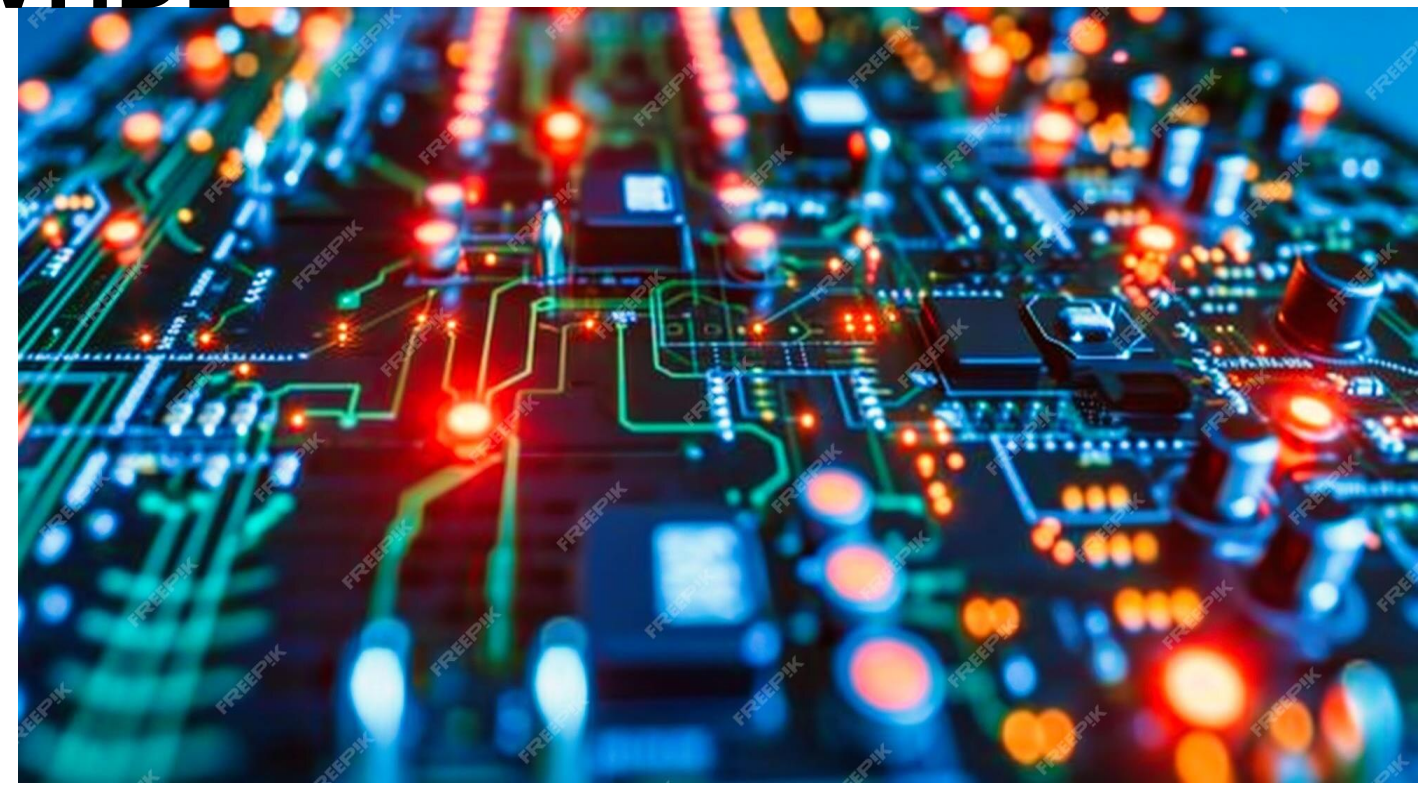
LAE1 VHDL

LAB6

registri



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

a.a. 2025-26

matteo.re@unimi.it

Obiettivi di apprendimento

- **Descrizione** di registri in VHDL
  - Descrizione RTL
  - Descrizione strutturale
- **Tipi** di registro (e relativi testbench)
  - Serial in / serial out : SISO
  - Parallel in / serial out : PISO
  - Parallel in / parallel out : PIPO
  - Universal shift register (USR)
- **Register file**
  - Cosa e' un register file?
  - Esempio (riotto) 8x8

Registro

Definizione generale: "Contenitore di informazione composto da piu' elementi di memoria".

Su di esso sono **definite 3 operazioni:**

I) **Scrittura:** posizionamento di un determinato elemento di memoria che in un determinato stato e tale stato viene mantenuto nel tempo.

II) **Lettura:** Rilievo dello stato corrente di un elemento di memoria

III) **Selezione:** individuazione di un elemento di memoria al fine di compiere una delle operazioni precedenti.

Registro

Definizione generale: "Contenitore di informazione composto da piu' elementi".

Puo' essere visto in diversi modi:

- **Insieme** (banco) di componenti hardware in grado di memorizzare lo stato di un bit (ad es. D flip-flop)
- Vettore **ordinato** di elementi omogenei
- All'interno di una gerarchia di componenti puo' essere **esso stesso** parte di un set di elementi di memoria

Registri

In un HDL **deve** essere possibile descrivere il registro in modalita' differenti !

Puo' essere visto, **E QUINDI DESCRITTO**. in **diversi** modi:

- **Insieme** (banco) di **componenti hardware** in grado di memorizzare lo stato di un bit (ad es. D flip-flop) : descrizione di tipo **strutturale**.
- Vettore **ordinato** di elementi omogenei (std_logic_vector) : descrizione di tipo register transfer level (**RTL**)

Registri

Modalita' di descrizione dei registri in VHDL.

- **RTL, Register transfer level:** livello di astrazione della descrizione in cui dati in forma vettoriale vengono manipolati in modo sincrono (e' la modalita' **piu' adatta** per la descrizione dei registri).
- **Strutturale:** Riutilizzo di componenti le cui architetture contengono descrizioni minimali (del calcolo delle uscite) di tipo comportamentale
- **Comportamentale:** Modalita' di descrizione ad alto livello. Si adatta a tutto ma non e' la piu' adatta da usare in quanto piu' ambigua. Non sempre sintetizzabile.

Nuovi concetti VHDL

Nuovi tipi definiti utilizzando **type** :

```
type arv is array (0 to 7) of std_logic_vector(7
downto 0);
```

Array composto da 8 std_logic_vector (ognuno di lunghezza 8). Inizializzazione del nuovo tipo di dato:

```
signal regset : arv := arv'(x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00");
```

Nuovi concetti VHDL

Selezione (pre lettura/scrittura) di singoli elementi di tipi vettoriali:

```
A : in std_logic_vector(7 downto 0);
B : out std_logic_vector(7 downto 0);
-- assegnamento in blocco (forma breve hex):
A <= x"AA";
-- contenuto di A :
-- "10101010"
```

Nuovi concetti VHDL

Ripasso teoria: tavola di conversione Hex to Bin :

Hexadecimal Digit	Decimal Digit	Binary Digit
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Nuovi concetti VHDL

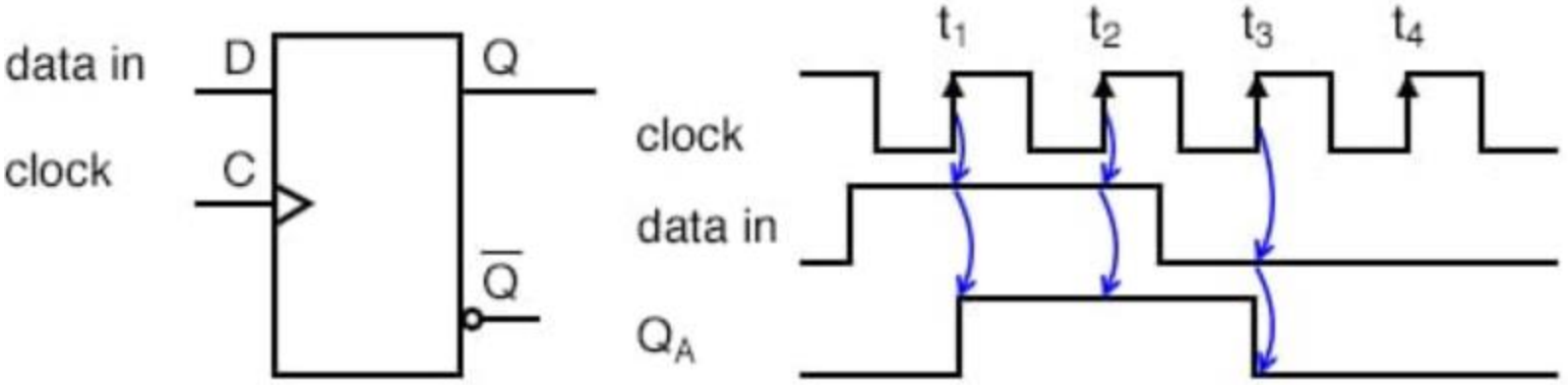
Selezione (pre lettura/scrittura) di sottoinsiemi di tipi vettoriali:

```
A : in std_logic_vector(7 downto 0);
B : out std_logic_vector(7 downto 0);
C : in std_logic;
-- assegnamento in blocco (forma breve hex):
A <= x"AA"; -- A contiene "10101010"e
C <= '1';
B <= A(6 downto 0) & C; -- B contiene"01010101"
--                      & : concatenamento
C <= A(2)  -- C contiene '0';
```

Registri

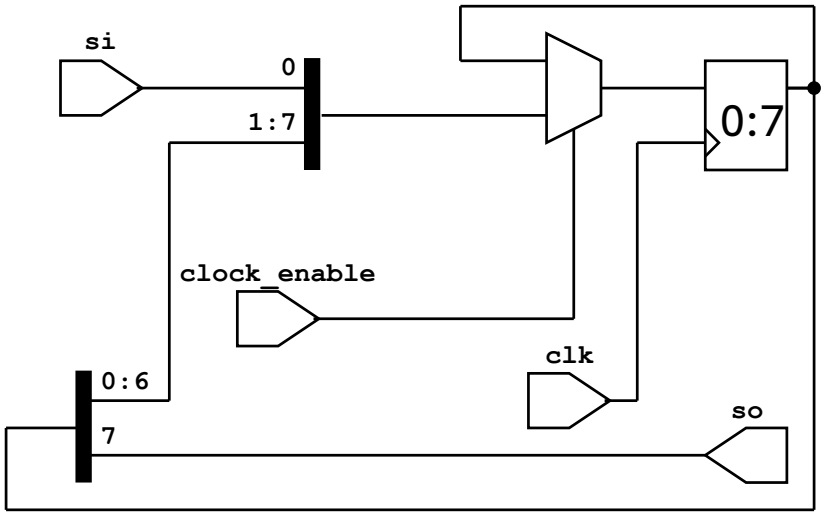
Ripasso teoria: I registri possono essere divisi in macrocategorie basate sulle loro modalita' di input, output e comportamento.

Serial in / Serial out: SISO register



Serial in / serial out : SISO

- In questo tipo di registro la linea di input ha ampiezza pari a 1 bit.
- Anche la linea di output ha ampiezza 1 bit.
- **Internamente** il registro e' composto da **N** elementi di memoria.
- Ad ogni ciclo di clock avviene uno shift a sinistra (verso MSB) degli N-1 bit meno significativi.
- Il nuovo valore da porre nel bit meno significativo (LSB) e' letto dalla linea di input.
- Sulla linea di output viene scritto sempre il valore corrente del bit piu' significativo (MSB)



siso.vhdl

Serial in / serial out : SISO

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

entity siso is
port(
    -- beside values related ports registers usually have a control line enabling them
    -- to be responsive or not to the clock signal. This is clock_enable. We also have
    -- a clock in line.
    clock_enable, clk, si : in std_logic;       -- serial IN  (read 1 bit at time): si
    so : out std_logic                          -- serial OUT (output 1 bit at time): so
);
end siso;

architecture arch of siso is
    -- Registers are usually described using the std_logic_vector type. To keep it simple
    -- we model a small 8 bit register (but the logic is the same for any register size).
    signal temp : std_logic_vector (7 downto 0) := x"00";
    begin
        process (clk)
        begin
                if (clk='1' and clock_enable='1') then
                -- if rising_edge(clock) advance 6 LSBs toward MSB (and overwrite it)
                        temp(7 downto 1) <= temp(6 downto 0);
                -- and overwrite the post update LSB using the si line actual value
                        temp(0) <= si;
                end if;
        end process;
    -- after process execution is complete output the updated MSB on so line
    so <= temp(7);
end architecture;
```

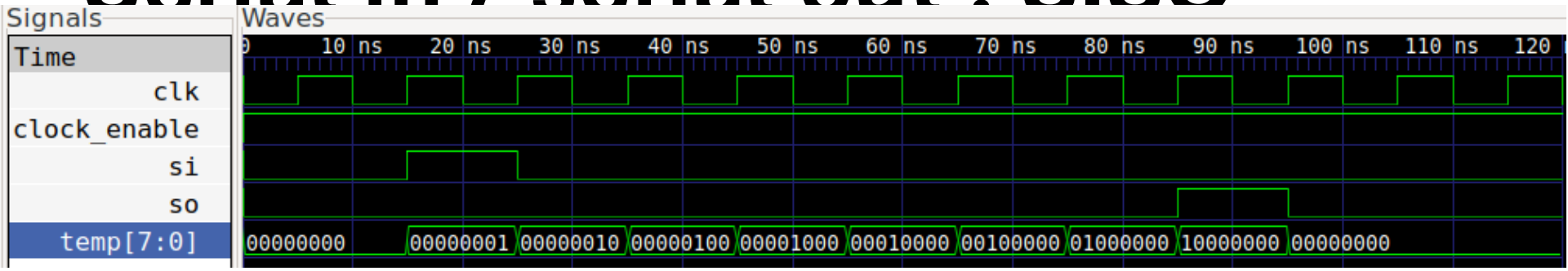
siso_tb.vhdl

Serial in / serial out : SISO

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

entity siso_tb is
end entity siso_tb;

architecture arch of siso_tb is
    -- component to test
        component siso is
        port(   clock_enable, si  :  in std_logic;
                clk : in std_logic;
                so  :  out std_logic);
        end component;
    -- signal to map to component
        signal clock_enable, si : std_logic;
        signal clk : std_logic := '1';
        signal  so : std_logic;
begin
    -- map signals
    uut : siso port map ( clock_enable => clock_enable,
                                clk => clk,
                                si => si,
                                so =>so);
        process is
        begin
            clk <= not clk;
            wait for 5 ns;
        end process;

    stimulus: process is
    begin
     -- write your test here
            si <= '0';
            clock_enable <= '1';
            wait for 15 ns;
            si <= '1';
            wait for 10 ns;
            si <= '0';
            wait for 55 ns;
    wait;
    end process;
end architecture;
```

simulazione

Serial in / serial out : SISO




```
ghdl -a siso.vhdl
ghdl -a siso_tb.vhdl
ghdl -e siso_tb
./siso_tb --stop-time=120ns --wave=siso.ghw
gtkwave siso.ghw
```

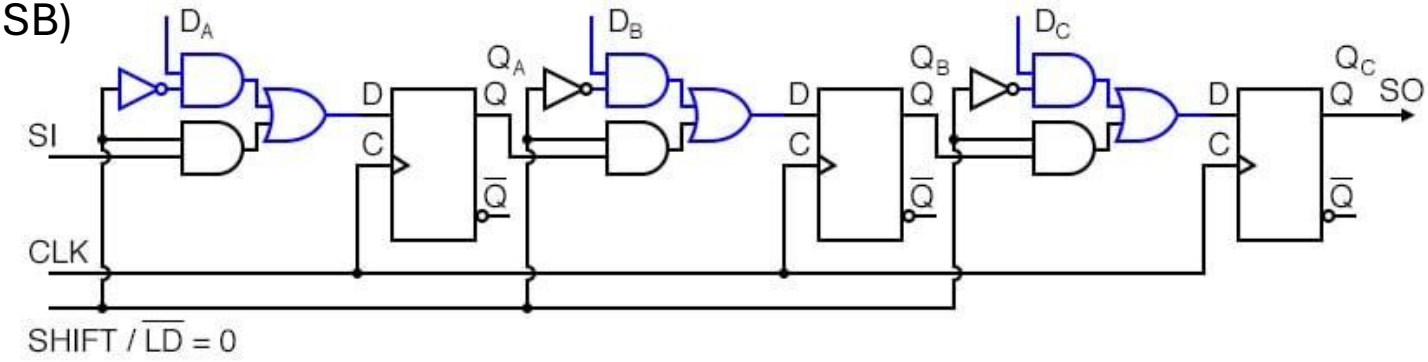
Domande:

- La direzione di shift e' quella attesa?
- Quanti fronti di salita del segnale di clock sono necessari perche' l'1 possa "percorrere" l'intero registro?

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

    stimulus: process is
    begin
     -- write your test here
            si <= '0';
            clock_enable <= '1';
            wait for 15 ns;
            si <= '1';
            wait for 10 ns;
            si <= '0';
            wait for 55 ns;
    wait;
    end process;
end architecture;
```

Parallel in / serial out : PISO

- In questo tipo di registro la linea di **input** ha ampiezza pari a quella del registro.
- La linea di **output** ha ampiezza 1 bit.
- **Internamente** il registro e' composto da **N** elementi di memoria.
- Ad ogni ciclo di clock avviene uno shift a sinistra (verso MSB) degli N-1 bit meno significativi.
- Il nuovo valore da porre nel bit meno significativo (LSB) e' sempre '0'.
- Sulla linea di output viene scritto sempre il valore corrente del bit piu' significativo (MSB)



Parallel-in/ serial-out shift register showing parallel load path

piso.vhdl

Parallel in / serial out : PISO

```vhdl
library ieee;
use ieee.std_logic_1164.all;
-- Parallel in/serial out: PISO register with load signal
entity piso is
port(
clk,load : in std_logic;         -- when load=1 the content of pi is bitwise copied into temp
    pi : in std_logic_vector(7 downto 0);  -- input (8 bit)
    so : out std_logic);         -- outpit (1 bit)
end entity;

architecture arch of piso is
        signal t : std_logic := '0';                    -- 1 bit signal for tmp output operations
        signal temp : std_logic_vector(7 downto 0); -- 8 bit signal for tmp whole reg operations
    begin
        process (clk,pi,load)
        begin
                if (load='1') then
                        temp(7 downto 0) <= pi(7 downto 0);
                elsif (clk='1') then
                        t <= temp(7);
                        temp(7 downto 1) <= temp(6 downto 0);
                        temp(0) <= '0';
                end if;
        end process;
    so <= t;
end architecture;
```

Piso_tb.vhdl

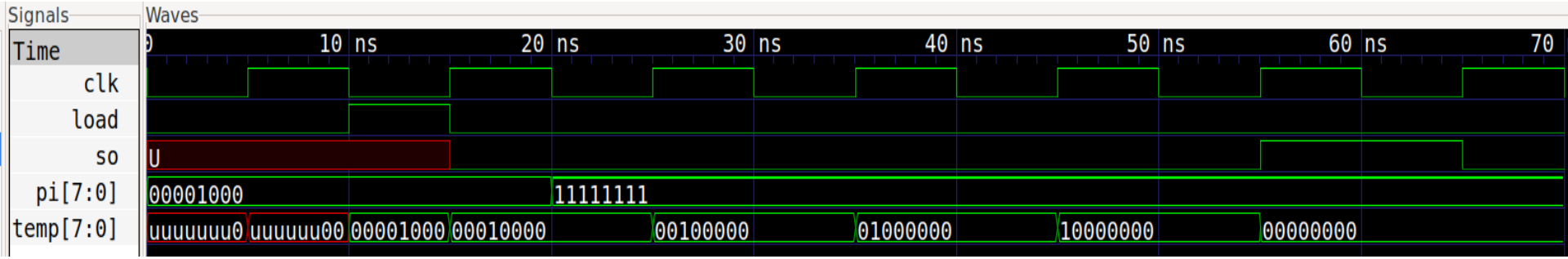
Parallel in / serial out : PISO

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

entity piso_tb is
end entity piso_tb;

architecture arch of piso_tb is
    -- component to test
component piso is
port(   clk,load  :  in std_logic;
        pi  :  in std_logic_vector(7 downto 0);
        so  :  out std_logic);
end component;
    -- signal to map to component
signal clk  : std_logic := '1';
signal load : std_logic := '0';
-- 00 hexadecimal is 00000000 binary
signal  pi : std_logic_vector(7 downto 0) := x"00";
signal  so : std_logic;
begin
    -- map signals
    uut: piso port map ( clk=>clk,
                         load=>load,
                         pi=>pi,
                         so=>so);
```

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

    stimulus: process is
    begin
     -- write your test here
            pi <= "00001000";
            wait for 10 ns;
            load <= '1';
            wait for 5 ns;
            load <= '0';
            wait for 5 ns;
            pi <= "11111111";
            wait for 5 ns;
    wait;
    end process;
end architecture;
```

simulazione

Parallel in / serial out : PISO




```
ghdl -a piso.vhdl
ghdl -a piso_tb.vhdl
ghdl -e piso_tb
./piso_tb --stop-time=120ns --wave=piso.ghw
gtkwave piso.ghw
```

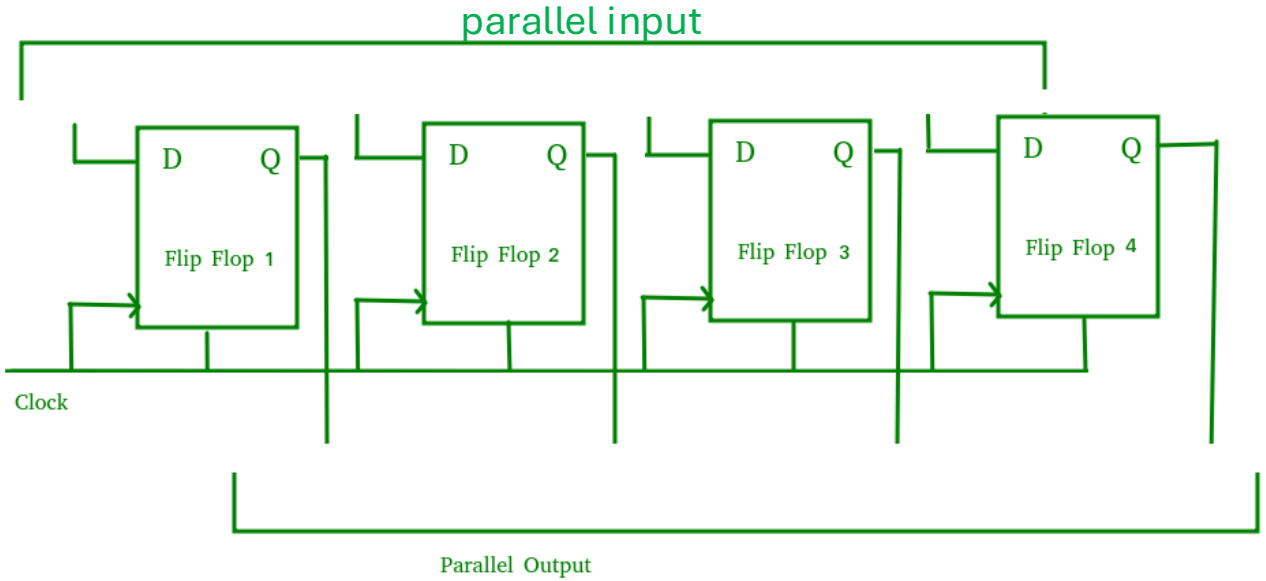
```
    process is
    begin
            clk <= not clk;
            wait for 5 ns;
    end process;

    stimulus: process is
    begin
     -- write your test here
            pi <= "00001000";
            wait for 10 ns;
            load <= '1';
            wait for 5 ns;
            load <= '0';
            wait for 5 ns;
            pi <= "11111111";
            wait for 5 ns;
    wait;
    end process;
end architecture;
```

Domande:

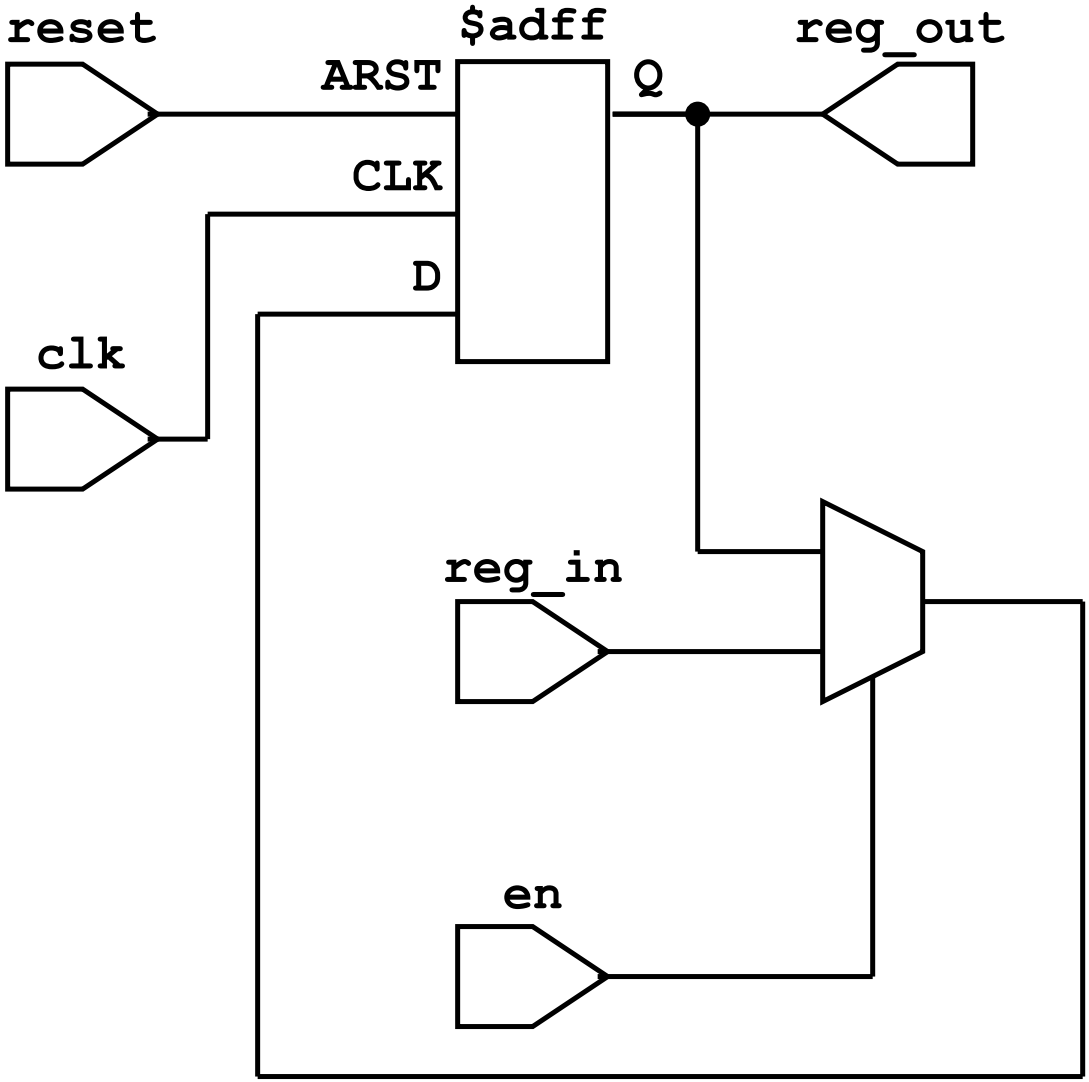
- La direzione di shift e' quella attesa?
- Quanti fronti di salita del segnale di clock sono necessari perche' la parola da 8 bit caricata possa "percorrere" l'intero registro?

Parallel in / parallel out : PIPO

- In questo tipo di registro la linea di **input** ha ampiezza pari a quella del registro.
- La linea di **output** ha ampiezza pari a quella del registro.
- **Internamente** il registro e' composto da **N** elementi di memoria.
- Ad ogni ciclo di clock I dati sulla linea di input vengono scritti nel registro
- Sulla linea di output viene scritto sempre so stato attuale di tutti gli elementi di memoria che compongono il registro



Parallel in / parallel out : PIPO



pipo.vhdl

Parallel in / parallel out : PIPO

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

entity pipo is
        port(clk        : in std_logic;
             reset      : in std_logic;
             reg_in     : in std_logic_vector(7 downto 0);
             en         : in std_logic;
             reg_out    : out std_logic_vector(7 downto 0) := x"00");
end entity;

architecture arch of pipo is
begin
        reg_proc : process(clk, reset)
        begin
                if(reset = '1') then
                        reg_out <= x"00";
                elsif(rising_edge(clk)) then
                        if(en='1') then
                                reg_out <= reg_in;
                        end if;
                end if;
        end process;
end architecture;
```

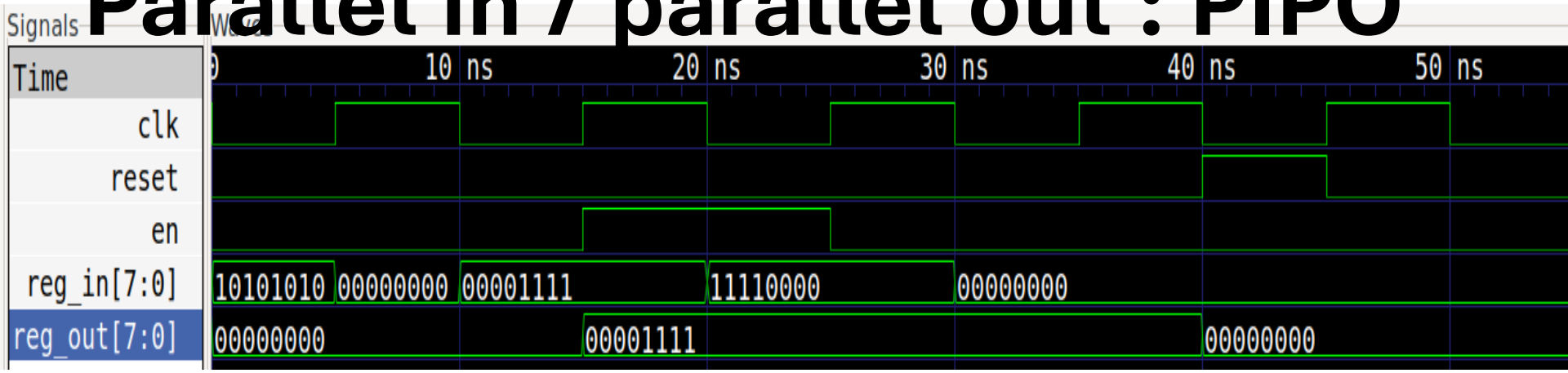
pipo_tb.vhdl

Parallel in / parallel out : PIPO

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

entity pipo_tb is
end entity pipo_tb;

architecture Behavior of pipo_tb is

    -- component to test
component pipo is
        port(clk         :  in std_logic;
             reset       :  in std_logic := '0';
             reg_in      :  in std_logic_vector(7 downto 0);
             en          :  in std_logic;
             reg_out     :  out std_logic_vector(7 downto 0)  := x"00");
end component;

    -- signal to map to component
signal clk : std_logic := '1';
 signal  reset : std_logic := '0';
 signal  reg_in : std_logic_vector(7 downto 0);
 signal  en : std_logic := '0';
 signal  reg_out : std_logic_vector(7 downto 0) := x"00";

begin
    -- map signals
    uut: pipo port map ( clk=>clk,
                         reset=>reset,
                         reg_in=>reg_in,
                         en=>en,
                         reg_out=>reg_out);
    process is
    begin
            clk <= not clk;
            wait for 5 ns;
    end process;

    stimulus: process is
    begin
     -- write your test here
            reg_in <= "10101010";
            wait for 5 ns;
            reg_in <= x"00";
            wait for 5 ns;
            reg_in <= "00001111";
            wait for 5 ns;
            en <= '1';
            wait for 5 ns;
            reg_in <= "11110000";
            wait for 5 ns;
            en <= '0';
            wait for 5 ns;
            reg_in <= x"00";
            wait for 10 ns;
            reset <= '1';
            wait for 5 ns;
            reset <= '0';
            wait for 5 ns;
    wait;
    end process;
end architecture behavior;
```

simulazione

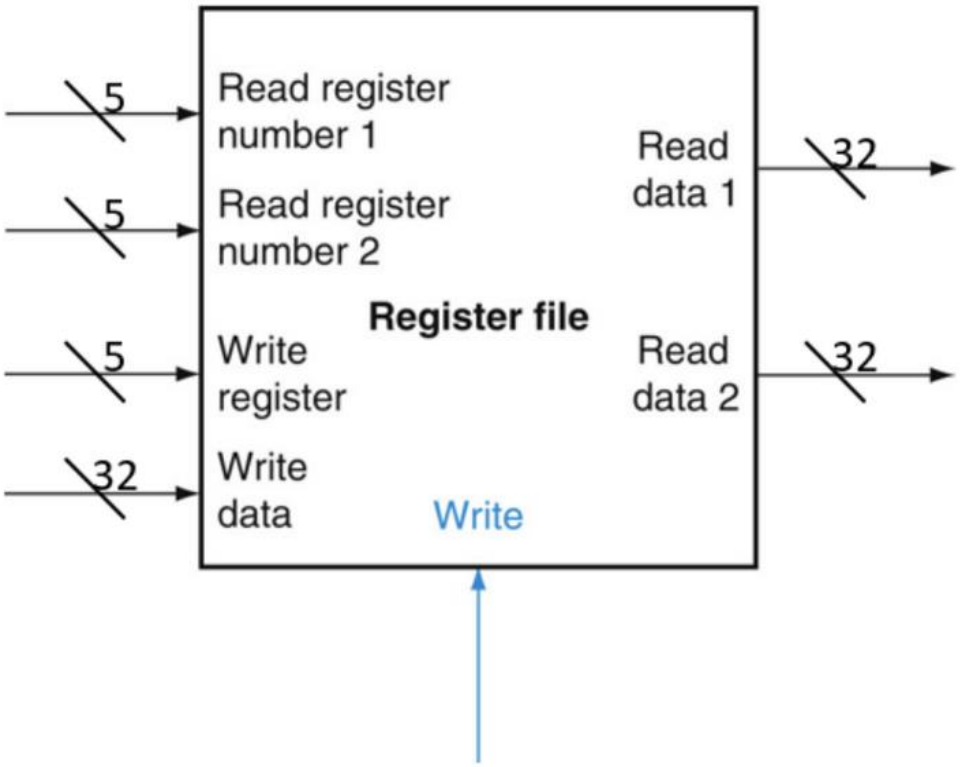
Parallel in / parallel out : PIPO



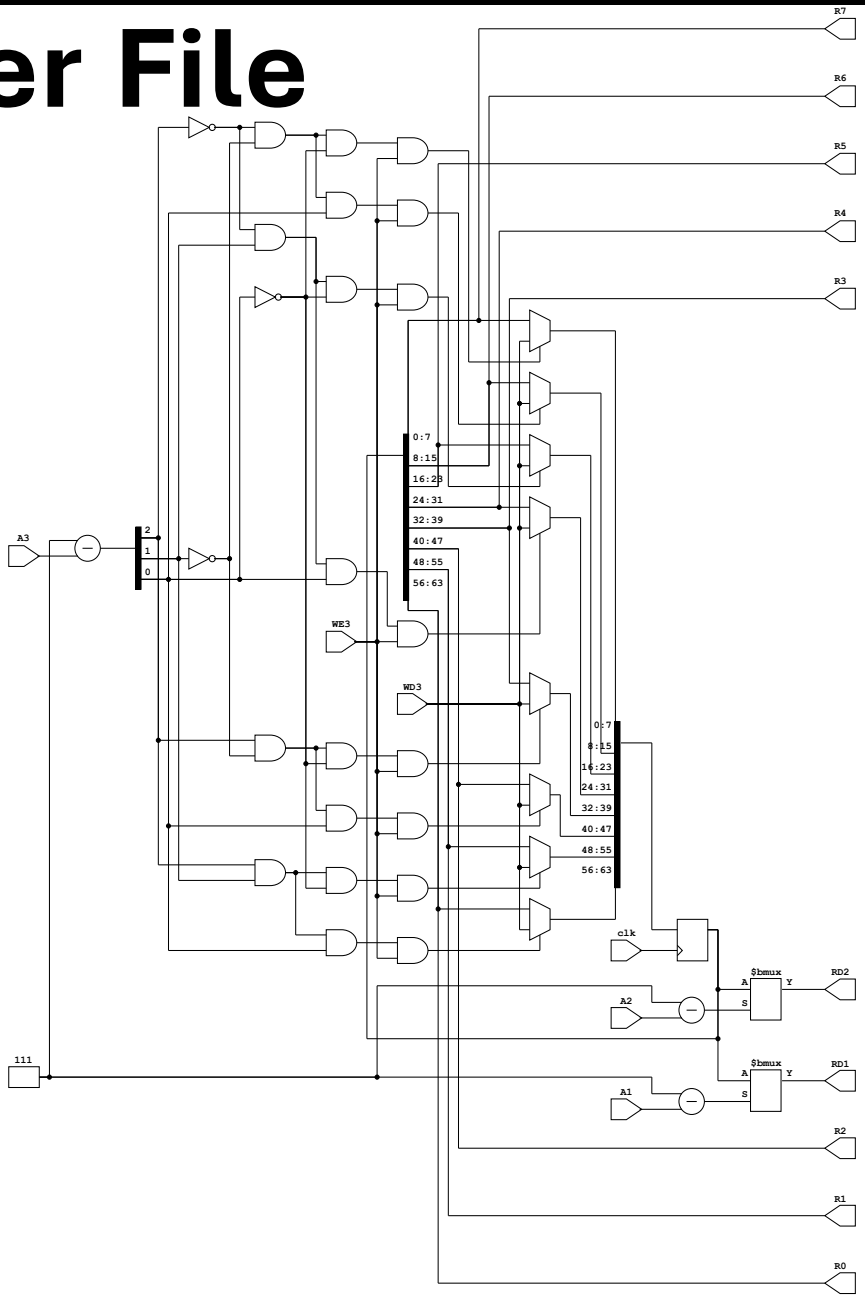

```
reg_in <= "10101010";
wait for 5 ns;
reg_in <= x"00";
wait for 5 ns;
reg_in <= "00001111";
wait for 5 ns;
en <= '1';
wait for 5 ns;
reg_in <= "11110000";
wait for 5 ns;
en <= '0';
wait for 5 ns;
reg_in <= x"00";
wait for 10 ns;
reset <= '1';
wait for 5 ns;
reset <= '0';
wait for 5 ns;
```

Register File

- Unita' funzionale che rappresenta un collezione di registri
- In questo esempio … 2 linee in ingres specificano 2 registri di cui effettuare letture. I dati letti vengono scritti su due linee di output dedicate
- Una inea in ingresso specifica quale registro scrivere
- Una linea in ingresso acquisizce I dat da scrivere nel registro selezionato.
- Un segnale in ingresso Write attua la scrittura nel registro selezionato attraverso la linea Write register.



Register File



Register File

- Riceverete il file vhdl della descrizione del register file 8x8 ed esso verra' commentato in classe
- Scrivete un testbench per questa unita' funzionale
- Simulate

Compito per la prossima lezione :

Modificate il sorgente registerfile8x8.vhdl in modo che non descriva piu' una collezione di 8 registri da 8 bit chascuno ma 32 registri da 32 bit ciascuno.

Non e' richiesto (se non volete farla) la simulazione ma assicuratevi che il sorgent modificato sia in grado di superare gli step di analisi ed elaborazione.