1 – α (1 – ALPHA)

AMPIEZZA TOTALE COSTANTE

Considerato che si sta lavorando in "hardware" è necessario tener conto del problema dell'*overflow*. Ciò significa che, desiderando variare l'intensità dell'eco, si deve mantenere all'incirca costante l'intensità sonora totale, o al più, non incrementarla. Supponendo che il segnale audio abbia un valore efficace costante, si desidera che:

(1) $$V_u \leqslant V_i \quad [V]$$

Ricordando la seguente convenzione:

- a_b : valore istantaneo della componente variabile;
- a_B : valore istantaneo totale;
- A_b : valore efficace della componente variabile;
- A_B : valore di riposo.

In tal modo, dato che l'uscita del A/D ha un valore interno al range di overflow, si ha che l'uscita non dovrebbe saturare.

La (1) può essere riscritta nel seguente modo:

(2) $$V_u \leqslant V_i \cdot (1-\alpha+\alpha) = \alpha V_i + (1-\alpha) V_i \quad [V]$$

Quindi, per un qualsiasi valore del parametro α si ha che il *D/A* (convertitore d'uscita) non andrà in overflow. Ora, scegliendo un valore $0<\alpha<1$ si ha che la (2) è sempre verificata se:

(3) $$V_u \leqslant \alpha V_i + (1-\alpha)\alpha V_i = \alpha[1+(1-\alpha)]V_i \quad [V]$$

AMPLIFICAZIONE SEGNALE RITARDATO

Il blocco 1 – α si occupa della moltiplicazione del segnale $s[n-N]=\alpha \cdot x[n-N]+w[n]$, ove $x_a(t) \mapsto x_a[n] \mapsto x[n]$. Risultato della moltiplicazione è $w[n]=(1-\alpha)s[n-N]$. Una più accurata analisi della maglia di reazione è trattata nella sezione di moltiplicazione diretta per α.

CODICE ARCHITETTURALE

```
-- alpha.vhd
-- Canziani Alfredo & Viviani Emanuele production
-- DEEI, Università degli studi di Trieste

-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
-- %                                                                          %
-- %                            LIBRERIE INCLUSE                              %
-- %                                                                          %
-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

```
-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
-- %                                                                               %
-- %                           PORTE DI 1-ALPHA                                    %
-- %                    (terminali di ingresso ed uscita)                          %
-- %                                                                               %
-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

entity uno_meno_alpha is
    Port (   data_delayed : in  std_logic_vector (15 downto 0);
             switch       : in  std_logic_vector (1 downto 0);
             data_out     : out std_logic_vector (15 downto 0)
             );
end uno_meno_alpha;

-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
-- %                                                                               %
-- %                              FUNZIONAMENTO                                    %
-- %                                                                               %
-- %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

architecture comportamento of uno_meno_alpha is

signal dato2: std_logic_vector (15 downto 0); -- segnali utilizzati e associati
signal dato4: std_logic_vector (15 downto 0); -- ad uno scalaggio per un fattore
signal dato8: std_logic_vector (15 downto 0); -- 1/2, 1/4 e 1/8 per il segnale ritardato

begin
    process (switch, data_delayed)
    begin

    case (switch) is  -- ricordando che gli switch si chiudono a massa e si aprono a Vcc
        when "11" =>    -- 1-alpha = 0
            data_out <= "0000000000000000"; -- ammazzo l'eco

        when "01" =>    -- 1-alpha = 1/8
            dato8 (15 downto 0) <= data_delayed (15) & data_delayed (15) &…
                …data_delayed (15) & data_delayed (15 downto 3); -- shift compl. a 2
            data_out <= dato8;

        when "10" =>    -- 1-alpha = 1/4
            dato4 (15 downto 0) <= data_delayed (15) & data_delayed (15) &…
                …data_delayed (15 downto 2);
            data_out <= dato4;

        when others =>  -- 1-alpha = 1/2     switch = 00
            dato2 (15 downto 0) <= data_delayed (15) & data_delayed (15 downto 1) ;
            data_out <= dato2;

        end case;

    end process;
end comportamento;
```

SHIFT PER NUMERI IN COMPLEMENTO A 2

Uno dei tanti errori in cui siamo incappati è stato quello inerente lo *shift*. Sapendo che la divisione per due, per i numeri binari, si effettua tramite lo shift a destra (come con i numeri decimali, la divisione per 10 si effettua semplicemente "spostando" i numeri verso destra), abbiamo cercato di creare questo operatore.

Quindi, partendo da un numero, espresso in base due, come ad esempio:

$$1\ 0010\ 1101_2$$

il risultato dell'operazione $301_{10} : 2_{10}$ è:

$$1001\ 0110_2$$

e fin qua non vi è alcun problema.

Consideriamo ora un dispositivo hardware, con quindi una dimensione *finita* di *bit* per *word*. Supponiamo essere il numero di bit a disposizione maggiore della *lunghezza* di 301_{10}. Riproponiamo lo stesso esempio:

$$0000\ 0001\ 0010\ 1101_2 : 10_2 =$$

$$= 0000\ 0000\ 1001\ 0110_2$$

A rigor di logica sembra che per effettuare uno shift sia necessario "spostare" i bit di una (o più) posizioni verso destra (sinistra, se moltiplichiamo) e aggiungere degli zeri "davanti" ("dietro").

Dopo aver implementato tale soluzione, ciò che si poteva udire dalle casse connesse al *J*2 era parte del segnale d'ingresso più una buona quantità di *rumore*. Inizialmente si era pensato ad un overflow del *blocco α*, ma non era così.

Consideriamo sempre il numero binario:

$$1\ 0010\ 1101_2$$

che può essere interpretato come -211_{10}; il risultato dell'operazione $-211_{10} : 2_{10} = -106_{10}$, pari a:

$$1\ 1001\ 0110_2 \neq 0\ 1001\ 0110_2 = 150_{10}$$

Quindi, a lato pratico, considerando una dimensione delle parole a 16 bit, si avrà che:

$$1111\ 1111\ 0010\ 1101_2 : 10_2 =$$

$$= 1111\ 1111\ 1001\ 0110_2$$

Ciò si traduce nel seguente *algoritmo di shift* (divisione per 2) per una *parola* a *N bit*:

- si copino gli $N-1$ bit della parola di partenza: da $bit1_{old}$ a $bit15_{old}$ in $bit0_{new}$ fino a $bit14_{new}$;
- si effettui la seguente associazione: $bit15_{new} = bit15_{old}$.

UN ESEMPIO DI DIVISIONE PER 8

```
new_word (15 downto 0) <= old_w (15) & old_w (15) & old_w (15) & old_w (15 downto 3);
```