

REALIZZAZIONE DI UN SISTEMA A RITARDO PROGRAMMABILE SU FPGA

In questa esperienza di laboratorio ci siamo occupati dell'implementazione di un circuito in grado di leggere i dati pervenuti con un ritardo programmabile dall'esterno ed un aggiornamento dei dati doppio rispetto al tempo di clock disponibile sulla scheda fpga in dotazione.

Problema fisico

L'idea del nostro progetto è nata dalla necessità di poter ritardare a proprio piacimento i segnali provenienti dall'apparato di lettura di un RPC.

Un RPC è un rivelatore di particelle studiato per aree attive molto grandi. Si basa sulla variazione di resistività generata in un gas quando in esso passano delle particelle cariche. Possiamo immaginare il nostro rivelatore come una sorta di condensatore a facce piane e parallele costituite da un piano di bachelite ricoperto da un sottile strato di grafite. Quando una particella attraversa il gas posto fra i due piani del "condensatore" questo viene ionizzato permettendo il passaggio di una scarica: questa verrà captata da alcune zone metalliche (strip line) poste sulla grafite che porteranno il segnale al sistema di lettura. Per non avere false rilevazioni dovute a rumore poniamo due scintillatori attorno all'RPC in modo tale che solo al reale passaggio di una particella avremo tre segnali: gli scintillatori forniranno infatti un gate di trigger al cui interno dovrà trovarsi l'hit generato dall'RPC.

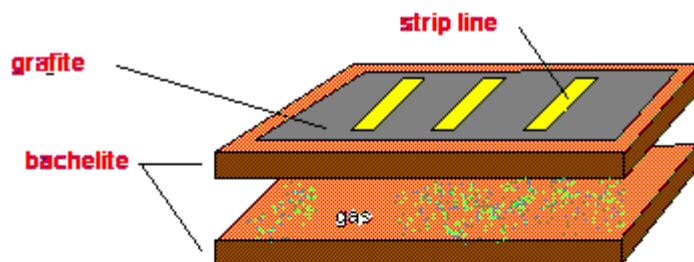


Figura 1:
Schema di un RPC

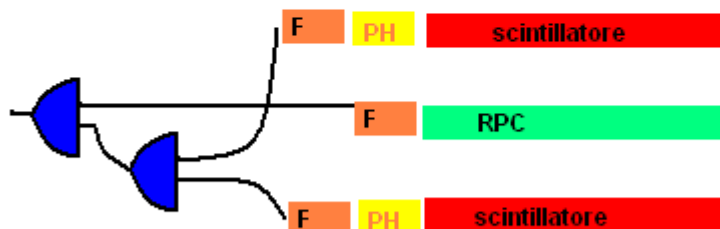


Figura 2:
Apparato di misura
connesso all'RPC
Ph: fotomoltiplicatore.
F: serie di amplificatore
operazionale,
discriminatore e formatore.

Un segnale in uscita dalla strip line sarà variabile in intensità e durata: è per questo che deve essere fatto passare all'interno di un amplificatore, di un discriminatore e di un formatore. Il segnale può essere ora trattato mediante un computer e applicazioni software.

Andiamo ora a spiegare quali sono i problemi connessi alla propagazione dei segnali dal rivelatore al computer.

Il gate fornito dagli scintillatori sarà ritardato rispetto al reale passaggio di una particella essendo il tempo di transito nello scintillatore di 3-4 ns, nei fotomoltiplicatori da 1 a 10 ns e nel circuito di trigger di 20-30 ns.

Di contro l'RPC risponde in meno di 1 ns: approssimativamente avremo quindi :

$$\Delta t_{RPC} = 2ns$$

$$\Delta t_{trigger} = 50ns$$

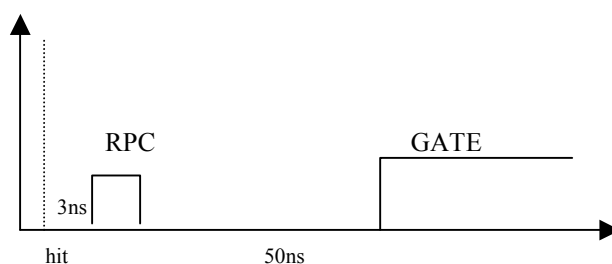


Figura 3: risposta reale

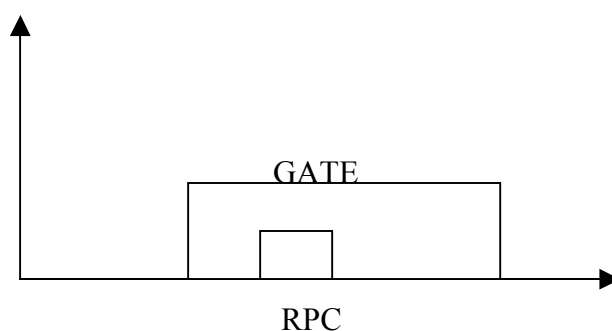


Figura 4: risposta ideale

Di qui la necessità di un circuito logico che permetta di andare a leggere i segnali con un ritardo programmabile.

Specifiche richieste

L'RPC fornisce 64 canali d'ingresso mentre uno sarà necessario per l'input di trigger.

L'obiettivo è di ottenere un ritardo programmabile da 10 a 100 ns.

L'indeterminazione massima sul tempo di arrivo di un hit deve essere di 5 ns.

Realizzazione pratica

Abbiamo risolto il problema in due modi differenti: il primo mediante l'uso di uno shift register, il secondo mediante l'uso di celle ram.

1) Realizzazione mediante Shift Register

Il modello fondamentale su cui ci siamo basati è composto di 2 oggetti: un registro a scorrimento e un multiplexer che permette di andare a scegliere dall'esterno che punto della catena avere in uscita.

Un registro a scorrimento è costituito da N flip-flop di tipo D posti in maniera tale che l'uscita dell'i-esimo flip-flop coincida con l'ingresso del successivo. Ad ogni segnale di clock il dato presente all'ingresso viene memorizzato sul primo flip-flop ed ogni altro elemento della catena verrà aggiornato assumendo il valore dell'elemento precedente.

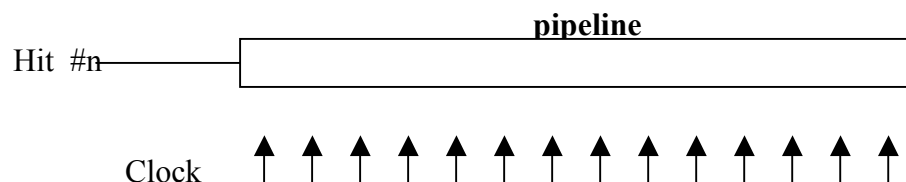


Figura 5: schema di un registro a scorrimento

Segue il semplice listato vhdl con cui è stato possibile realizzare questo processo:

```
if rising_edge(clk) then
  matrice(1) <= hit;
  for i in 1 to 19 loop
    matrice(i+1) <= matrice(i);
  end loop;
end if;
```

Come si vede il processo comincia solo sul fronte di salita del clock (rising_edge). In questa situazione il primo elemento di ogni pipe line viene aggiornato (ovvero ogni riga dell'array bidimensionale "matrice") al valore in ingresso mentre mediante il ciclo "for" tutti gli altri elementi vengono aggiornati al precedente.

Essendo 64 le strip line di un RPC avremmo dovuto realizzare 64 shift register. In realtà per ottenere un'indeterminazione sull'acquisizione dei dati pari alla metà del tempo di clock abbiamo utilizzato due registri a scorrimento per ogni segnale di ingresso, uno aggiornato ad ogni fronte di salita del clock, l'altro su ogni fronte di discesa, connettendo poi le uscite delle due mediante una porta logica or.

Questo procedimento deve essere realizzato in due "process" differenti e non è possibile far aggiornare lo stesso registro per via di un problema di driver (avremmo avuto la medesima uscita per due differenti segnali).

Visto il grande numero di flip-flop necessari abbiamo utilizzato l'fpga XC4085X della Xilinx che è una matrice di 56 x 56 clb (7168 flip-flop).

Durante la simulazione abbiamo verificato un ritardo dal tempo d'ingresso di un hit alla sua lettura dal primo elemento del registro (ritardo quindi che influenza ogni lettura) di 57ns.

Segue il listato vhd1 al termine della relazione.

2) Realizzazione mediante celle ram

In questo caso i componenti fondamentali utilizzati sono stati tre: una memoria leggi-scrivi dual port da 16 bit, un contatore e un sottrattore.

La ram, o memoria ad accesso casuale (random access memory) è una schiera di celle da un bit che memorizzano l'informazione in forma binaria. La cella di memoria monolitica di base è il flip-flop. L'informazione può essere scritta o letta in ciascun elemento di memoria in modo casuale.

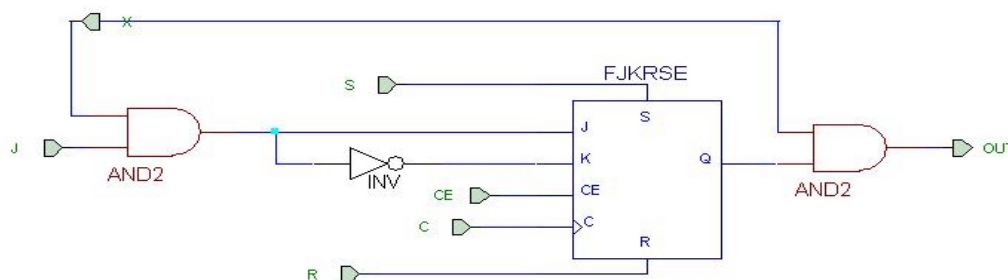


Figura 6: Memoria a scrittura e lettura ad un bit.

Ad ogni segnale di clock il contatore avanza di un'unità modificando contemporaneamente l'indirizzo di scrittura in modo tale da poter leggere tutti i dati ricevuti fino a 15 segnali di clock precedenti.

Per andare a leggere un dato con un ritardo fisso dall'attuale segnale d'ingresso è necessario che anche l'indirizzo di lettura vari col tempo. Questo può essere facilmente ottenuto con l'uso di un sottrattore valendo la relazione:

$$\text{Indirizzo} = \text{contatore} - \text{registro}$$

dove "indirizzo" rappresenta l'effettiva codifica in binario del punto in cui si vuole andare a leggere sulla ram, "contatore" fornisce l'indirizzo di scrittura e infine "registro" esprime il ritardo che deve avere il dato che ci interessa rispetto all'attuale segnale di clock.

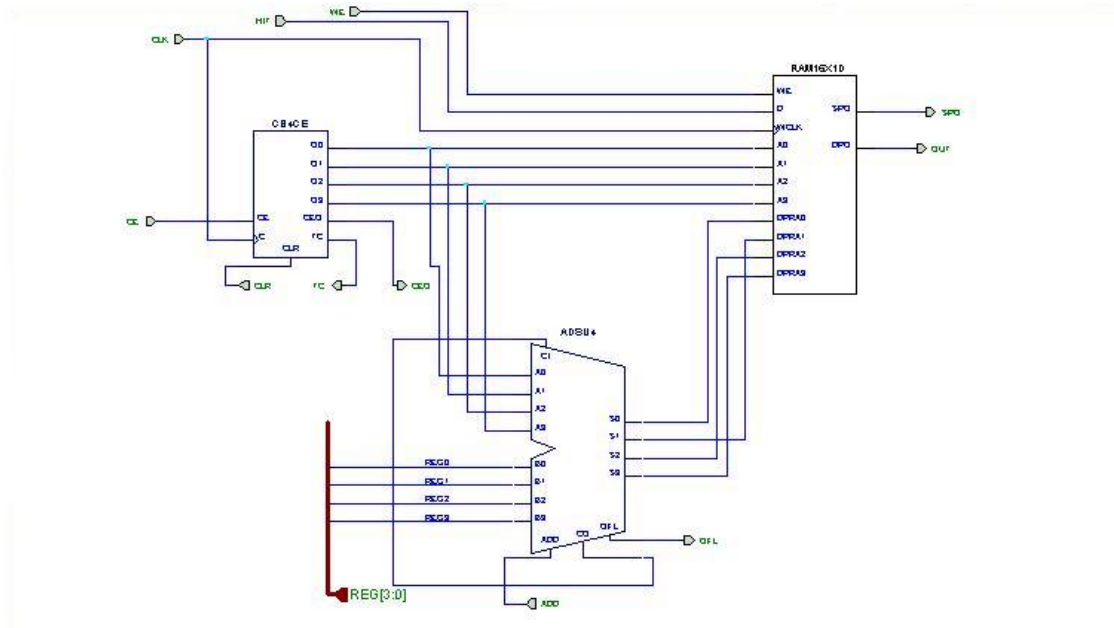


Figura 7: schema di base del circuito realizzato mediante ram.

Per ridurre l'utilizzo di Clb all'interno dell'fpga abbiamo utilizzato i componenti predefiniti dalla Xilinx anche in vhdl: per fare ciò è necessario creare una nuova libreria in cui salvare gli elementi che verranno richiamati nella descrizione strutturale del circuito. Mediante l'utility "logicBlox" o "core generator" è possibile accedere ad un menù da cui selezionare il componente necessario specificandone inoltre alcune caratteristiche.

La dichiarazione strutturale degli oggetti così creati avviene poi mediante il comando "port map".

Anche in questo caso per raddoppiare il tempo di clock abbiamo implementato due serie di ram con due contatori differenti, uno aggiornato su ogni fronte di salita, l'altro su ogni fronte di discesa.

Abbiamo utilizzato di nuovo l'fpga XC4052XL della Xilinx.

Durante la simulazione abbiamo verificato un ritardo dal tempo d'ingresso di un hit alla sua lettura dal primo elemento del registro (ritardo quindi che influenza ogni lettura) di 72ns.

Segue il listato in vhdl.

Realizzazione mediante uno shift-register


```

matrice(4) or matrice1(4) when (reg="00011") else
matrice(5) or matrice1(5) when (reg="00100") else
matrice(6) or matrice1(6) when (reg="00101") else
matrice(7) or matrice1(7) when (reg="00110") else
matrice(8) or matrice1(8) when (reg="00111") else
matrice(9) or matrice1(9) when (reg="01000") else
matrice(10) or matrice1(10) when (reg="01001") else
matrice(11) or matrice1(11) when (reg="01010") else
matrice(12) or matrice1(12) when (reg="01011") else
matrice(13) or matrice1(13) when (reg="01100") else
matrice(14) or matrice1(14) when (reg="01101") else
matrice(15) or matrice1(15) when (reg="01110") else
matrice(16) or matrice1(16) when (reg="01111") else
matrice(17) or matrice1(17) when (reg="10000") else
matrice(18) or matrice1(18) when (reg="10001") else
matrice(19) or matrice1(19) when (reg="10010") else
matrice(20) or matrice1(20) ;

end Pipeline_arch;

```

Realizzazione mediante celle ram

```

library IEEE;
use IEEE.std_logic_1164.all;

```

entity ram is

```

    port (
        -- a1: in STD_LOGIC_VECTOR (3 downto 0);
        hit: in STD_LOGIC_vector(63 downto 0);
        reg: in STD_LOGIC_VECTOR (3 downto 0);
        we: in STD_LOGIC;
        ce,cin: in STD_LOGIC;
        spo1,spo2: out STD_LOGIC_vector(63 downto 0);
        oout: out STD_LOGIC_vector(63 downto 0)
    );
end ram;

```

architecture struct of ram is

```

signal a1,dport,a2,dport2: STD_LOGIC_VECTOR (3 downto 0);
signal dpo1,dpo2: STD_LOGIC_vector(63 downto 0);
signal nce: std_logic;
component dpram64 port(
    WR_EN: in STD_LOGIC;
    WR_CLK:in STD_LOGIC;
    DI: in STD_LOGIC_vector(63 downto 0);
    DPRA: in STD_LOGIC_vector(3 downto 0);
    A: in STD_LOGIC_VECTOR (3 downto 0);
    SPO: out STD_LOGIC_vector(63 downto 0);
    DPO: out STD_LOGIC_vector(63 downto 0)
);
end component;

```

```

component subtr port(
c_in: in STD_LOGIC;
A: in STD_LOGIC_VECTOR (3 downto 0);
b: in STD_LOGIC_vector(3 downto 0);
sum: out STD_LOGIC_vector(3 downto 0)
);
end component;
component cont port(
clock:in std_logic;
q_out:out std_logic_vector(3 downto 0)
);
end component;
begin
nce <= not ce;
u0: cont port map (clock => ce, q_out => a1);
u1 : subtr port map (c_in => cin, A => a1,b => reg,Sum => dport);
u2 : dpram64 port map (WR_EN => we,WR_CLK =>ce , DI => hit, DPRA =>
dport,A => a1,SPO => spo1,DPO => dpo1);

-----

u3: cont port map (clock => nce, q_out => a2);
u4 : subtr port map (c_in => cin, A => a2,b => reg,Sum => dport2);
u5 : dpram64 port map (WR_EN => we,WR_CLK =>nce , DI => hit, DPRA =>
dport2,A => a2,SPO => spo2,DPO => dpo2);

-----

oout<=dpo1 or dpo2;
end struct;

```