Progetto di Sistemi Elettronici Digitali per l'Internet of Things

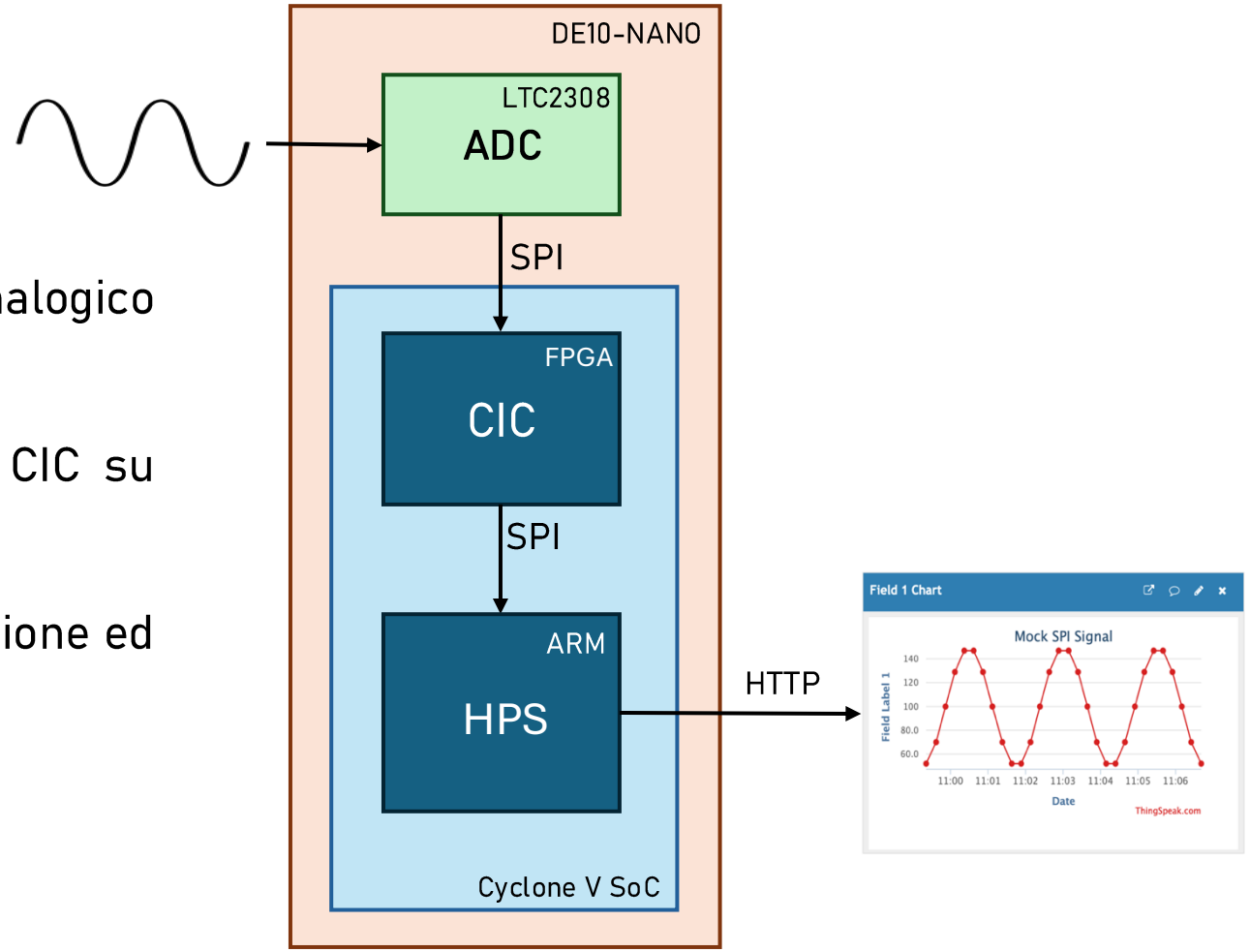
Implementazione di un filtro CIC su FPGA Cyclone V

Belotti Nicolò, Shqepa Frenki

Obiettivi del progetto

Si vuole utilizzare la scheda di sviluppo DE10-NANO per implementare un sistema di acquisizione remoto.

- Campionamento di un segnale analogico tramite ADC on-board
- Filtraggio passa basso hardware tramite filtro CIC su FPGA
- Trasmissione all'HPS ARM per ulteriore elaborazione ed invio alla piattaforma cloud ThingSpeak



Filtro CIC

Struttura e principio di funzionamento

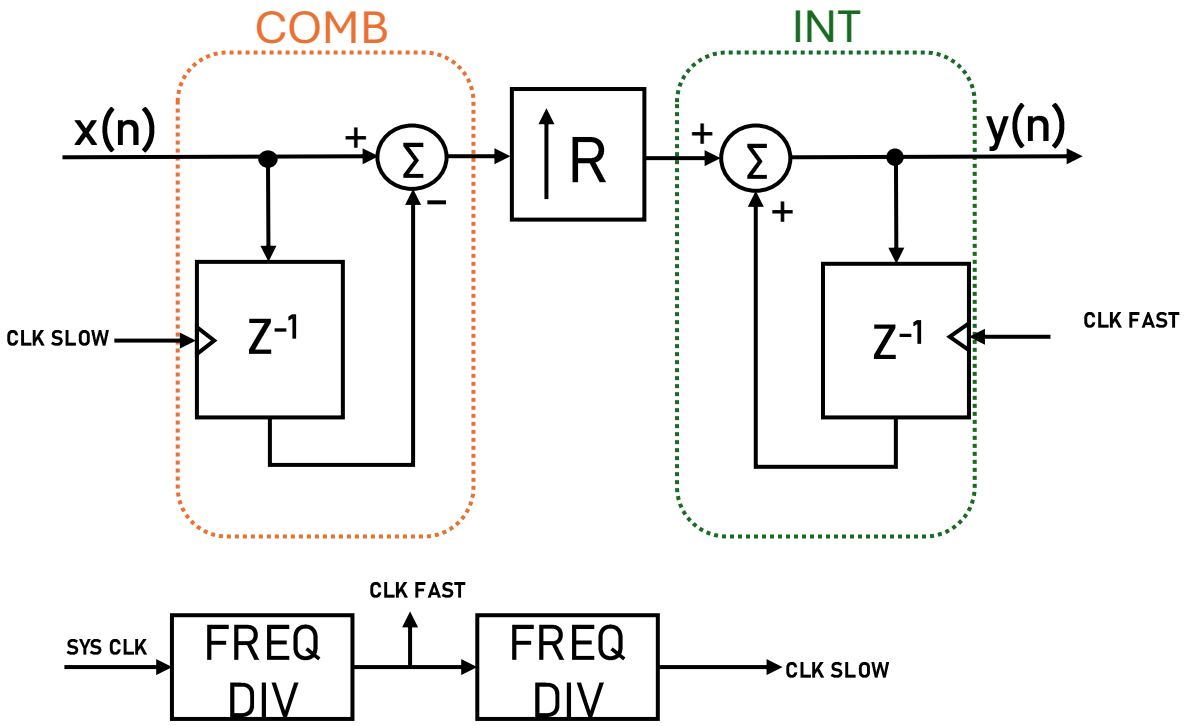
✓ Un filtro CIC implementa una risposta di tipo passa basso FIR senza l'utilizzo di moltiplicatori

Utilizza solo sommatori, registri e blocchi per il cambio di frequenza, permettendo di risparmiare risorse hardware.

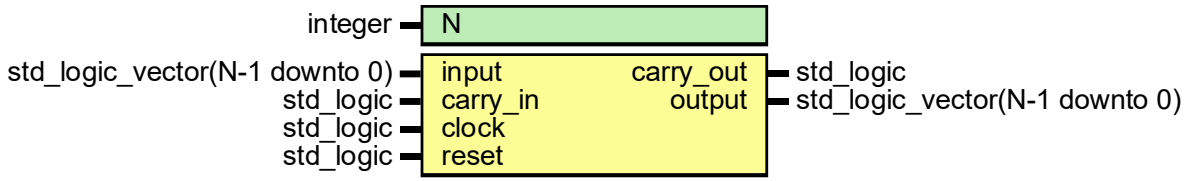
Due clock a frequenza $F_{fast}/F_{slow} = R$ permettono di interpolare il segnale d'ingresso

È composto da tre sezioni fondamentali:

- N stadi COMB con clock lento
- Stadio di inserimento di R-1 zeri,
- N stadi INT con clock veloce



Stadio integratore

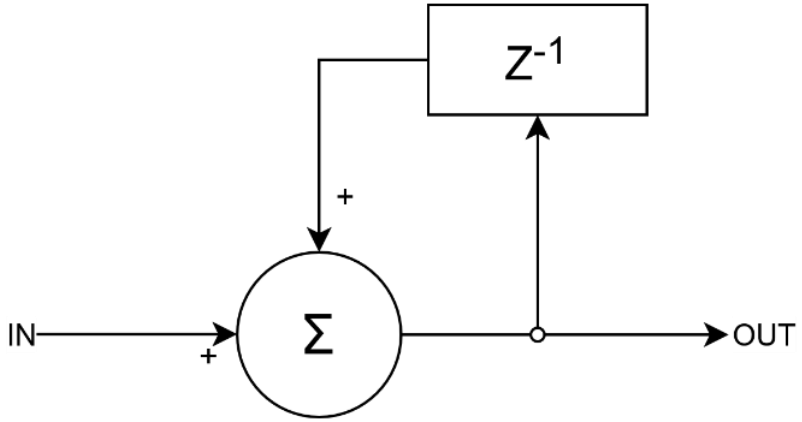


Effettua la somma cumulativa dei campioni in ingresso

Componente generic in N: la dimensione dei bus di ingresso e uscita.

Implementato tramite:

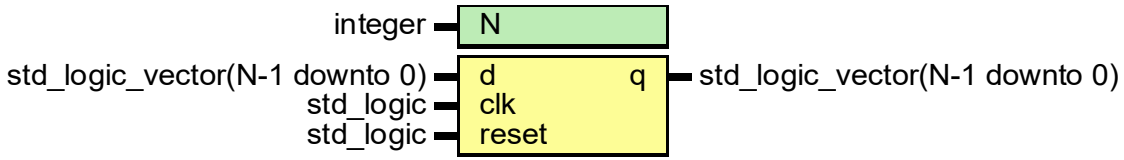
- Un Flip Flop di tipo D per la memorizzazione
- Un sommatore per eseguire la somma del campione attuale con i precedenti



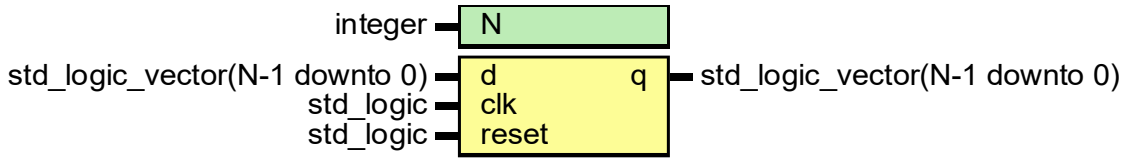
$$y[n] = y[n-1] + x[n]$$

Stadio integratore
sotto-blocchi elementari

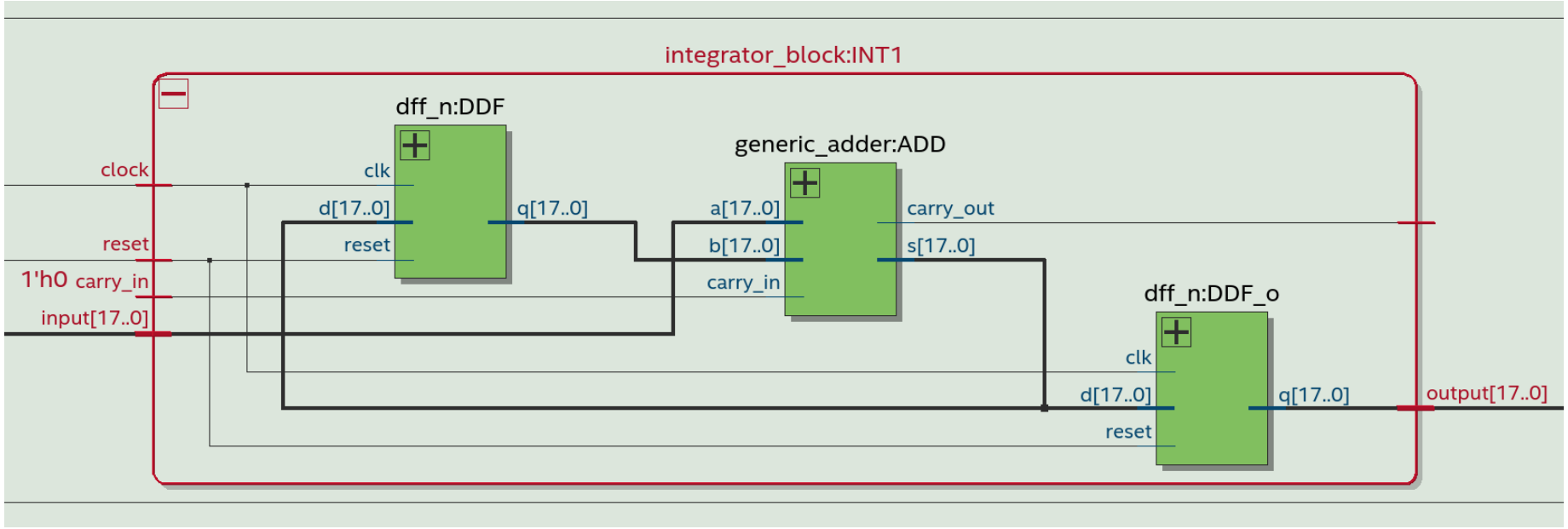
```
architecture behavioural of dff_n is

begin
  def:Process(clk)
      begin
        if(clk = '1' and clk'EVENT) then
          for i in 0 to N-1 loop
            q(i)<=d(i) and reset;
          end loop;
      end if;
  end process def;
end behavioural;
```



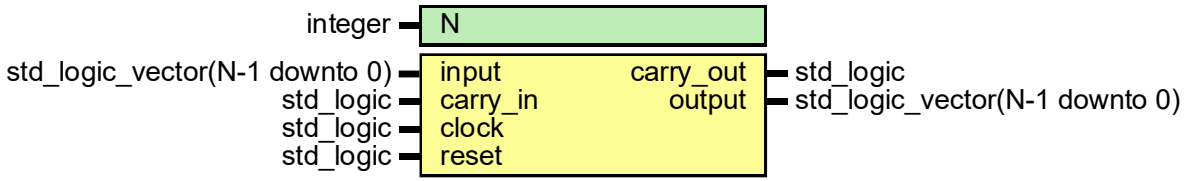

```
architecture behavioural of generic_adder is

begin
    --sum:process(clock)
    sum:process(a,b,carry_in)
      variable C:std_logic;
     begin
     C:=carry_in;
        for i in 0 to N-1 loop

     --if rising_edge(clock) then
            s(i)<= a(i) XOR b(i) XOR C;
         C:= (a(i) AND b(i)) OR (a(i) AND C) OR (b(i) AND C);
        end loop;
      carry_out <= C;
      --end if;
    end  process SUM;


end behavioural;
```



Stadio integratore
struttura sintetizzata



Stadio comb

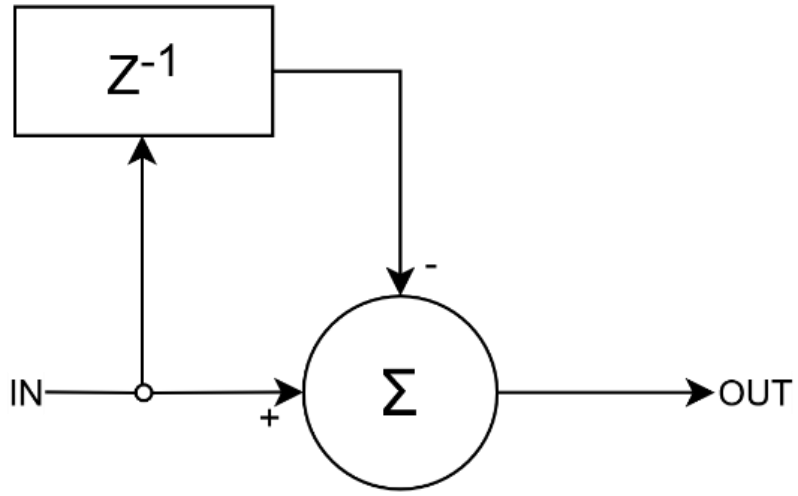


Effettua la differenza tra il campione attuale e quello M istanti prima

Componente generic in N: la dimensione dei bus di ingresso e uscita.

Implementato tramite:

- Un Flip Flop di tipo D per il ritardo
- Un blocco che esegue il complemento a due per il cambio del segno
- Un sommatore



$$y[n] = x[n] - x[n - M]$$

Stadio comb
sotto-blocco complemento a due

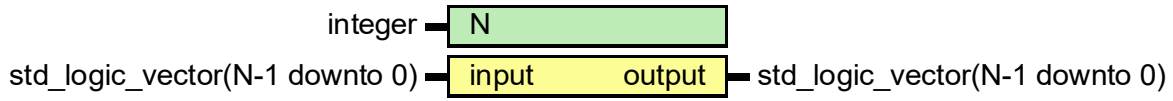
Inverte tutti i bit dell'ingresso e somma 1

$C_2(A) = NOT(A) + 1$

Descrizione VDHL comportamentale

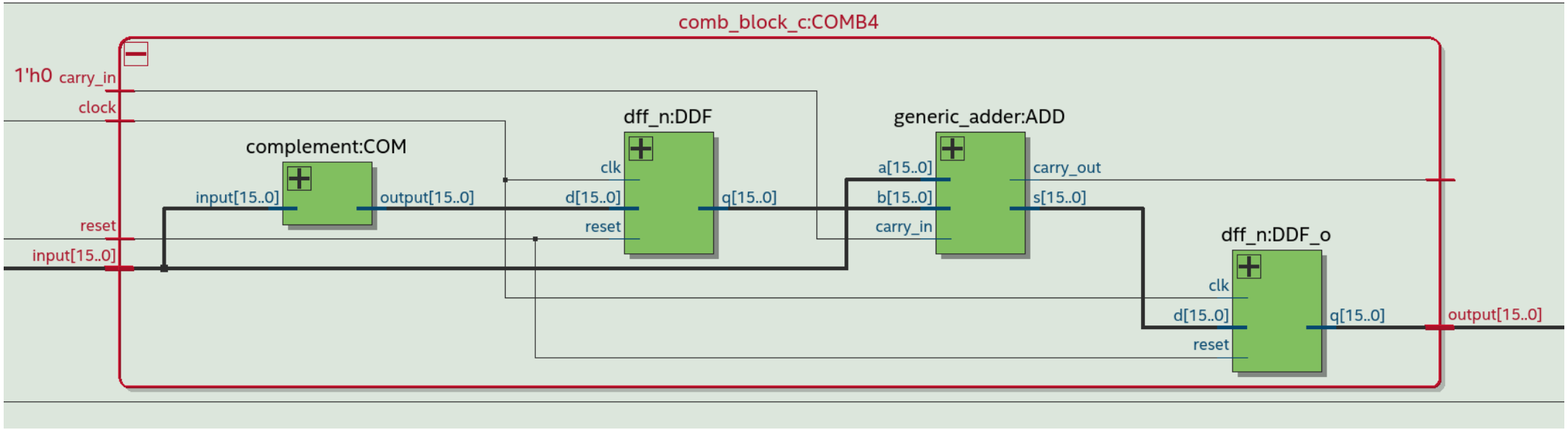
Generico nella dimensione N

```
architecture BEHAVIORAL of complement is
begin
  process(input)
    variable uno : std_logic_vector(N-1 downto 0);
    begin
  for i in 1 to N-1 loop
    uno(i) := '0';
  end loop;
  uno(0) := '1';

  output <= (NOT input) + uno;
end process;
end BEHAVIORAL;
```



Stadio comb

struttura sintetizzata



Stadi COMB e INT

Bit growth

A causa delle continue somme, avanzando nella catena cresce la dimensione necessaria a rappresentare il dato.

G_j il fattore di crescita del j-esimo stadio è dato da:

$$G_j = \begin{cases} 2^j & j = 1,2,\dots,N \\ \dfrac{2^{(2N-j)}(RM)^{j-N}}{R} & j = N+1,\dots,2N \end{cases}$$

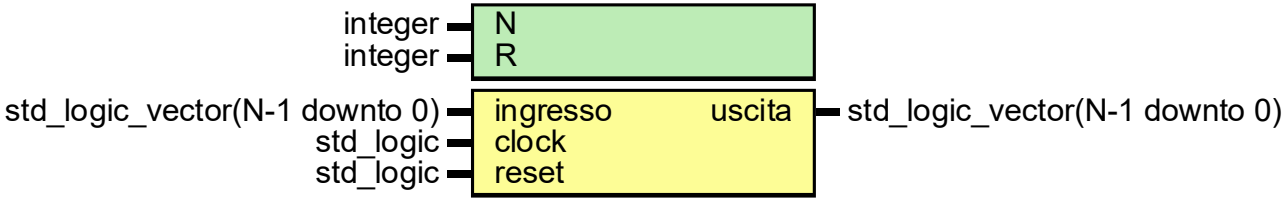
N l'ordine del filtro
R il fattore di interpolazione
M la profondità della memoria (delay)

La dimensione del j-esimo stadio:

$$W_j = \left[B_{in} + \log_2(G_j)\right]$$

B_{in} la dimensione del dato in ingresso

Inseritore di zeri



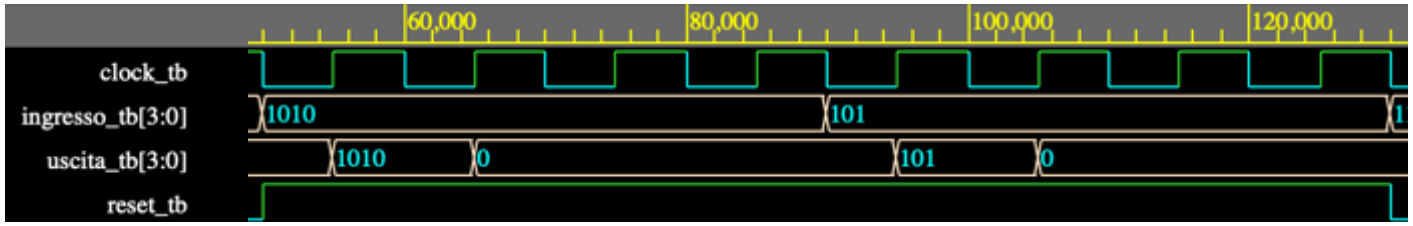
Implementato tramite la descrizione comportamentale di un contatore.

Componente generic in:

R il limite del contatore

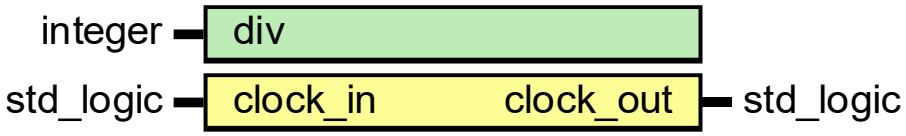
N la dimensione del bus.

L'uscita è pari al segnale d'ingresso quando il contatore è a 0, nulla altrimenti.



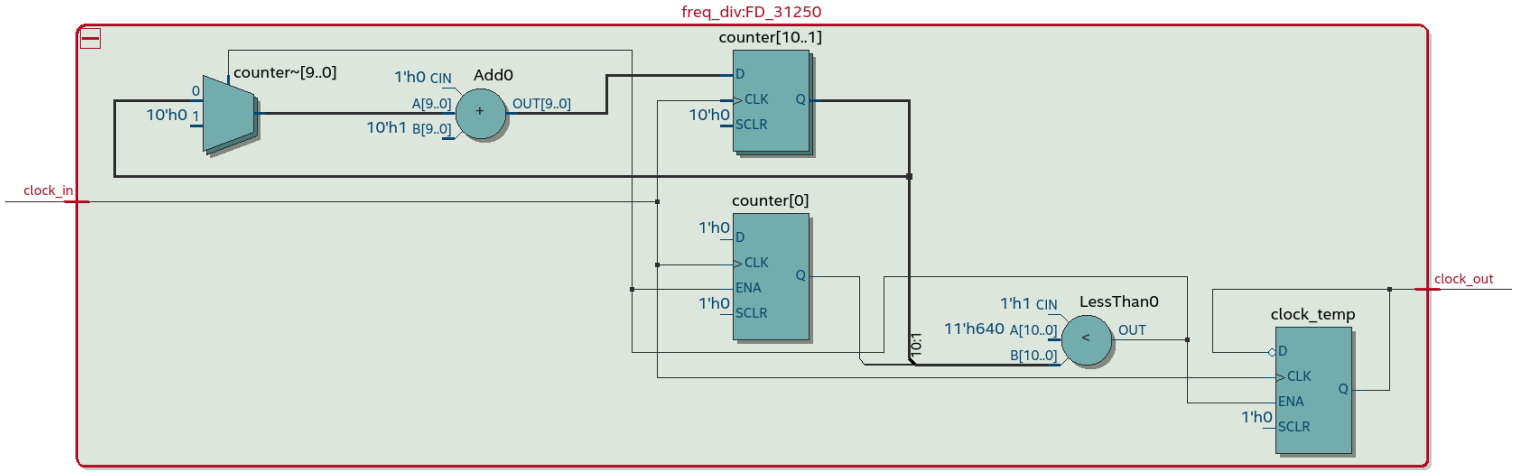

```
process(clock)
  variable counter : integer range 0 to R+1 :=0;
begin
if rising_edge(clock) then
  if (counter=0) and reset='1' then
  uscita <= ingresso;
    else
      uscita <=  (others => '0');
  end if;
  counter := counter+1 ;

      if (counter = 0) and reset = '1' then
        uscita <= ingresso;
      else
        uscita <= (others => '0');
      end if;
      counter := counter + 1;

      if (counter = R or reset = '0') then
        counter := 0;
      end if;
    end if;
end process;
```

Divisore di frequenza



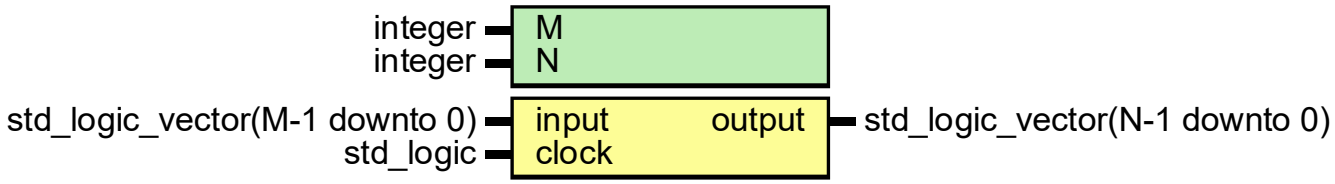
Implementato con un process sincrono con il fronte di salita di clock_in

Componente generic del fattore di divisione




```
architecture behavioral of freq_div is

signal clock_temp : std_logic :='1';
begin
clock_out <= clock_temp;

process(clock_in)
variable counter : integer range 0 to div+2 :=0;
begin
    if rising_edge(clock_in)  then
        if (counter>=div) then
            counter :=0;
            clock_temp <= not clock_temp;
        end if;
        counter := counter+2;
    end if;
end process;

end behavioral;
```

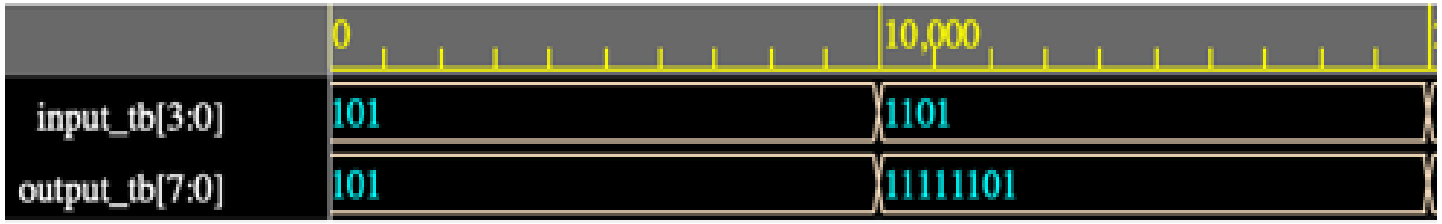
Shifter



Realizzato con un process sincrono con il fronte di salita del clock.

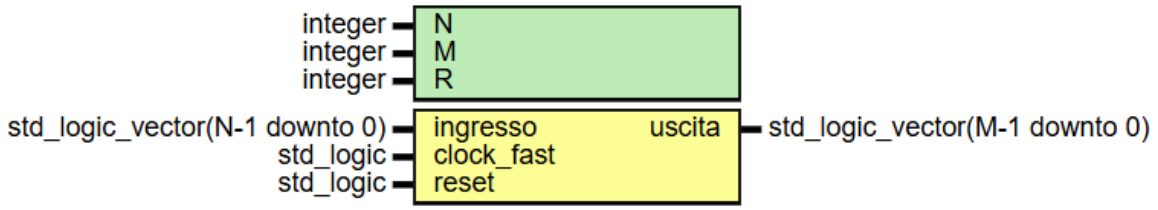
Accetta in ingresso un segnale ad M bit in complemento a 2 e ne aumenta la dimensione a N bit.

Se il numero in ingresso è negativo propaga il bit di segno.




```
process (clock)
begin
  if rising_edge(clock) then
    for i in 0 to M - 1 loop
      output(i) <= input(i);
    end loop;
    for i in M to N - 1 loop
      if input(M - 1) = '0' then
        output(i) <= '0';
      else
        if input(M - 1) = '1' then
          output(i) <= '1';
        else
          output(i) <= input(M - 1);
        end if;
      end if;
    end loop;
  end if;
end process;
```

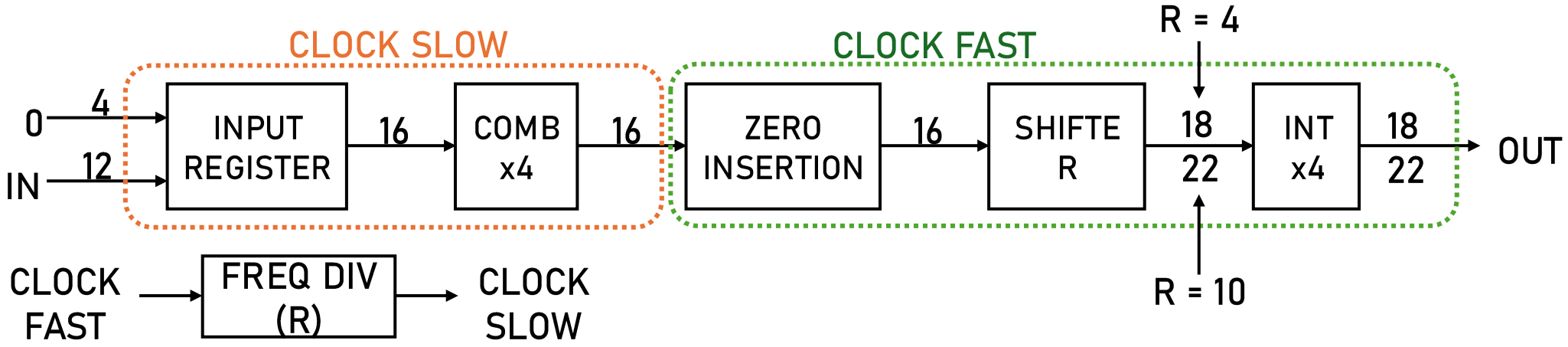
Filtro CIC
Parametri del progetto



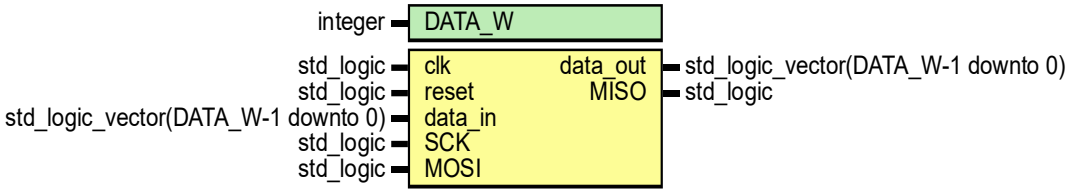
Generic di:
- N la dimensione dell'ingresso
- M la dimensione dell'uscita
- R il fattore di interpolazione

Filtro CIC del 4° ordine, provati R=4 e R=10.

- ADC 12 bit, interfacciamento tramite codice Verilog di terze parti
- Dimensione massima COMB: $W_4 = [12 + \log_2(2^4)] = 16$
- Dimensione massima INT: $W_8 = \left[12 + \log_2\left(\frac{(R)^4}{R}\right)\right] = 18\ se\ R = 4 \mid 22\ se\ R = 10$
- Sample rate SPI uscita 31,25 ksps → CLOCK FAST 31,25 kHz



Interfaccia SPI con HPS



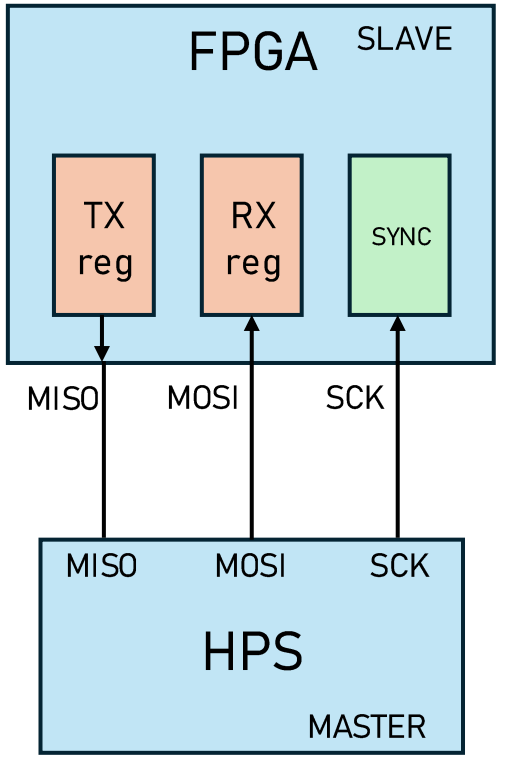
Basato su due registri a scorrimento (TX reg e RX reg) con clock esterno SCK generato dal master.

SCK (1 MHz) viene filtrato con sincronizzatore a 3 stadi per rilevazione dei fronti tramite logica combinatoria.

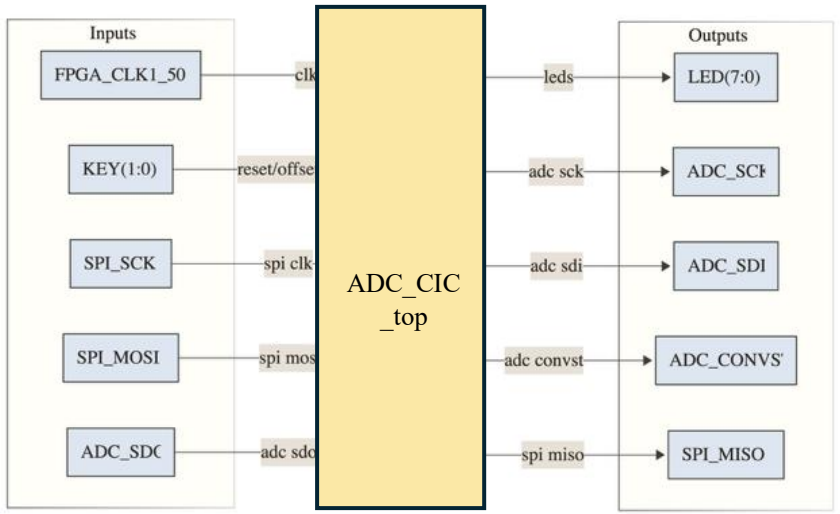
SPI mode 1:

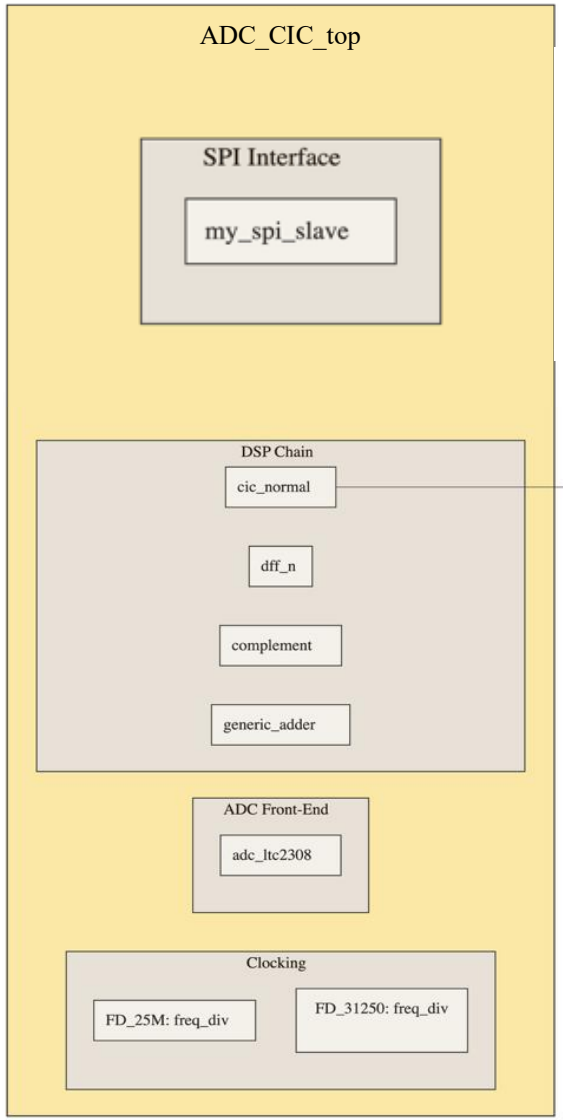
- TX reg carica il dato su MISO sul fronte di salita di SCK.
- RX reg campiona il dato da MOSI sul fronte di discesa di SCK

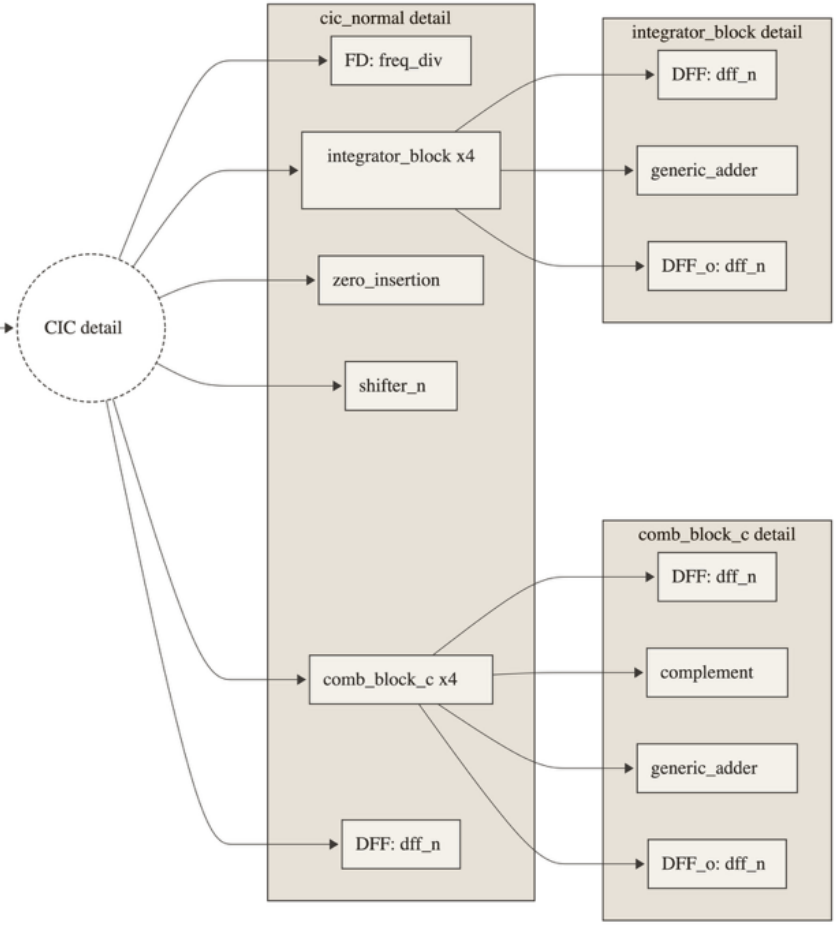
Registri TX e RX gestiti da un process sincrono con il clock di sistema



Gerarchia del progetto

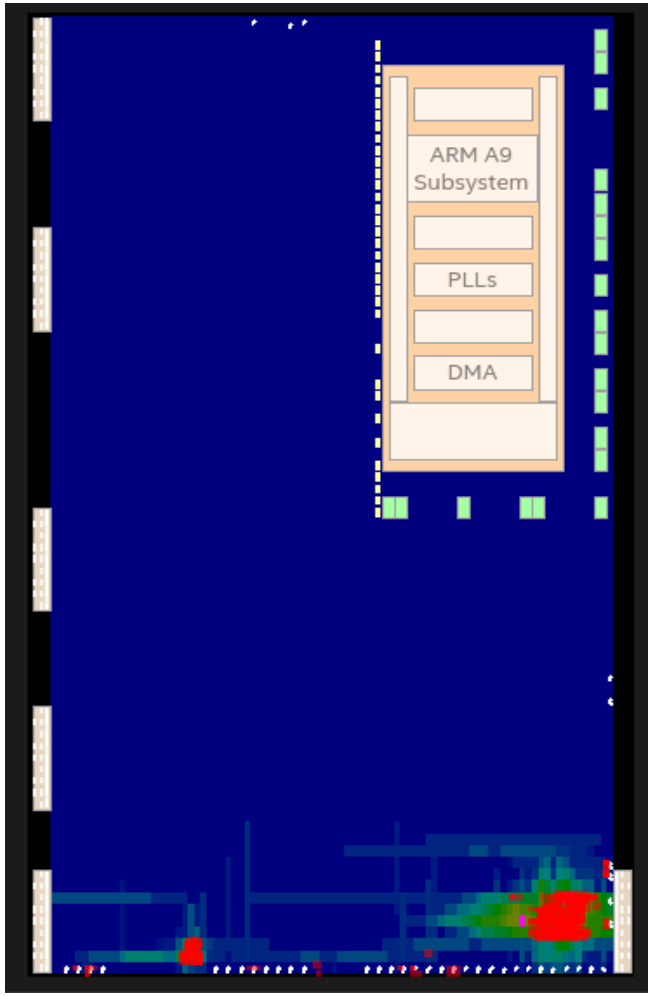




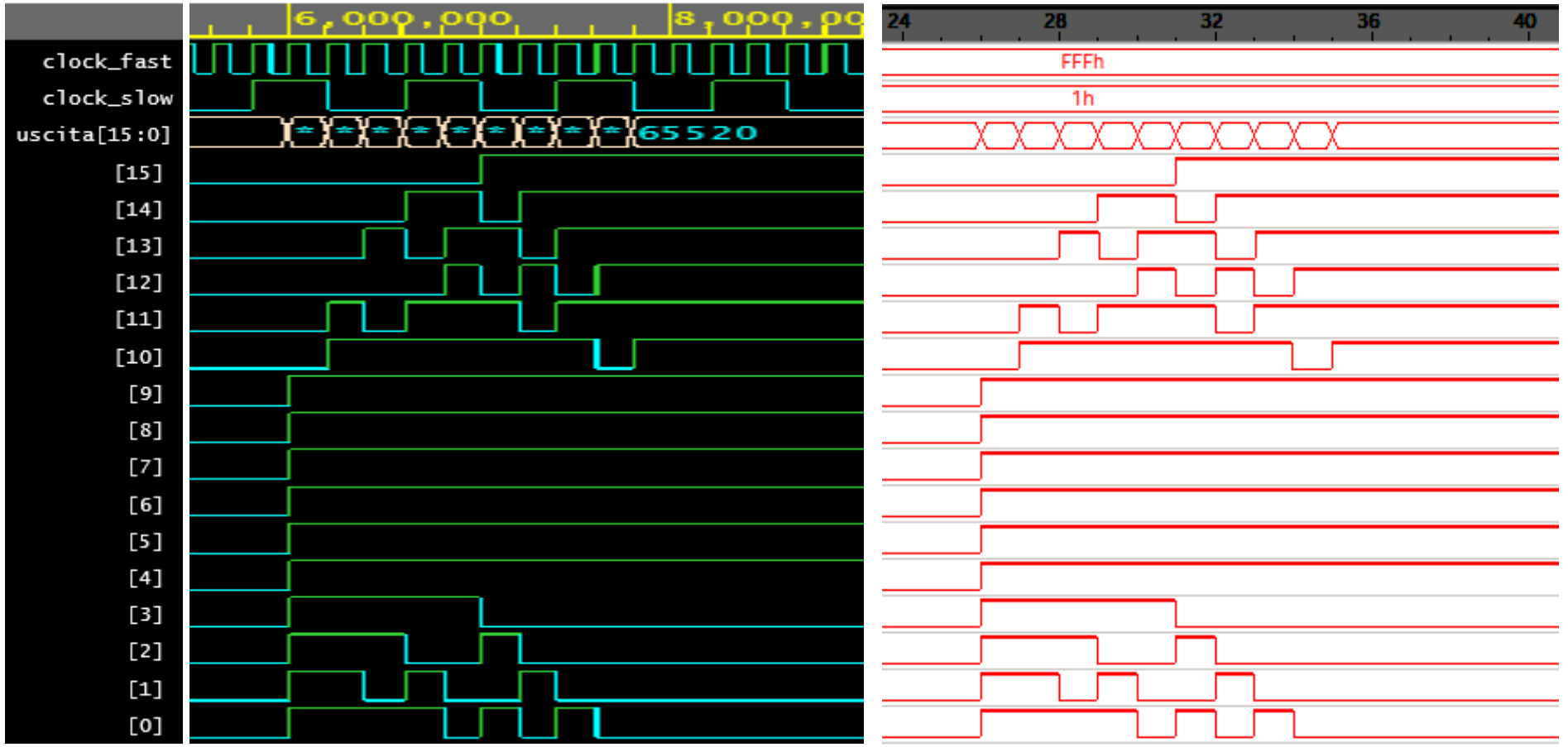


Risorse utilizzate su FPGA

Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	ADC_CIC_SPI
Top-level Entity Name	ADC_CIC_top
Family	Cyclone V
Device	5CSEBA6U23I7
Timing Models	Final
Logic utilization (in ALMs)	267 / 41,910 (< 1 %)
Total registers	383
Total pins	18 / 314 (6 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total RAM Blocks	0 / 553 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

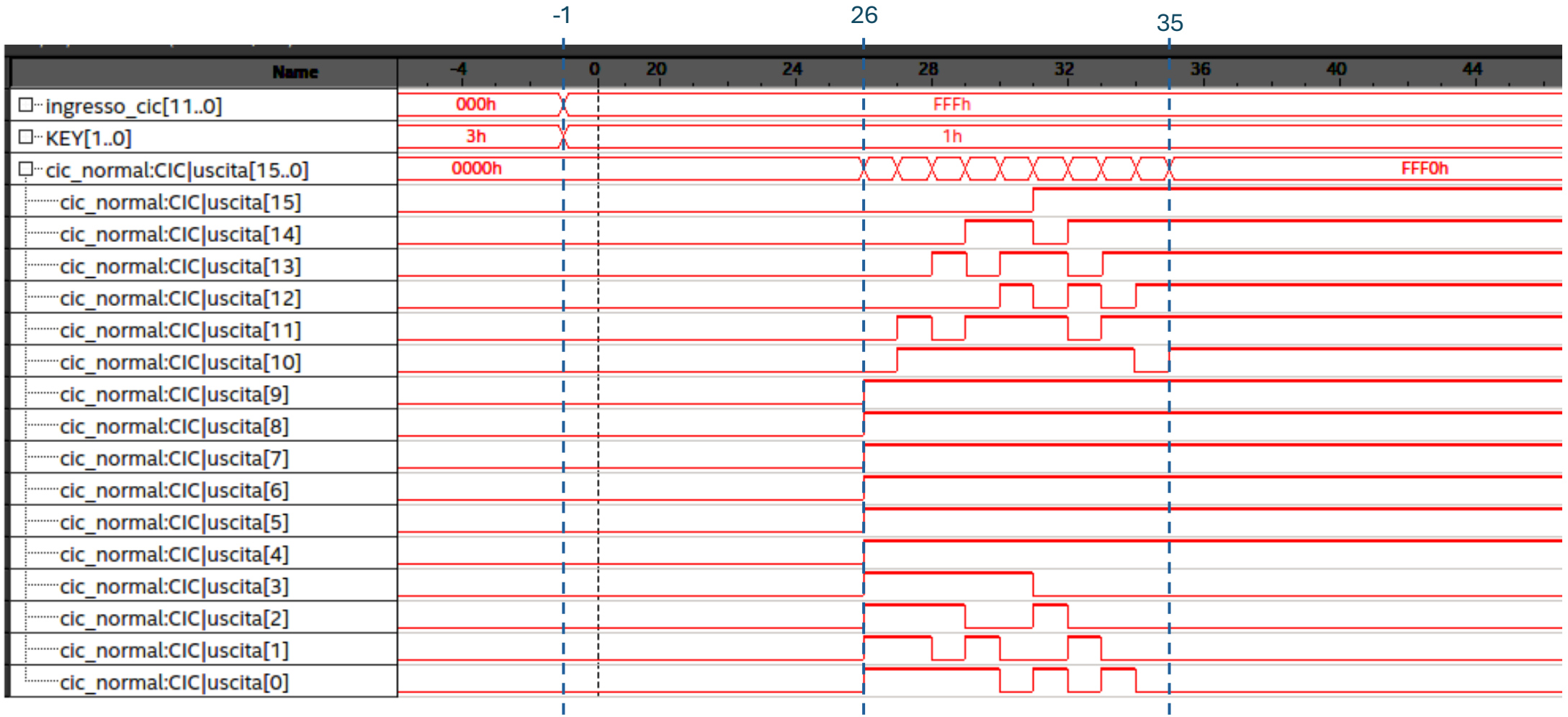


Risposta al gradino del filtro
simulazione vs SignalTap



Ritardo del filtro

$$delay = (26 - (-1)) * T_{CLK\ FAST} = 27 * \frac{1}{3125\ Hz} = 864\ us$$

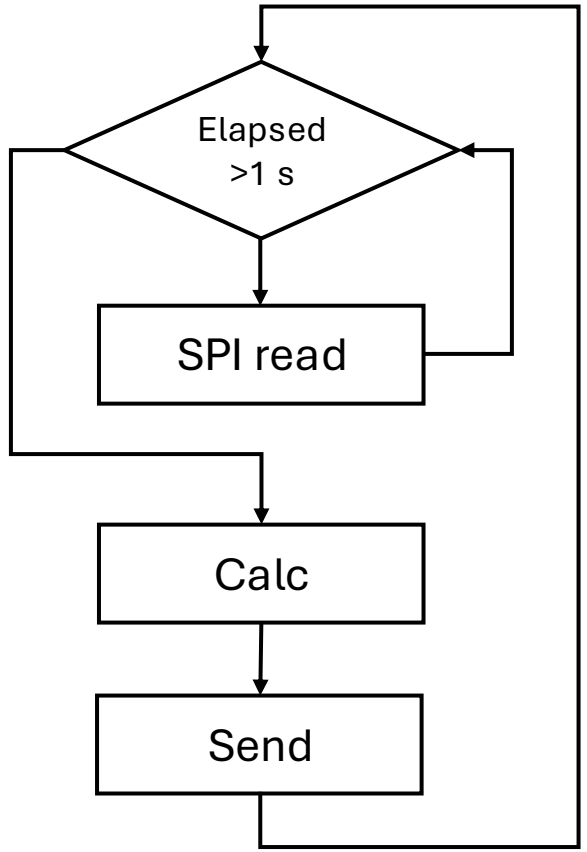


Elaborazione sull'HPS

Il programma acquisisce i dati dall'SPI e ne calcola il valore RMS su finestre di un secondo.

Il valore RMS viene inviato al PC tramite SSH o seriale.

Ogni 15 secondi viene aggiornato il canale ThingSpeak con l'ultimo valore calcolato.



Risposta in frequenza