MAPD mod. A - Progetto Finale

Agostini Federico, Bottaro Federico, Pompeo Gianmarco

18 Ottobre 2019

1 Rappresentazione grafica del progetto

A scopo introduttivo, riportiamo una schematica di quello che risulterà essere il circuito costruito per il progetto finale.

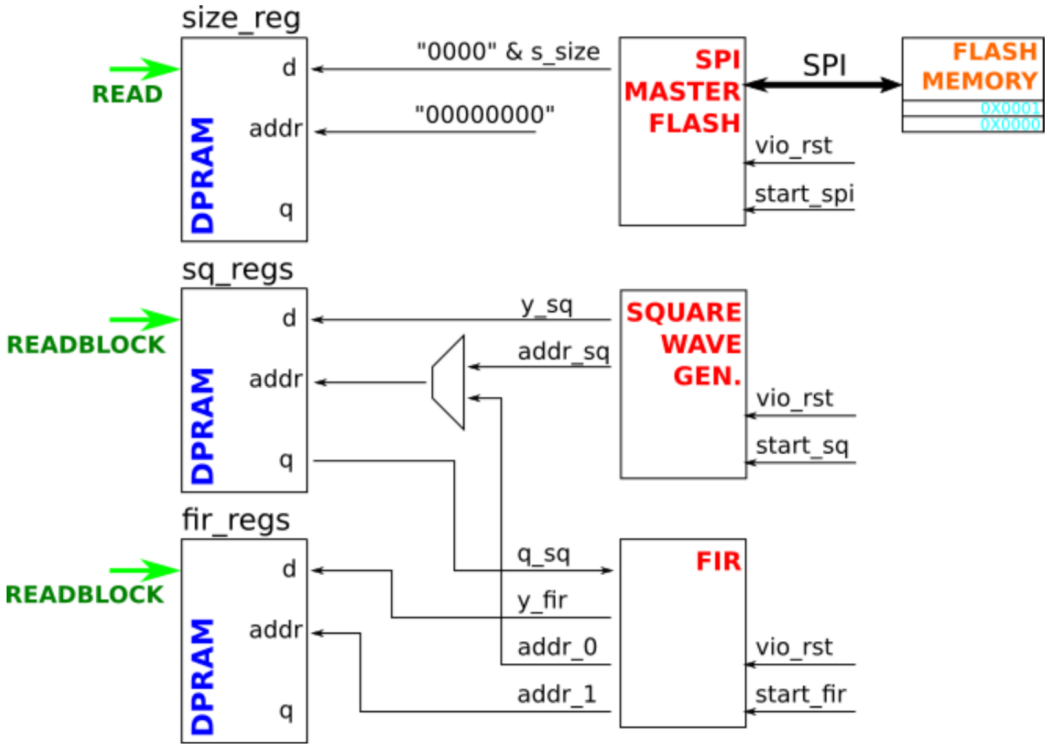


Figura 1: *Schema dei blocchi del circuito implementato per il progetto finale.*

2 VIO policy

Il VIO (Virtual Input Output) è stato implementato utilizzando tre input virtuali. In particolare, tali input si riferiscono al segnale di reset, al segnale di start connesso al generatore di onda quadra e al segnale di start connesso al filtro FIR.
Al fine di una corretta esecuzione del software è necessario avviare dapprima il generatore di onda quadra e, dopo qualche istante, procedere con lo start del FIR e lanciare allora gli script in Python per ottenere i grafici desiderati.

3 Codice VHDL

Come richiesto, vengono qui riprodotti gli *snippets* di codice in VHDL contenenti le porzioni più salienti del progetto in esame.

3.1 FSM in `spi_master_flash.vhd`

Nella FSM (Finite State Machine), solo lo stato `s_buildword` è stato modificato; si riporta di seguito il codice relativo.

```
when s_buildword =>
    if cnt_o < 3 then
       we_out <= '1';
       word <= s_word; -- NEW
       cnt_o := cnt_o + 1;
       if bcnt = 0 then
          s_word(RXBITS-1 downto 0 ) <= s_rxd; -- NEW
          bcnt  := N_BYTES - 1;
          state <= s_stop;
          ready_s <= '1';
       else
          s_word(((bcnt+1)*RXBITS-1) downto (bcnt*RXBITS )) <= s_rxd; -- NEW
          bcnt  := bcnt - 1;
          state <= s_getbyte;
          ready_s <= '0';
       end if;
    else
       we_out <= '0';
       cnt_o := 0;
    end if;
```

3.2 Generatore di onda quadra: `low-state`

Come ultima sezione rilevante, riportiamo lo stato `low-state` della FSM utilizzata per implementare il generatore di onda quadra; esso corrisponde, come è evidente, allo stato in cui l'onda quadra assume il suo valore minimo.

```
when s_low =>
    if sample_cnt = SAMPLE_N-1 then
     state_fsm <= s_idle;
     we_out <= '0';
    else
     if period_cnt < PERIOD then
       we_out <= '1';
       state_fsm <= s_low;
     else
       state_fsm <= s_high;
       period_cnt := 0;
     end if;
    end if;
    address_out <= std_logic_vector(to_unsigned(sample_cnt, address_out'length));
    y <= std_logic_vector(to_signed(-1024, y'length));
    period_cnt := period_cnt + 1;
    sample_cnt := sample_cnt + 1;
```

3.3 FSM nell'*architecture* del FIR

All'interno dell'architettura del filtro FIR è contenuta una FSM; essa viene riportata di seguito.

```
case state_fsm is

      when s_idle =>
        if en_in = '1' and en_p = '0' then
          state_fsm <= s_read;
          selector <= '1';     -- take 'ownership' of square register
          we_s <= '0';         -- don't allow writing: still reading
        else
          state_fsm <= s_idle;
          selector <= '0';
        end if;
        cnt := 0;

      when s_read =>
        if cnt < N_sample then
          addr_out0 <= std_logic_vector(to_unsigned(cnt, addr_out0'length));
          we_s <= '0';
          selector <= '1'; --we say the mux to look at our address
          state_fsm <= s_filter;
        else
          state_fsm <= s_idle; --finished
        end if;

      when s_filter =>

        o0 <= fir_in;

        m0 <= signed(o0) * C0;
        m1 <= signed(o1) * C1;
        m2 <= signed(o2) * C2;
        m3 <= signed(o3) * C3;
        m4 <= signed(o4) * C4;

        s0 <= m0;
        s1 <= s0 + m1;
        s2 <= s1 + m2;
        s3 <= s2 + m3;
        s4 <= s3 + m4;

        x_sum <= std_logic_vector(resize(SHIFT_RIGHT(s4, Q), N));

        state_fsm <= s_write;

      when s_write =>
        we_s <= '1';
        addr_out1 <= std_logic_vector(to_unsigned(cnt,addr_out1'length));
        cnt := cnt + 1;
        state_fsm <= s_read;

      end case;
```

3.4 Implementazione del MUX

Di seguito riportiamo il codice relativo all'implementazione del MUX (multiplexer).
I tre segnali di input corrispondono ad un indirizzo di memoria proveniente dal generatore di onda quadra e ad un secondo indirizzo generato dal filtro FIR. A questi si aggiunge un terzo input, costituito da un selettore che permette di decidere quale dei due precedenti segnali trasmettere.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```
4  entity mux21 is
5      generic (
6      N_BITS : integer := 10
7      );
8      Port ( a_in : in std_logic_vector(N_BITS-1 downto 0);
9             b_in : in std_logic_vector(N_BITS-1 downto 0);
10            sel_in : in std_logic;
11            y_out : out std_logic_vector(N_BITS-1 downto 0));
12 end mux21;
13
14 architecture rtl of mux21 is
15
16 begin
17
18 selp : process(a_in, b_in, sel_in)
19     begin
20     if sel_in = '0' then
21         y_out <= a_in;
22     else
23         y_out <= b_in;
24     end if;
25 end process;
26
27
28 end rtl;
```

4 Filtro FIR

Sono riportate di seguito le figure ottenute dallo script Python fornitoci. Esse raffigurano nel piano Ampiezza-Frequenza il filtro FIR (Finite Impulse Response) che è stato implementato.

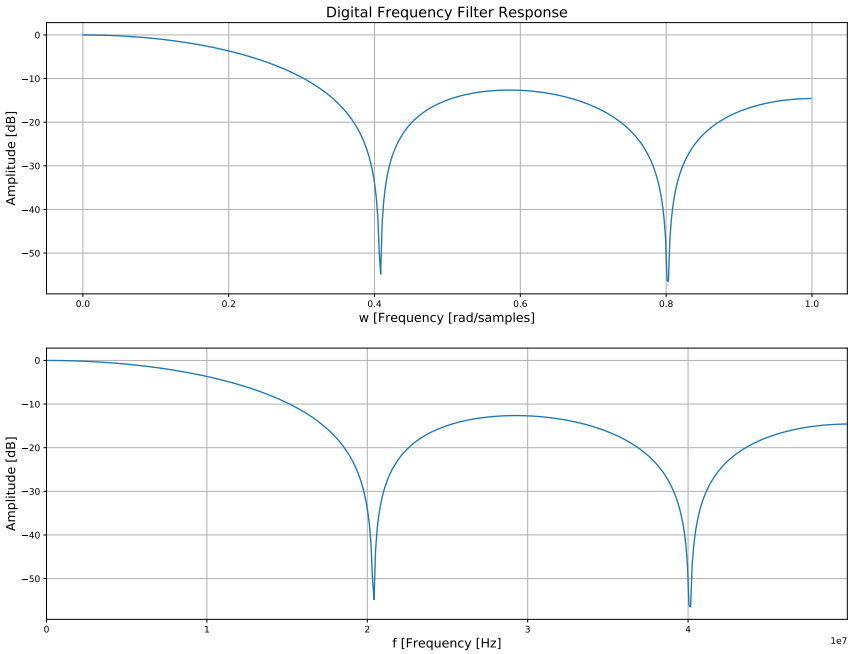


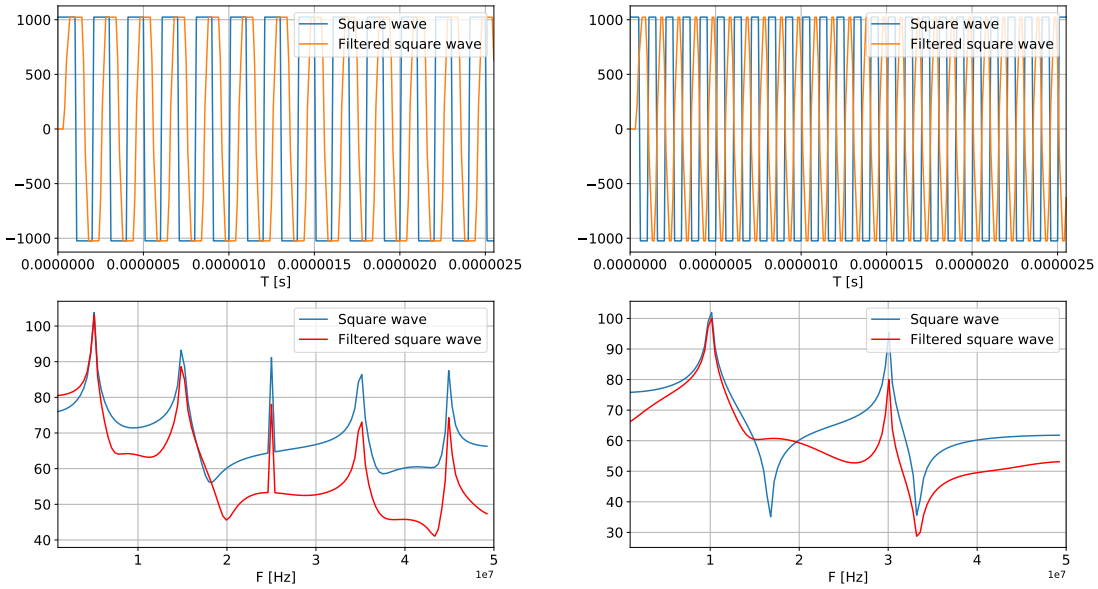
Figura 2: *Immagine prodotta dallo script* `coeff.py` *relativa al filtro FIR.*

5 Segnale generato e filtrato

Di seguito vengono riportate le figure prodotte dallo script `final_project.py` in Python al variare dei parametri `PERIOD` and `DUTY_CIC`.

(A) *PERIOD = 20, DUTY_CYC = 50*

(B) *PERIOD = 10, DUTY_CYC = 50*



(C) *PERIOD = 20, DUTY_CYC = 30*

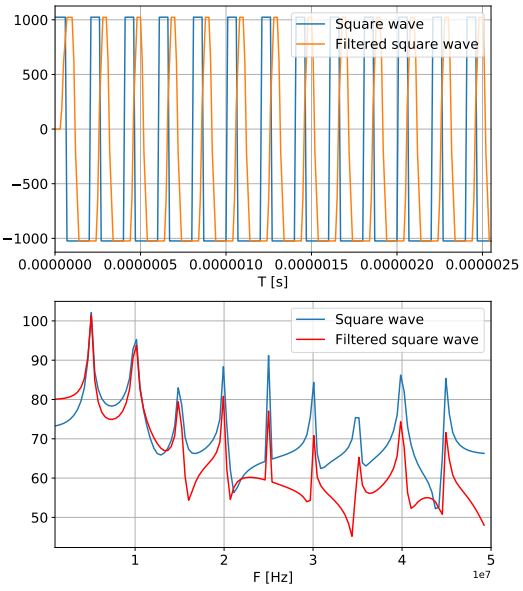


Figura 3: *Immagini prodotte dallo script* `final_project.py` *per differenti valori di periodo e duty cycle.*