   when C3 => if(up = '1') then
                                next_state <= C0;
                           else
                                next_state <= C2;
                           end if;
                when others => next_state <= C0;
        end case;
end process;
-----------------------------------------
```



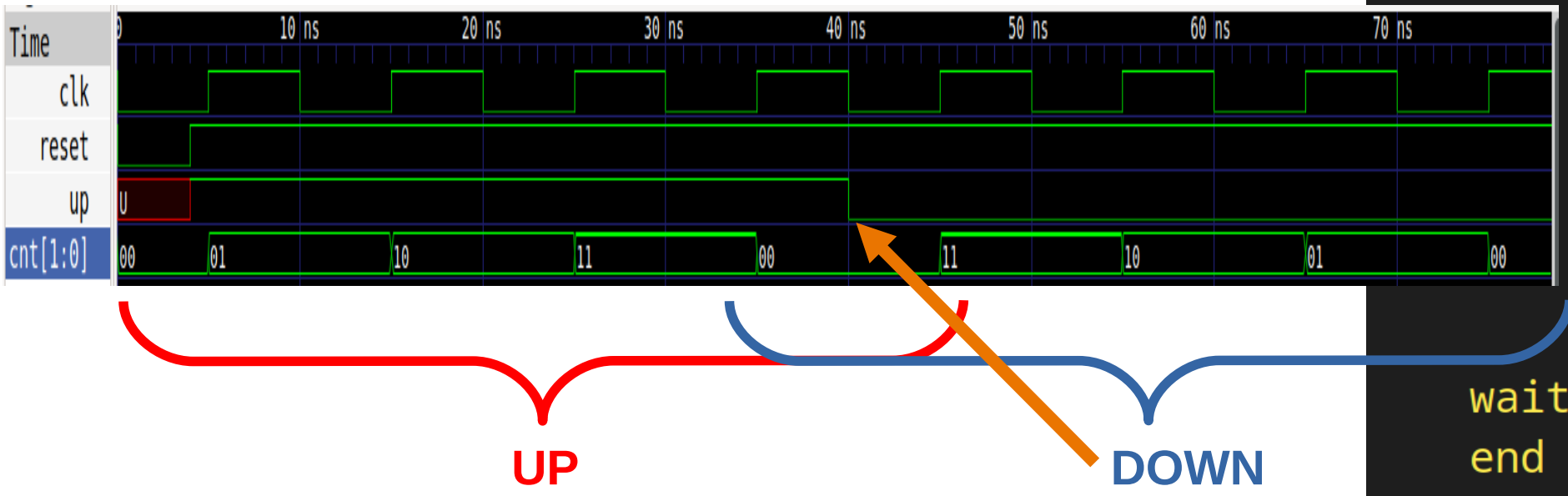
Contatore modulo 4 : cnt2bitud.vhdl

```
        ------------------------------------------
        OUTPUT_LOGIC : process(current_state) is
        begin
                case(current_state) is
                        when C0 => cnt <= "00";
                        when C1 => cnt <= "01";
                        when C2 => cnt <= "10";
                        when C3 => cnt <= "11";
                        when others => cnt <= "00";
                end case;
        end process;
end architecture;
```



Contatore modulo 4 : cnt2bitud_tb.vhdl

ANALISI ONDA:




```
    process is
    begin
            clk <= not clk;
            wait for 5 ns;
    end process;

    stimulus: process is
    begin
     -- write your test here
            reset <= '0';
            wait for 4 ns;
            reset <= '1';
            up <= '1';
            wait for 36 ns;
            up <= '0';
            wait for 40 ns;
    wait;
    end process;
end architecture;
```

Contatori: descrizione con 1 processo

- **RECAP:** un contatore e' una sottoclasse di FSM con vincoli logici specifici. STD e' **ciclico** e viene attraversato **SEMPRE in modo lineare** (ci si sposta solo lungo l'anello). La direzione di percorrenza dell'anello puo' essere specificata mediante opportuni segnali di controllo.
- Il valore emesso sulla linea di output, data la contiguita' degli stati, puo' essere impostato come **coincidente** con la codifica binaria degli stati.
- **SONO PIU' SEMPLICI DI ALTRE FSM:** VHDL permette la loro descrizione in modo piu' compatto applicando ad essi regole speciali … in particolare permette di desciverli utilizzando l'operatore **+** e descrivendo il loro comportamento in un **solo** processo.

PROBLEMA : l'operatore + **NON** e' definito per il tipo std_logic_vector ! Dobbiamo utilizzare un **tipo di dato diverso**! … **UNSIGNED**!

Contatori: descrizione con 1 processo

Descrivere in VHDL un contatore su **4 bit** che si muove **solo in avanti** (up). La linea di output si chiama **CNT**. Il contatore parte dal valore 0000 e, una volta raggiunto il valore 1111 riparte da 0000. Il contatore risponde anche ad un segnale di controllo reset che, se basso, riporta il contatore a 0000 in modo asincrono!

Per poter utilizzare il tipo UNSIGNED dobbiamo includere un package aggiuntivo dalla libreria ieee :

ieee.**numeric_std**

cnt4bitu.vhdl

Contatori: descrizione con 1 processo

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

entity cnt4bitu is
        port(clk : in std_logic;
             reset : in std_logic;
             cnt : out unsigned(3 downto 0));
end entity;

architecture arch of cnt4bitu is
        signal CNT_tmp : unsigned(3 downto 0);
begin
        COUNTER : process(reset, clk) is
        begin
                if(reset = '0') then
                        CNT_tmp <= "0000";
                elsif(rising_edge(clk)) then
                        CNT_tmp <= CNT_tmp + 1;
                end if;
        end process;
        CNT <= CNT_tmp;
end architecture;
```

numeric_std e' necessario per avere a disposizione l'operatore + ma esso funziona solamente sul tpo unsigned. Dichiariamo l'**uscita** di tipo **unsigned**!

cnt4bitu.vhdl

Contatori: descrizione con 1 processo

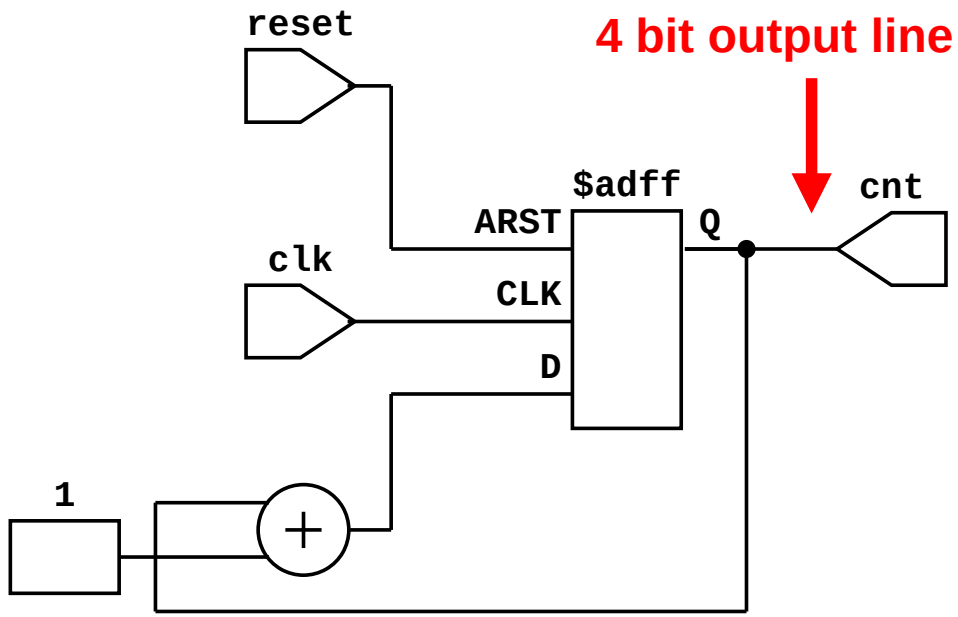
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cnt4bitu is
        port(clk : in std_logic;
             reset : in std_logic;
             cnt : out unsigned(3 downto 0));
end entity;

architecture arch of cnt4bitu is
        signal CNT_tmp : unsigned(3 downto 0);
begin
        COUNTER : process(reset, clk) is
        begin
                if(reset = '0') then
                        CNT_tmp <= "0000";
                elsif(rising_edge(clk)) then
                        CNT_tmp <= CNT_tmp + 1;
                end if;
        end process;
        CNT <= CNT_tmp;
end architecture;
```

Un segnale interno e' **necessario** per poter effettuare un assegnamento a segnale nella forma **C <= C + 1** poiche' **una porta NON puo' essere utilizzata** in un assegnamento!

cnt4bitu.vhdl

Contatori: descrizione con 1 processo

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

entity cnt4bitu is
        port(clk : in std_logic;
             reset : in std_logic;
             cnt : out unsigned(3 downto 0));
end entity;

architecture arch of cnt4bitu is
        signal CNT_tmp : unsigned(3 downto 0);
begin
        COUNTER : process(reset, clk) is
        begin
                if(reset = '0') then
                        CNT_tmp <= "0000";
                elsif(rising_edge(clk)) then
                        CNT_tmp <= CNT_tmp + 1;
                end if;
        end process;
        CNT <= CNT_tmp;
end architecture;
```



Un assegnamento a segnale **di tipo <u>concorrente</u>** e' utilizzato per aggiornare continuamente il valore del segnale in uscita assegnando ad esso il valore di CNT_tmp .

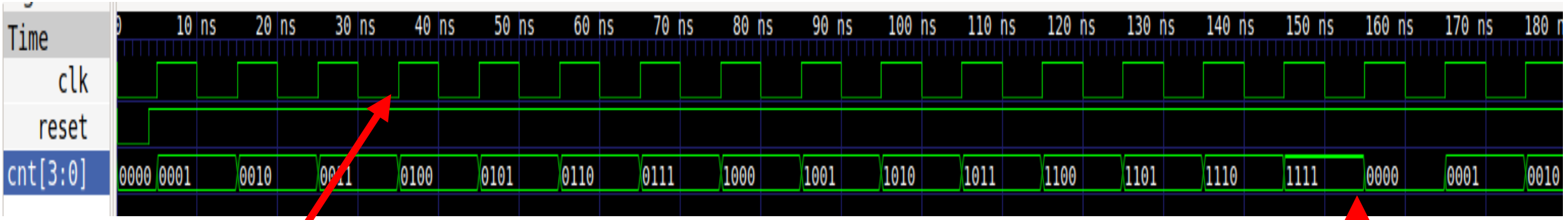
cnt4bitu_tb.vhdl

Contatori: descrizione con 1 processo

```
    process is
    begin
            clk <= not clk;
            wait for 5 ns;
    end process;

    stimulus: process is
    begin
     -- write your test here
            reset <= '0';
            wait for 4 ns;
            reset <= '1';
            wait for 346 ns;
    wait;
    end process;
end architecture;
```

Contatori: descrizione con 1 processo

Descrivere in VHDL un contatore su **4 bit** che si muove **solo in avanti** (up). La linea di output si chiama **CNT**. Il contatore parte dal valore 0000 e, una volta raggiunto il valore 1111 riparte da 0000. Il contatore risponde anche ad un segnale di controllo reset che, se basso, riporta il contatore a 0000 in modo asincrono!



Il valore del contatore si aggiorna in corrispondenza dei fronti di **salita** del segnale di clock.

Una volta raggiunto il valore massimo (1111) il contatore riparte da 0000 senza la necessita di boundary check nel codice VHDL … dopodiche' continua …

Utilizzando questo metodo di descrizione (unsigned+mono processo) e' possibile descrivere contatori espressi su un numero **ARBITRARIO di bit** !

Esercizio: contatore modulo 64

- Contatore modulo 64: opera su 6 bit (2^6=64).
- Quando il conteggio raggiunge il valore massimo, torna a 000000.
- Modificare **cnt4bitu.vhdl** per produrre cnt6bitu.vhdl
- In fase di testbench servono 63 fronti di salita del clock (uno ogni 10 secondi se il processo di clock inverte il valore di clk ogni 5 ns. Quindi il tempo di simulazione e' 640 ns (63 * 10 ns + 10 ns per osservare il ritorno a 000000).

Esercizio: contatore con caricamento

- Servono 2 segnali in piu’ in ingresso. DATA riceve un numero senza segno espresso su N bit dove N e’ anchel’ampiezza dell’uscita e LOAD che, se alto, causa la scrittura del valore nel segnale interno **CNT_tmp**.
- Gestite la scrittura in CNT_tmp nel processo COUNTER (quello che gestisce reset e incremento).
- Non preoccupatevi di aggiornare CNT al valore di CNT_tmp (ricordatevi che questo avviene con assegnamento concorrente out of process.
- Modificate **cnt4bitu.vhdl** per produrre **cnt4bituload.vhdl**. Adattate il precedente testbench per verificare il comportamento di load.