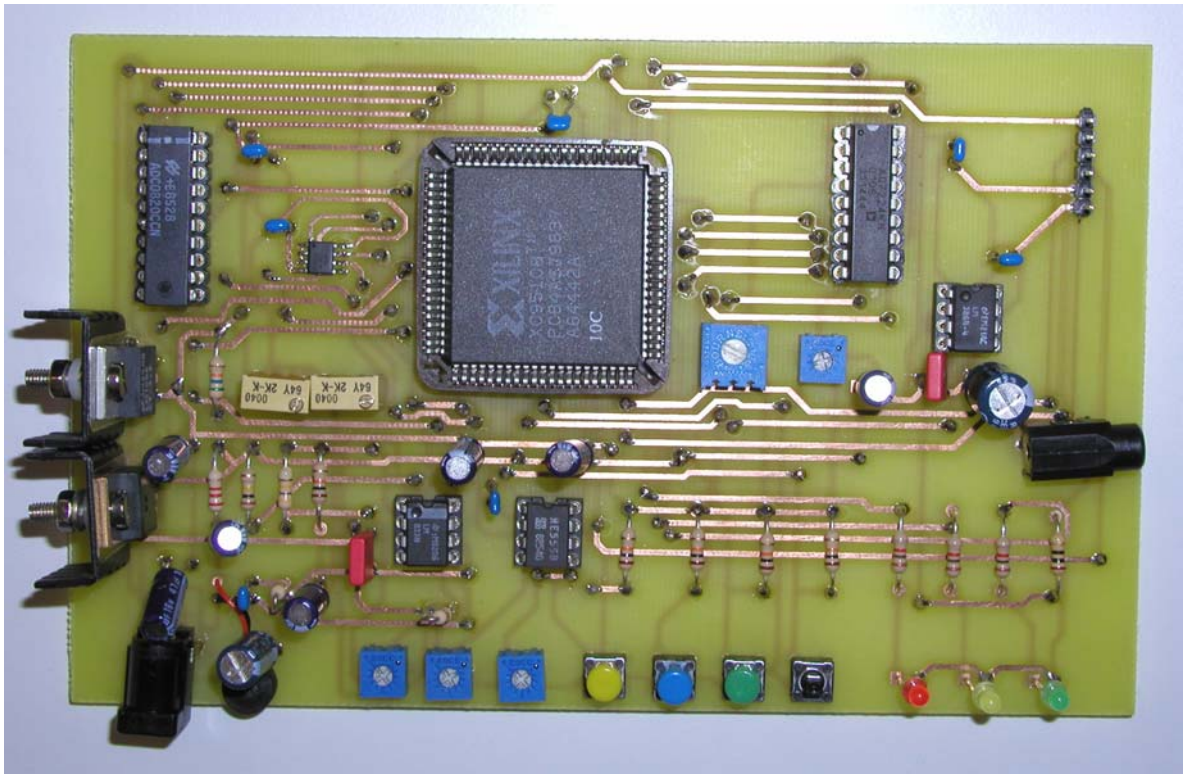
Universita' degli studi di Trieste

INGEGNERIA ELETTRONICA

Progettazione e realizzazione di un

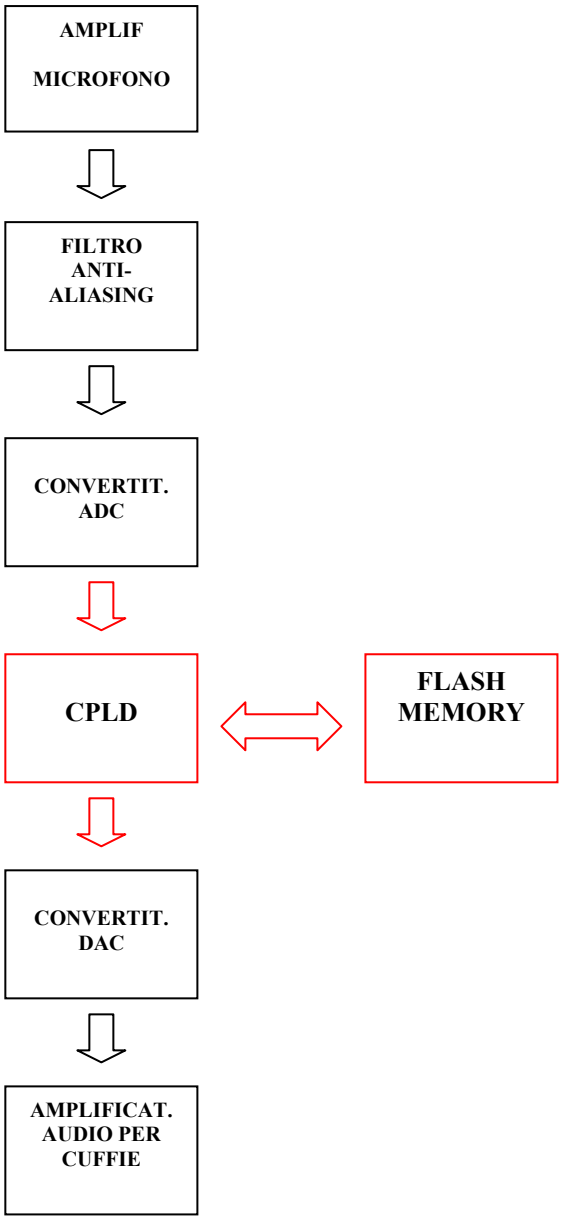


Registratore audio digitale

Docente: Stefano Marsi
Studente: Boz Rudi (rudiboz@inwind.it)

INTRODUZIONE

L'obbiettivo di questo progetto e' quello di realizzare un registratore audio digitale allo stato solido in grado di acquisire il segnale sonoro captato da un microfono e memorizzarlo in una memoria di tipo statico, in modo di riuscire a riprodurlo a piacere anche dopo aver spento e riacceso l'apparecchio.
Il progetto si divide quindi in 2 parti ben distinte, ovvero analogica e digitale.
Alla parte analogica spetta il compito di condizionamento iniziale del segnale microfonico e il condizionamento finale di quest'ultimo per interfacciarlo con delle cuffie; alla parte digitale spetta invece il controllo logico di funzionamento, nonche' la gestione della memorizzazione.
Per questa parte si e' fatto uso di una CPLD della Xilinx, interfacciata con una memoria di tipo flash della SST.
Lo schema a blocchi del circuito e' quindi gia' intuibile e puo' essere sintetizzato nel seguente:



In rosso si e' voluto evidenziare la parte digitale del progetto.

SCELTE PROGETTUALI

Di seguito si effettuano alcune considerazioni sui componenti da utilizzare e sulle strategie da seguire che possano influenzare la scelta dei componenti stessi.
Per quanto riguarda la CPLD e' stato scelto il modello XC95108 della Xilinx, nel package a 84 pin che garantisce un adeguato numero di pin di I/O accompagnato da un sufficiente numero di blocchi logici necessari allo sviluppo di tutte le funzioni.
Per quanto riguarda il supporto di memorizzazione e' stata scelta una FLASH memory di tipo seriale, e precisamente uno dei modelli SST25VFXXX appena prodotti dalla ditta californiana e inviatici in qualita' di campioni.
Questo particolare tipo di flash,rispetto agli altri dispositivi in commercio, ha il pregio di memorizzare i dati in maniera estremamente veloce (max 20us per byte) permettendone l'utilizzo per un campionamento audio.
Prevede inoltre la possibilita' di scrittura continua dell'intera area di memoria senza dichiararne ripetutamente l'indirizzo. In questo caso questa opzione risulta comoda perche' le scritture e letture sono sempre eseguite dall'inizio, senza quindi bisogno di indirizzamenti particolari.
Si tenga presente che la CPLD va alimentata con 5V ma prevede per la gestione degli I/O una tensione separata che puo' essere 5V oppure 3.3V. La flash invece funziona unicamente tra i 3V e i 3.6V.
Si e' scelto quindi di lavorare con questi 2 livelli di tensione, progettando il tutto in modo che non fosse necessario l'utilizzo di tensioni duali che di solito vengono usate negli amplificatori operazionali.
Scelta della frequenza di campionamento: considerando che la memoria usata ha una capacita' di 2Mbit (ovvero 262.144 byte) si e' cercato un compromesso tra qualita' dell'audio e tempo di registrazione. Si e' quindi deciso di effettuare un campionamento a 10Khz, che comporta approssimativamente una durata di registrazione di $262.144 \cdot \frac{1}{10.000} = 26$ secondi
Per fare un calcolo preciso della frequenza di campionamento bisogna tener conto del tempo di attesa per la scrittura di un byte tipico della flash (la SST assicura che dopo 20us il byte e' stato scritto) e il numero di bit necessari per scrivere ogni singolo byte.
Nel caso particolare di questa flash, la scrittura di ogni byte necessita degli 8 bit di dato e di una parola di 8 bit per far sapere alla flash che la scrittura continua all'indirizzo successivo.
Si intuisce che la frequenza di lavoro generale sara' molto piu' alta di quella di campionamento, e precisamente

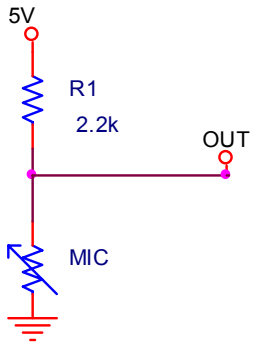
f = freq camp * (8+8+x)

dove con 'x' indichiamo il numero di cicli alla frequenza "f" tali da raggiungere 20us ovvero il tempo di scrittura.
E' stata considerata una frequenza di lavoro di 200KHz, cioe' T=1/f = 5us, quindi in questo caso x=4 e per scrivere ogni byte sono necessari 20 cicli di clock a 200KHz.
In questo modo la frequenza di campionamento risulta essere 20 volte piu' piccola, ovvero esattamente 10KHz.

PARTE ANALOGICA

AMPLICATORE MICROFONICO

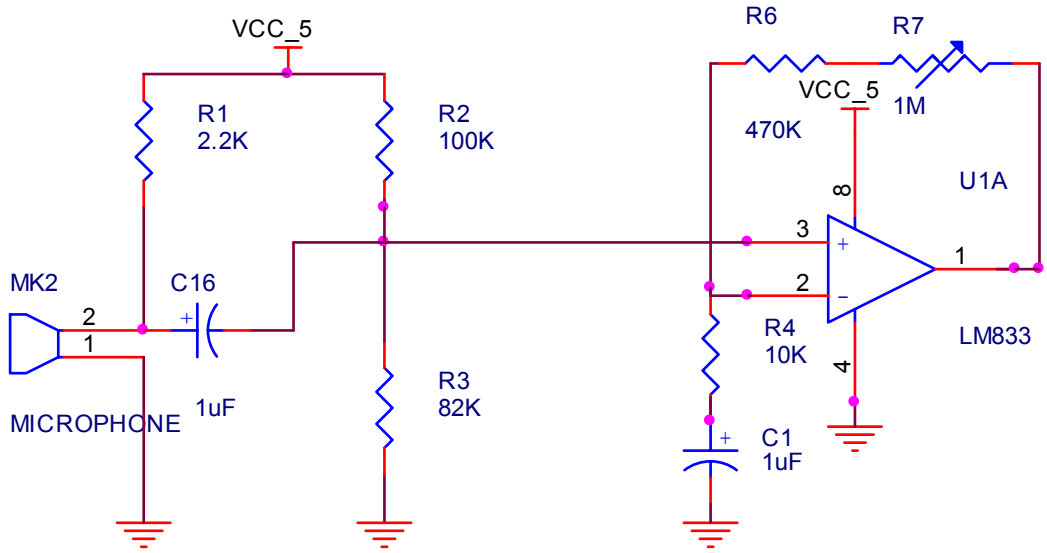
Scopo di questo primo stadio progettuale e' quello di ottenere da una normale capsula microfonica preamplificata, un segnale elettrico gestibile in maniera efficiente da un convertitore ADC.
Le normali capsule microfoniche preamplificate in commercio necessitano di essere alimentate dallo stesso pin da cui si preleva il segnale.
La ditta costruttrice garantisce il corretto funzionamento utilizzando 5V di alimentazione e una resistenza scelta tra un range di 1.5KΩ e 3KΩ.
Il microfono si comporta come una resistenza variabile la cui variazione e' causata dalla pressione sonora captata:



Montando il circuito su bread-board e utilizzando una resistenza da 2.2 kΩ come da figura, si osserva che la corrente assorbita a riposo (condizione di silenzio) e' di 317 μA mentre la caduta ai capi del microfono risulta di 4.31V.
La resistenza interna del microfono in stato di riposo risulta essere:

$$R_{\text{int}} = \frac{4.31}{317 \times 10^{-6}} = 13.596\Omega$$

Parlando in prossimita' del microfono si osserva una piccola variazione della resistenza interna dello stesso , che comporta in uscita una variazione di tensione di ampiezza molto ridotta.
L'operazione da eseguire e' una forte amplificazione del segnale microfonico, almeno di un fattore 50.
Lo schema utilizzato e' quello di un amplificatore non-invertente ad operazionale, con qualche accorgimento in piu' per adattare il segnale il meglio possibile al nostro scopo:



La resistenza R1 fornisce alla capsula microfonica la giusta alimentazione.
Il condensatore C16 e' un condensatore di accoppiamento.
La coppia di resistenze R2 e R3 creano un offset di tensione pari a

$$V_{cc} \times \frac{R_3}{R_2 + R_3} = 2.25V \ .$$

Il segnale proveniente dal microfono viene quindi sommato a questo offset, che non viene amplificato grazie a C1, il cui compito e' proprio di non amplificare la continua.
Questo stratagemma e' stato utilizzato per poter utilizzare l'amplificatore con singola alimentazione. In questo modo allo stato di riposo il microfono comporta un'uscita fissa a 2.25V, ed eventuali variazioni si ripercuotono facendo oscillare quest'ultima attorno questa soglia.
Si noti che l'offset di tensione non si poteva ottenere solo tramite opportuno dimensionamento di R1 che serve essenzialmente per l'alimentazione del microfono:
con il valore assegnato l'offset ammonta a 4.3V, valore troppo elevato per il nostro scopo,
per abbassarlo si puo' pensare di aumentare la R1, ma cio' comporterebbe una corrente insufficiente al corretto funzionamento del microfono stesso.
Si e' preferito quindi disaccoppiare la continua con C16 e risommarci un offset opportuno con la coppia di resistenze R2-R3.

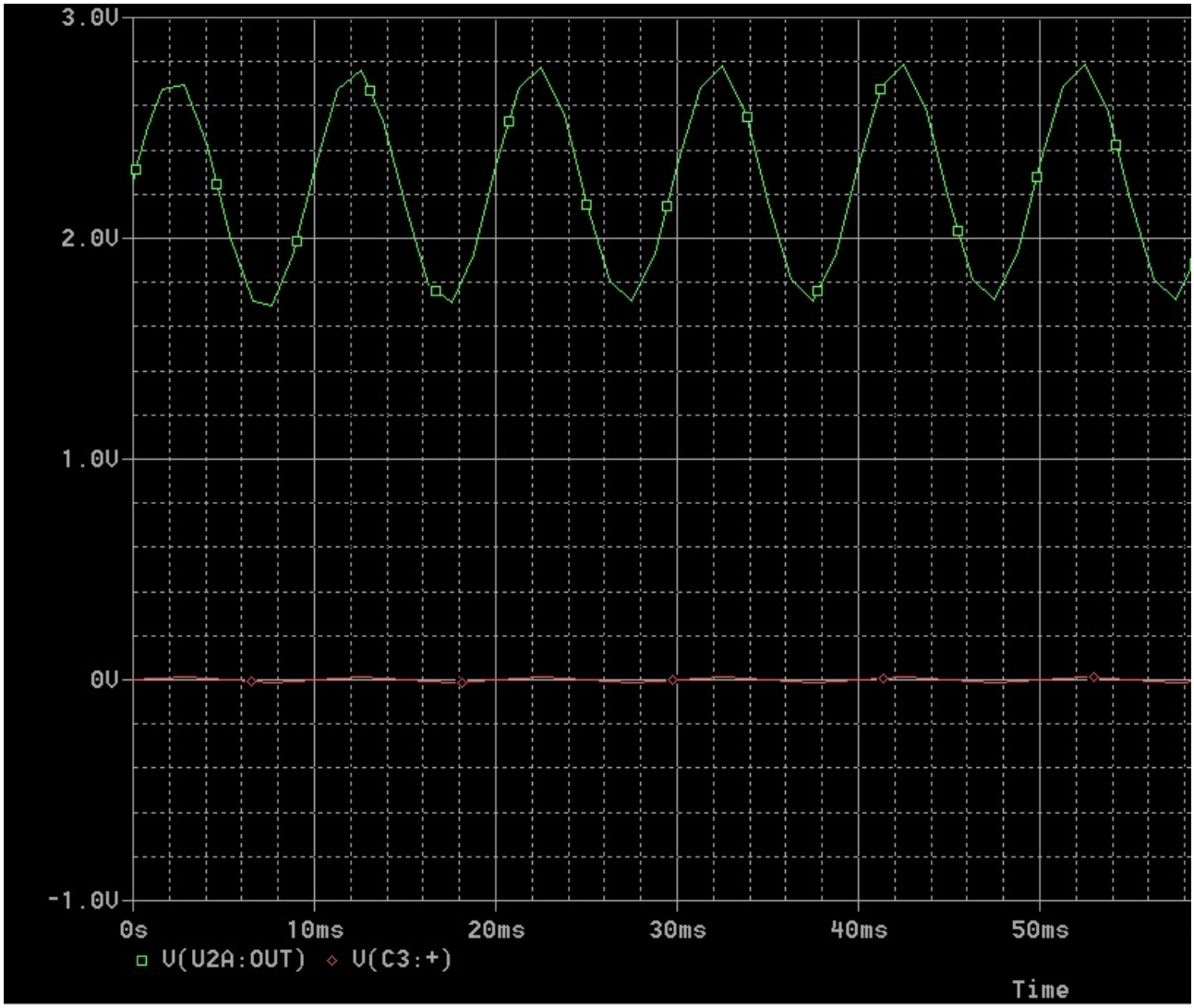
Le resistenze R6 ed R7 stabiliscono assieme alla R4 il guadagno del circuito tramite la relazione:

$$A_v = 1 + \frac{R_6 + R_7}{R_4}$$

La R6 e' stata messa in modo che con R7 in cortocircuito il guadagno sia limitato inferiormente al valore 48 , mentre variando la R7 si riesce a raggiunger un guadagno massimo di 148 .

.SIMULAZIONE CON SPICE

Il circuito e' stato simulato con SPICE utilizzando come segnale in ingresso un generatore sinusoidale alla frequenza di 1Khz e di ampiezza 5mV picco-picco. La R7 e' stata considerata in corto (guadagno di 48).
Per la simulazione si e' usato l'amplificatore operazionale TL082:

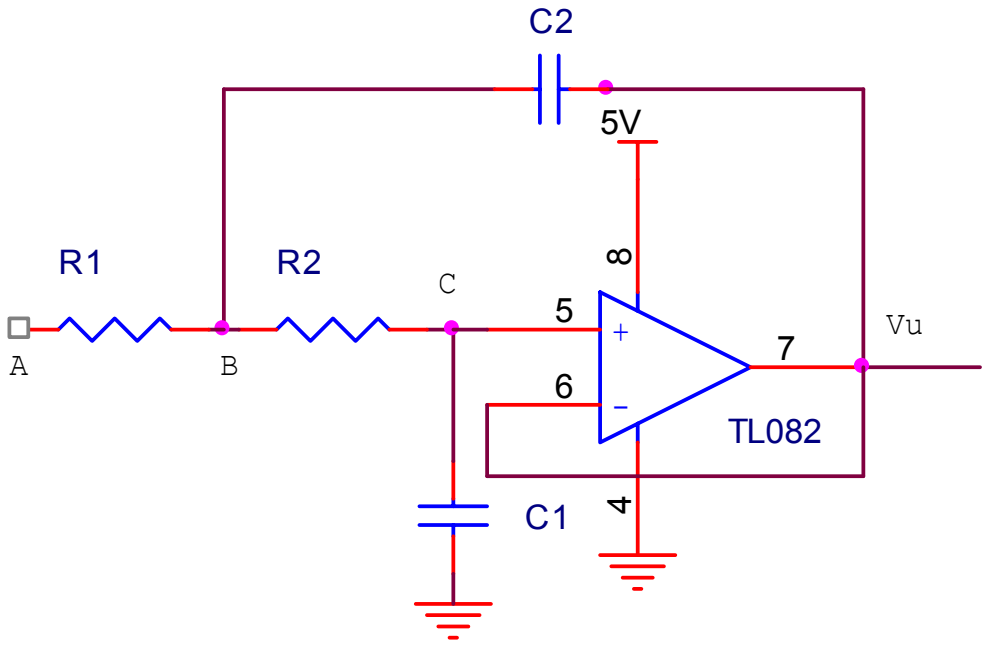


Il segnale in rosso rappresenta l'ingresso mentre quello in verde (come voluto con un offset a 2.25V) e' il segnale in uscita.

Il circuito e' stato montato su bread-board e testato verificandone l'uscita con l'ausilio di un oscilloscopio.

FILTRO ANTI-ALIASING

Effettuando un campionamento e volendo poi ricostruire il segnale di partenza, bisogna tener conto del teorema di Shannon che impone una frequenza minima di campionamento doppia di quella massima contenuta nel segnale.
Essendo la frequenza di campionamento di 10Khz, e' stato implementato un filtro attivo passa-passo del secondo ordine con frequenza di taglio a 5KHz utilizzando il seguente circuito :



Studiamone la funzione di trasferimento considerando l'operazionale ideale, quindi morsetti positivo e negativo allo stesso potenziale, impedenza d'ingresso infinita, impedenza d'uscita nulla.

Applicando il metodo dei nodi al nodo B si ottiene:

$$\frac{V_b - V_a}{R_1} + (V_b - V_u) \cdot Y_{c2} + \frac{V_b - V_c}{R_2} = 0$$

mentre per il nodo C si ottiene:

$$\frac{V_c - V_b}{R_2} + V_c \cdot Y_{c1} = 0$$

con $Y_{cx} = s \cdot C_x$ e $V_c = V_u$

Dalla seconda equazione si ricava Vb:

$$V_b = V_u \cdot (s \cdot C_1 \cdot R_2 + 1)$$

e sostituendola nella prima si ottiene:

$$V_u \cdot (s \cdot C_1 \cdot R_2 + 1) \cdot \left(\frac{1}{R_1} + s \cdot C_2 + \frac{1}{R_2} \right) = \frac{V_a}{R_1} + V_u \cdot \left(s \cdot C_2 + \frac{1}{R_2} \right)$$

Essendo la funzione di trasferimento definita dalla relazione

$$G(s) = \frac{V_{out}}{V_{in}} = \frac{V_u}{V_a}$$

si ottiene la seguente funzione di trasferimento:

$$G(s) = \frac{\dfrac{1}{R_1 \cdot R_2 \cdot C_1 \cdot C_2}}{s^2 + s \cdot \left(\dfrac{1}{R_1 \cdot C_2} + \dfrac{1}{C_2 \cdot R_2} \right) + \dfrac{1}{R_1 \cdot R_2 \cdot C_1 \cdot C_2}}$$

Si ricorda che per un generico quadripolo passa-basso del secondo ordine, la generica funzione di trasferimento risulta essere:

$$G(s) = \frac{\omega_0^2 \cdot A_0}{s^2 + \dfrac{\omega_0}{Q_0} \cdot s + \omega_0^2}$$

Dal confronto delle ultime due equazioni trovate risulta che:

1) $\omega_0^2 \cdot A_0 = \dfrac{1}{R_1 \cdot R_2 \cdot C_1 \cdot C_2}$

2) $\dfrac{\omega_0}{Q_0} = \dfrac{1}{C_2 \cdot R_1} + \dfrac{1}{C_2 \cdot R_2}$

3) $\omega_0^2 = \dfrac{1}{R_1 \cdot R_2 \cdot C_1 \cdot C_2}$

Confrontando la 1 e la 3 si vede che il guadagno in tensione e' unitario, mentre dalla 3 si ricava la pulsazione di taglio.

Considerando i valori commerciali dei componenti, sono state effettuate le seguenti scelte:

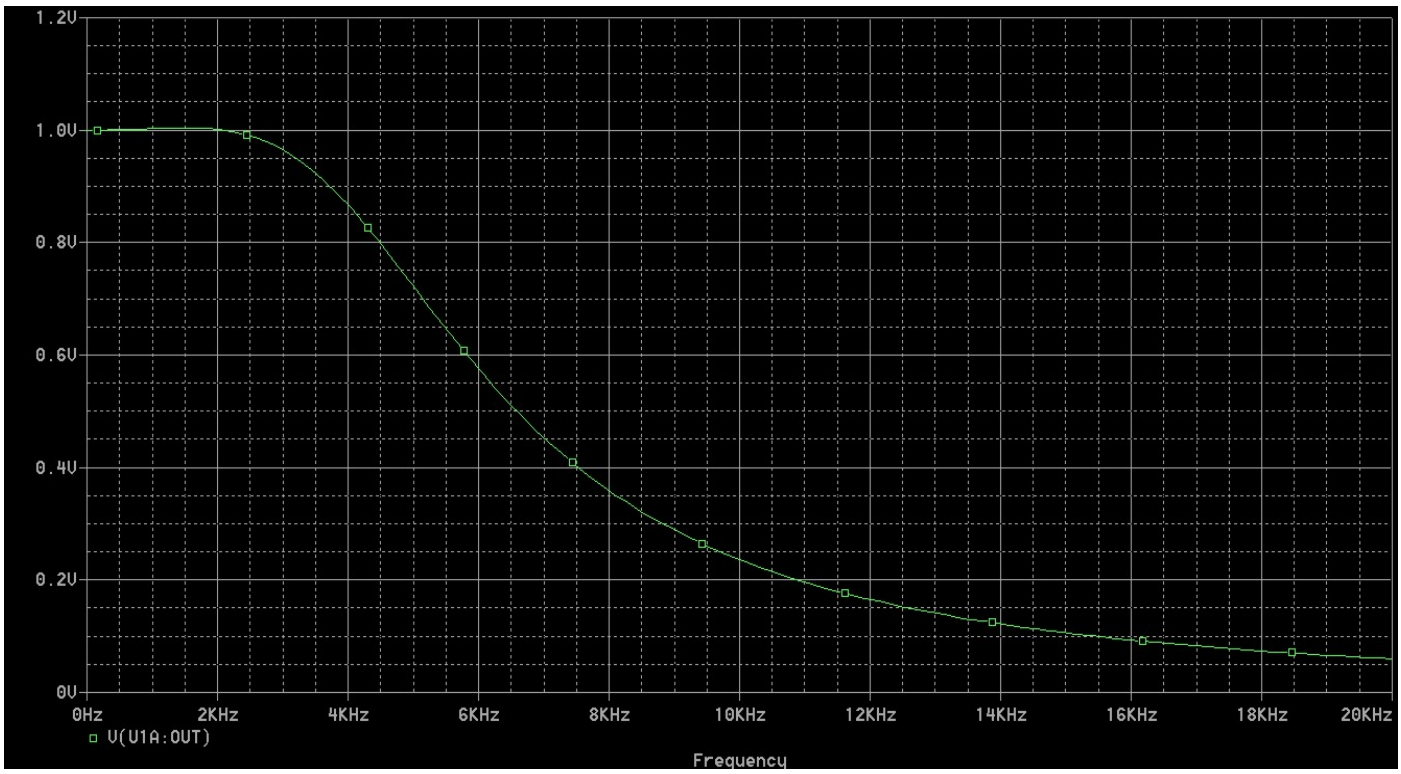
R1=R2=R= 2.2KΩ
C1=10nF
C2=22nF

Ottenendo quindi una frequenza di taglio di:

$$f_0 = \frac{\omega_0}{2\pi} = \sqrt{\frac{1}{R^2 \cdot C_1 \cdot C_2}} \cdot \frac{1}{2 \cdot \pi} = 4.877 Hz$$

.SIMULAZIONE CON SPICE

La risposta in frequenza del filtro progettato e' stata simulata con SPICE utilizzando come amplificatore operazionale il TL082.
Di seguito e' riportato l'andamento della simulazione:



Il circuito e' stato montato su bread-board e testato con l'ausilio di un generatore di funzioni e di un oscilloscopio.

CONVERTITORE ADC

La scelta del convertitore ADC deve soddisfare le seguenti richieste:

1) Funzionamento con alimentazione singola a 3.3V (per poter essere interfacciato direttamente con la CPLD la cui gestione degli I/O si ricorda essere stata fissata a 3.3V).

2) Tempo di conversione ridotto per poter lavorare anche con segnali in ingresso con elevato slew-rate.

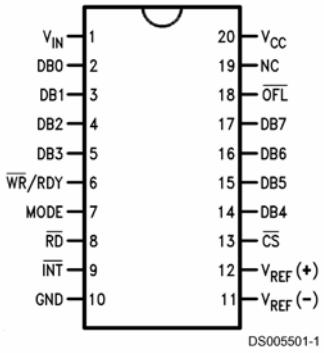
3) Gestione semplice del dispositivo.

E' stato scelto il modello a 8 bits della National ADC0820 che presenta i seguenti vantaggi:

1) Tempo di conversione ridotto (1.5us).
2) Non necessita di clock esterni.
3) Uscite provviste di latch.
4) Singola alimentazione.
5) Non necessita di circuiti di sample&hold esterni per segnali con slew-rate che non superino i 100mV/us.

Connection and Functional Diagrams

Dual-In-Line, Small Outline and SSOP Packages



Top View

Funzioni dei singoli pin:

Vin: ingresso analogico [IN]
DB0÷DB7: uscite digitali (DB0=LSB e DB7=MSB) [OUT]
Mode: ci sono 2 possibili funzionamenti, ovvero ‘’WR#/RD’’ quando questo pin e’ allo stato ‘1’ oppure ‘’RD’’ quando il pin e’ allo stato ‘0’. A seconda di questa scelta le funzioni dei prossimi pin saranno diversificate. [IN]
WR#/RDY: In mode “WR#/RD” e con CS#=’0’, una transizione 1→0 su questo pin comporta l’inizio della conversione. La successiva transizione 0→1 comporta la trasmissione del codice convertito sui pin d’uscita. [IN]
In “RD” mode questo pin diventa un output (RDY). Il suo stato va a ‘0’ dopo che sul pin CS# e’ avvenuta una transizione 1→0. Va in TRISTATE quando la conversione e’ terminata e il risultato e’ disponibile in uscita. [OUT]

RD#: In mode “WR#/RD” questo pin viene usato per mettere in alta impedenza le uscite, ovvero RD=’1’ significa uscite in alta impedenza, RD=’0’ significa uscite attivate. [IN]
In mode “RD” e con CS#=’0’ una transizione 1→0 su questo pin fa cominciare la conversione, e automaticamente abilita le uscite al termine di quest’ultima. La fine della conversione e’ indicata da ‘’RDY’’ che va in alta impedenza e da ‘’INT’’ che passa da 1→0. [IN]

INT#: Una transizione 1→0 di questo pin indica in entrambi i “mode” che la conversione e’ terminata e il risultato e’ disponibile in uscita. [OUT]

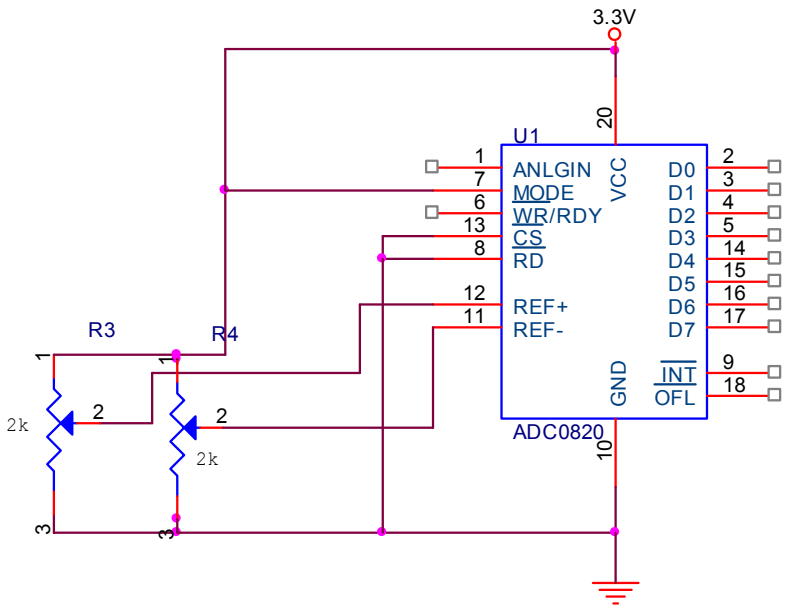
Vref(-): Indica la tensione che verra’ associata al codice ‘00000000’. [IN]

Vref(+): Indica la tensione che verra’ associata al codice ‘11111111’. [IN]

CS# : chip select, il dispositivo e’ attivo se il pin e’ a ‘0’ mentre e’ disattivato se e’ a ‘1’. [IN]

OFL#: pin di overflow, se l’ingresso analogico supera la tensione applicata a V(+), il pin va allo stato ‘0’ al termine della conversione. [OUT]

Per maggiori informazioni tecniche sul dispositivo si faccia riferimento al datasheet allegato.
La modalita’ di funzionamento scelta e’ quella “WR#/RD” e lo schema utilizzato e’ il seguente:



Giustificazioni:

CS# = RD# = GND (dispositivo sempre attivo e uscite sempre valide)
Mode = Vcc (seleziona mode WR#/RD)
INT# = OFL# = non connessi (il controllo sara’ effettuato dalla CPLD)
Vref(+) e Vref(-) dotati di ingresso variabile tramite un trimmer da 2kΩ.
Anlgin = qui verra’ collegata l’uscita del filtro vista in precedenza.
WR#/RDY = questo pin verra’ pilotato direttamente dalla CPLD per far partire la conversione.
D0÷D7 = sono collegati come ingressi alla CPLD (vettore d’ingresso ‘’fromadc’’)

Considerazioni:

Lavorando con una frequenza di sistema di 200KHz che comporta T=5us, si puo’ star sicuri che gia’ in un ciclo di clock il convertitore e’ in grado di fornire un dato valido in uscita.

Questo permette di evitare controlli sull'avvenuta conversione da parte della CPLD.
Inoltre si puo' star sicuri che il segnale in ingresso (filtrato a 5KHz) non varia durante il tempo di conversione (in pratica bisogna assicurare che il segnale analogico in ingresso durante il periodo di conversione non vari piu' della risoluzione del convertitore stesso).
Essendo un convertitore ad 8 bit si ha un range di tensione codificato con 2^8 livelli cioe' con 256 livelli.

CONVERITORE DAC

La scelta del convertitore DAC deve soddisfare le seguenti richieste:

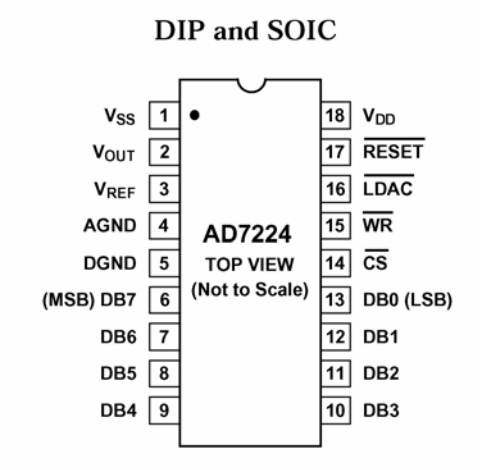
1) Tempo di conversione ridotto, campionando a 10KHz non dovra' superare 1/10000=100us.

2) Gestione semplice del dispositivo.

3) Funzionamento a singola alimentazione.

E' stato scelto il modello a 8bit della Analog Devices AD7224 le cui caratteristiche principali sono:

1) Tempo di conversione ridotto (90ns)
2) Funzionamento a singola alimentazione
3) Assenza di clock esterni
4) Uscita in tensione bufferizzata
5) Possibilita' di funzionamento in ''free-running'' ovvero in conversione continua.



Funzioni dei singoli pin:

Vss: nel caso di doppia alimentazione va' collegata alla tensione negativa, nel caso di singola alimentazione va' collegata a massa.
Vout: uscita analogica del dispositivo con riferimento ad AGND.
Vref: tensione d'uscita associata alla combinazione '11111111'.
AGND: massa del segnale d'uscita (puo' essere diversa da DGND).
DGND: massa del dispositivo (GND).
DB0÷DB7: ingressi digitali.

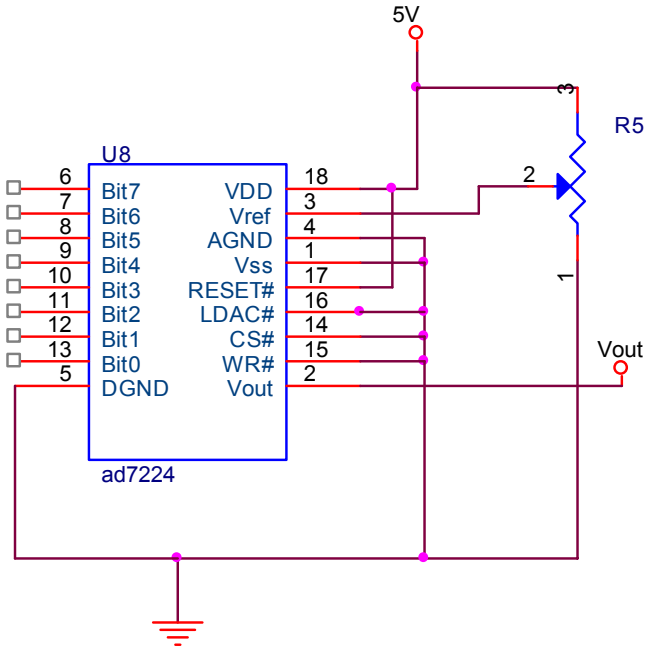
I segnali RESET#, LDAC#, WR#, CS# sono i segnali di controllo e soddisfano la seguente tabella di verita':

Table I. AD7224 Truth Table

$\overline{\text{RESET}}$	$\overline{\text{LDAC}}$	$\overline{\text{WR}}$	$\overline{\text{CS}}$	Function
H	L	L	L	Both Registers are Transparent
H	X	H	X	Both Registers are Latched
H	H	X	H	Both Registers are Latched
H	H	L	L	Input Register Transparent
H	H	↓	L	Input Register Latched
H	L	L	H	DAC Register Transparent
H	L	↓	H	DAC Register Latched
L	X	X	X	Both Registers Loaded With All Zeros
↓	H	H	H	Both Register Latched With All Zeros and Output Remains at Zero
↓	L	L	L	Both Registers are Transparent and Output Follows Input Data

H = High State, L = Low State, X = Don't Care.
All control inputs are level triggered.

La configurazione “free running” si ottiene dalla prima combinazione, infatti i registri sono trasparenti, e il dato convertito e’ trasferito in tempo reale all’uscita.
E’ stato utilizzato questo tipo di configurazione, e lo schema circuitale e’ il seguente:

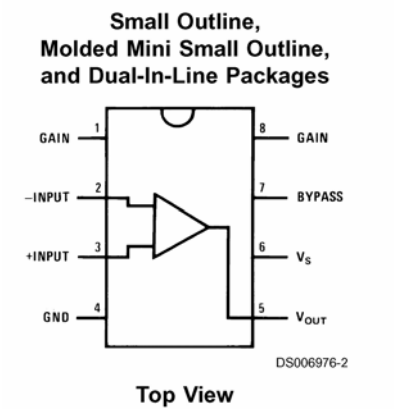


Il trimmer R5 permette di scegliere a piacere la tensione di riferimento.

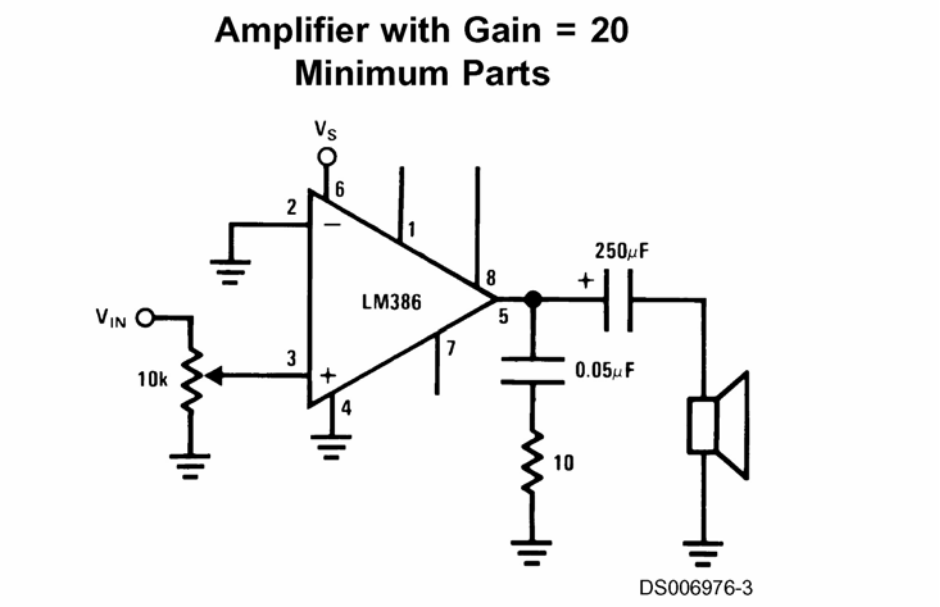
AMPLIFICATORE AUDIO PER CUFFIE

L’uscita del convertitore DAC non e’ in grado di pilotare direttamente un altoparlante oppure un paio di cuffie, quindi e’ stato inserito uno stadio amplificatore.
E’ stato scelto l’integrato LM386 che e’ un amplificatore audio a bassa tensione in grado di pilotare direttamente carichi da 4Ω a 32Ω.
Tra le sue caratteristiche principali si trova:

1) Ampio range di alimentazione (5V ÷ 18V)
2) Ampio range di guadagno (20 ÷ 200)
3) Componentistica esterna minima
4) Possibilita’ di alimentazione a batteria
5) Bassa distorsione



Il guadagno in tensione viene regolato agendo sui pin 1 e 8.
Se vengono lasciati aperti si ha il guadagno minimo pari a 20, se vengono collegati tramite una capacita' si ha il guadagno massimo pari a 200.
Porre in serie una resistenza e una capacita' tra questi due pin permette di impostare il guadagno tra 20 e 200.
Si e' scelto di lavorare col guadagno minimo pari a 20, lo schema elettrico suggerito dal datasheet e' il seguente:



Tramite il trimmer posto in ingresso si puo' regolare il livello del segnale.

Per maggiori informazioni relative al funzionamento interno dell'amplificatore si faccia riferimento al datasheet allegato.

TARATURA

Per la taratura dei circuiti si e' fatto uso della seguente strumentazione:

1) Oscilloscopio WaveTek 9020, 20MHz doppia traccia
2) Generatore di funzioni WaveTek FG2A
3) Multimetro digitale Metex M-3800
4) Alimentatore stabilizzato HP E3630A

Stadio microfono: si collega l'oscilloscopio in uscita e si varia il trimmer posto sulla retroazione finche' le escursioni del segnale sono accettabili (almeno 1V picco-picco).
Sono stati riscontrati i seguenti valori minimo e massimo di tensione in uscita, parlando a voce "normale":

V(max) = 2.7V
V(min) = 1.7V

Convertitore ADC: i rispettivi pin Vref(+) e Vref(-) del convertitore ADC vanno portati rispettivamente a 2.7V e 1.7V agendo sugli appositi trimmer.

Convertitore DAC: sapendo che l'amplificatore in cascata ha un guadagno fisso e pari a 20, e che una tensione di 2V su un paio di cuffie crea gia' un volume sufficiente, risulta:

$$V_{\max\_DAC} = \frac{2V}{20} = 100mV$$

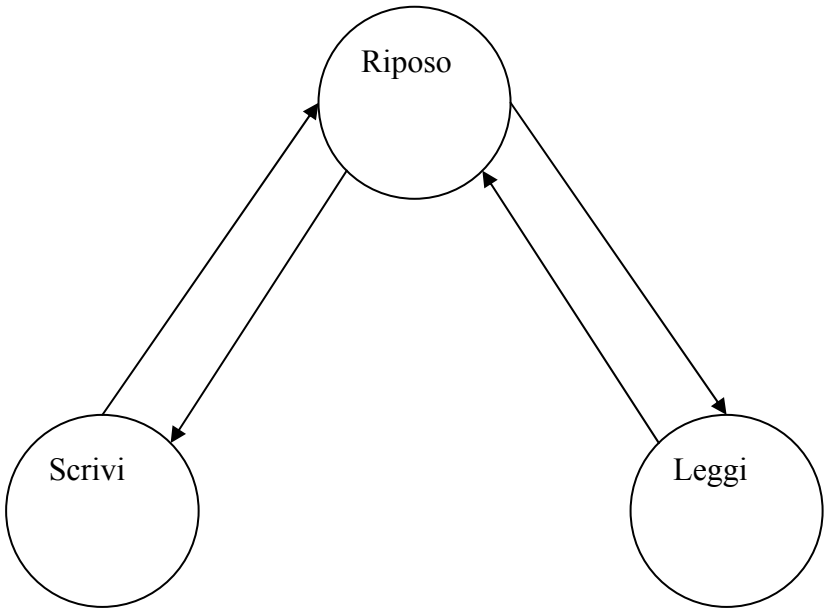
quindi e' stato portato il pin Vref del convertitore DAC a questa tensione, agendo sull'apposito trimmer.

NOTA: in questa maniera il DAC lavora in un range di 100 mV con una risoluzione di
0.1/256 = 0.39mV tra un codice binario e il suo adiacente.

PROGETTO DELLA SEZIONE DIGITALE

La sezione digitale comprende il colloquio tra la CPLD e la flash e l'interfaccia tra i due convertitori ADC e DAC.
La CPLD utilizzata e' della XILINX e precisamente il modello XC95108, dotata di 108 macrocelle, 2400 gates ,10 ns pin to pin delay, package da 84 pins (il package piu' ristretto possibile per questa versione).
La nostra esigenza in effetti e' quella di avere un adeguato numero di I/O disponibili ed abbastanza blocchi logici interni per poter implementare un'interfaccia per la flash.
A questo punto si consiglia di consultare il datasheet della flash memory perche' la maggior parte del progetto che segue ne deriva come conseguenza diretta.
Per il progetto software si e' utilizzato l' " ISE webpack 5.1" disponibile gratuitamente dal sito della Xilinx (www.xilinx.com).
Per quanto riguarda la stesura del codice sono possibili piu' vie, quella diretta tramite codice VHDL o Verilog, tramite schematico, oppure tramite definizione di macchina a stati finiti.
E' stato scelta l'ultima possibilita' in quanto l'algoritmo da sviluppare assomiglia molto a quello di un microprocessore e la presenza di cicli e routine facilita di molto questo tipo di approccio.
Compreso nell' ISE webpack c'e' il programma "STATECAD" che permette questo tipo di progettazione.
Un vantaggio e' che compreso nello "statecad" c'e' un simulatore "proprio" che permette direttamente di verificare il codice scritto ed eventualmente di correggerlo.

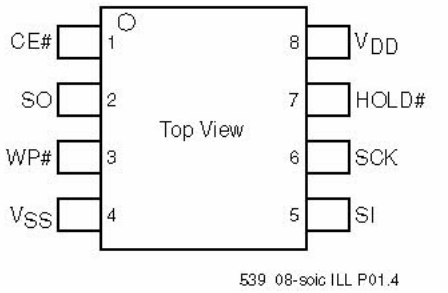
Schema di principio:



Lo stato “riposo” rappresenta lo stato in cui la nostra macchina sta’ di default. Da questo stato puo’ uscire (tramite pressione di pulsanti di comando) per andare univocamente in uno dei due stati previsti, o di scrittura o di lettura, che esegue interamente per poi tornare allo stato di riposo.

.Cenni sulla flash

Segue una breve descrizione della flash memory e dei suoi dati principali (si rimanda al datasheet per informazioni piu’ dettagliate):



8-LEAD SOIC
(SST25VF512/010/020 only)

Descrizione dei pin:

1) CE# = chip enable, attivo basso; deve essere attivo durante invio istruzioni o ricezione/invio dati,deve essere inattivo durante l’esecuzione delle istruzioni.
2) SO = serial output; e’ l’uscita da cui prelevare i dati in formato seriale.
3) WP# = write protect, attivo basso; se attivo protegge la memoria da qualsiasi scrittura, anche sul registro interno. Nel nostro caso va disattivato (messo a Vdd).
4) Vss = GND.
5) SI = serial input; pin di ingresso per dati e istruzioni in formato seriale.
6) SCK = clock di sistema.
7) HOLD# = con questo pin si puo’ sospendere qualsiasi sequenza sia in ingresso che in uscita senza dover resettare la memoria.
8) Vdd = Vcc

Il protocollo di accesso alla memoria e’ di tipo SPI, compatibile sia col modo “0” che col modo “3”. La differenza tra i due e’ che quando la periferica non e’ attiva, la linea di clock e’ settata a “0” nel primo caso oppure a “1” nel secondo.
Si e’ scelto di rispettare il modo “0”.

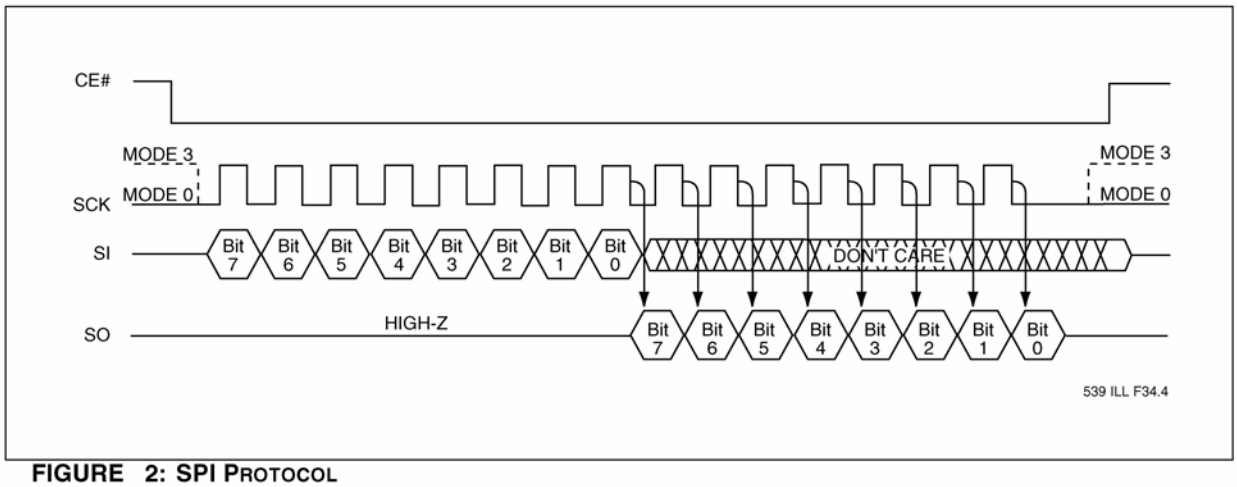


FIGURE 2: SPI PROTOCOL

Il controllo della periferica viene effettuato tramite istruzioni che sono specificate nella seguente tabella:

TABLE 6: DEVICE OPERATION INSTRUCTIONS[1]

Bus Cycle[2]	1		2		3		4		5	
Cycle Type/Operation[3,4]	S_{IN}	S_{OUT}	S_{IN}	S_{OUT}	S_{IN}	S_{OUT}	S_{IN}	S_{OUT}	S_{IN}	S_{OUT}
Read	03H	Hi-Z	A_{23}-A_{16}	Hi-Z	A_{15}-A_8	Hi-Z	A_7-A_0	Hi-Z	X	D_{OUT}
Sector-Erase[5,6]	20H	Hi-Z	A_{23}-A_{16}	Hi-Z	A_{15}-A_8	Hi-Z	A_7-A_0	Hi-Z	-	-
Block-Erase[5,7]	52H	Hi-Z	A_{23}-A_{16}	Hi-Z	A_{15}-A_8	Hi-Z	A_7-A_0	Hi-Z	-	-
Chip-Erase[6]	60H	Hi-Z	-	-	-	-	-	-	-	-
Byte-Program[6]	02H	Hi-Z	A_{23}-A_{16}	Hi-Z	A_{15}-A_8	Hi-Z	A_7-A_0	Hi-Z	D_{IN}	Hi-Z
Auto Address Increment (AAI) Program[6,8]	AFH	Hi-Z	A_{23}-A_{16}	Hi-Z	A_{15}-A_8	Hi-Z	A_7-A_0	Hi-Z	D_{IN}	Hi-Z
Read-Status-Register (RDSR)	05H	Hi-Z	X	D_{OUT}	-	Note[9]	-	Note[9]	-	Note[9]
Enable-Write-Status-Register (EWSR)[10]	50H	Hi-Z	-	-	-	-	-	-	-	-
Write-Status-Register (WRSR)[10]	01H	Hi-Z	Data	Hi-Z	-	-	-	-	-	-
Write-Enable (WREN)	06H	Hi-Z	-	-	-	-	-	-	-	-
Write-Disable (WRDI)	04H	Hi-Z	-	-	-	-	-	-	-	-
Read-ID	90H or ABH	Hi-Z	00H	Hi-Z	00H	Hi-Z	ID Addr[11]	Hi-Z	X	D_{OUT}[12]

T6.16 539

.Variabili usate come input/output

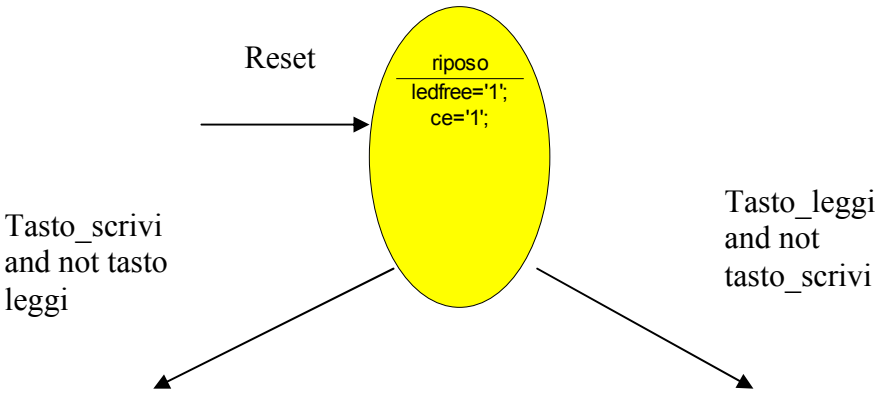
Si dichiarano di seguito gli ingressi e le uscite utilizzate in qualita' di tasti e led di visualizzazione:

1) Tasto_scrivi = tramite la pressione si passa alla routine di scrittura. (input)
2) Tasto_leggi = tramite la pressione si passa alla routine di lettura. (input)
3) Tasto_pause = tramite la pressione si puo' sospendere la riproduzione o terminare la registrazione. (input)
4) Reset = inizializza la macchina e la riporta allo stato "attesa" in qualsiasi momento .(input)
5) Ledfree = resta acceso quando la macchina e' libera ovvero nello stato "riposo". (output)
6) Lederase = resta acceso durante la procedura di cancellazione della memoria, passaggio che avviene in automatico prima di ogni scrittura).(output)
7) Ledbusy = indica che la macchina e' occupata, cioe' in qualsiasi stato che non sia quello di riposo.(output)
8) WR# = e' un comando che indica al convertitore ADC di partire con la conversione. E' attivo basso. (output)
9) NE = va collegato direttamente al clock della CPLD, per poter emulare assieme alla prossima variabile, il protocollo SPI mode "0".(input)
10) FLASHCLOCK = va collegato al clock della FLASH (SCK), in questo modo posso decidere tramite la CPLD quando NE=FLASHCLOCK oppure quando FLASHCLOCK="0", ovvero flash disattivata. (output)

A questi vanno aggiunti i due bus da 8bit, uno proveniente dall' ADC (fromadc, -input-) e uno diretto al DAC (todac, -output-).

Detto cio' si comincia ad analizzare il primo dei 3 stati definiti.
Per maggior chiarezza si consiglia di fare riferimento al file "definitivo.dia" allegato, che comprende l'intera macchina a stati finiti.

.STATO "RIPOSO"



Il circuito si inizializza alla prima pressione del tasto “reset” e resta in questo stato finche’ uno dei due tasti (tasto_scrivi o tasto_leggi) viene premuto, che sono le due condizioni di uscita.
Dentro questo stato le variabili definite assumono i seguenti valori:

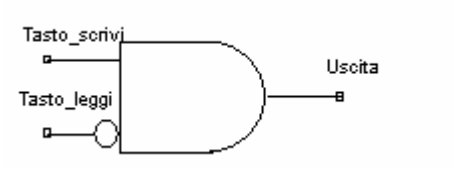
1)Ledfree=’1’ : il led verde e’ acceso, il sistema risulta libero.
2)CE=’1’ : indica che la flash e’ disabilitata.

Tutte le altre uscite non menzionate sono di default a “0”, quindi tutti gli altri led risultano spenti, FLASHCLOCK risulta a ‘0’ (flash in standby secondo protocollo SPI).

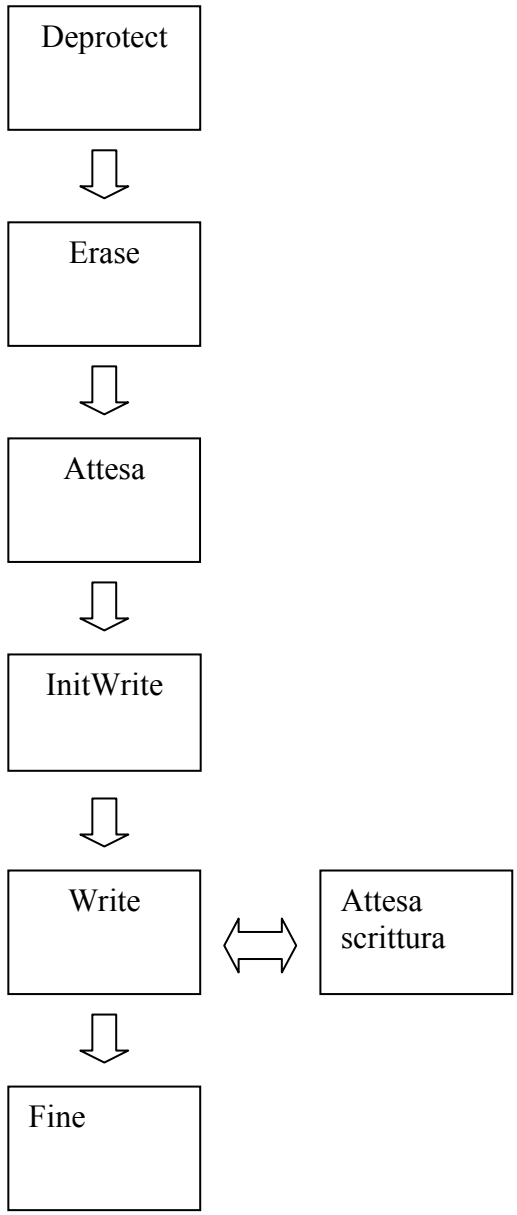
NB: nello “statecad”, scrivere “ ledfree=’1’ “ oppure “ledfree” e’ equivalente, ovvero scrivere la variabile omettendo lo stato logico equivale a sottintenderla a “1”, cosi’ come omettere il nome di una variabile equivale ad assegnarla a “0”.

.STATO “SCRITTURA”

Lo stato “scrittura” e’ stato suddiviso in piu’ stati elementari a se stanti, ciascuno dei quali svolge un’operazione di base della flash.
L’accesso a questo stato viene eseguito quando si verifica la seguente condizione: “ (tasto_scrivi and not tasto_leggi)=’1’ “ ovvero risulta attiva l’uscita della seguente porta:



Allo stesso modo viene eseguito l’accesso allo stato “lettura” semplicemente invertendo gli ingressi.
Le operazioni da fare per programmare la flash sono le seguenti:



Deprotect: ad ogni boot-up la flash di default e' impostata in modalita' "lettura" e tutta l'area di memoria risulta protetta. Gli stati della flash vengono modificati tramite scritture nel registro della flash stessa. Per una panoramica piu' dettagliata si faccia riferimento alla " tavola delle istruzioni" e allo "status register" della flash. Per prima cosa dobbiamo inviare alla flash il valore 50H (enable write status register, '01010000'), portare alto

CE# per un ciclo di clock (esecuzione istruzione), inviare il valore 01H (write status register, '00000001') e immediatamente inviare il byte con il valore desiderato per lo status register. In questo caso dobbiamo portare i bits 2 e 3 (BP0 e BP1) a '0' (intera memoria deprotetta), quindi il valore da inviare sara' "00000000" ossia 0H, seguito da una transizione 0 → 1 di CE# (esecuzione istruzione). I cicli necessari sono 8+1+16+1=26.
Se consideriamo i singoli bit in uscita al contatore come ingressi per la decodifica dei segnali da inviare alla memoria, possiamo definire la seguente tavola di verita:

Contatore	Ciclo	SerialOut	CE#
00001	1	0	0
00010	2	1	0
00011	3	0	0
00100	4	1	0
00101	5	0	0
00110	6	0	0
00111	7	0	0
01000	8	0	0
01001	9	X	1
01010	10	0	0
01011	11	0	0
01100	12	0	0
01101	13	0	0
01110	14	0	0
01111	15	0	0
10000	16	0	0
10001	17	1	0
10010	18	0	0
10011	19	0	0
10100	20	0	0
10101	21	0	0
10111	22	0	0
11000	23	0	0
11001	24	0	0
11010	25	0	0
11011	26	X	1

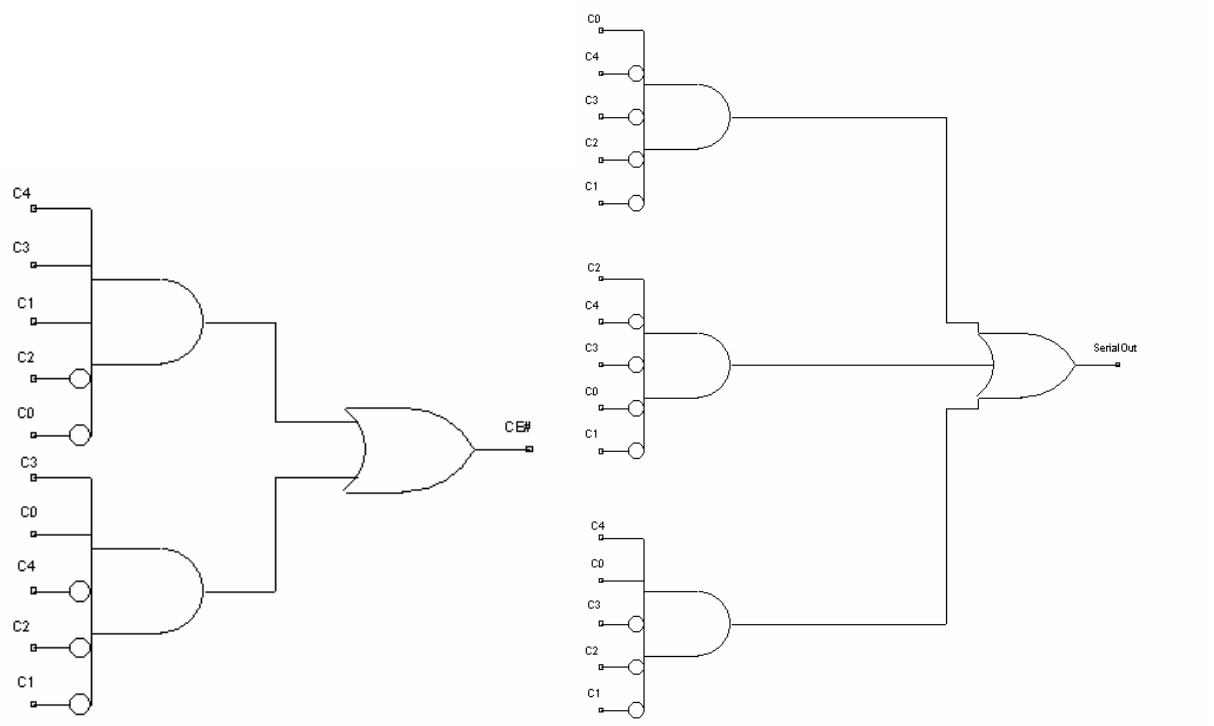
Si noti che "SerialOut" si riferisce all'uscita seriale della CPLD che quindi diventera' l'ingresso (SI) della flash. Creando un contatore che si incrementa ad ogni ciclo di clock, possiamo sfruttare le sue uscite piu' basse per decodificare i valori in tabella (in questo caso bastano i 5 bit meno significativi). I valori indicati con X sono valori ininfluenti e quindi vanno presi a piacere cercando di ottimizzare la logica.

Considerando le uscite a “1” risulta essere:

Serialout = $\overline{C_4}.\overline{C_3}.\overline{C_2}.\overline{C_1}.C_0+\overline{C_4}.\overline{C_3}.C_2.\overline{C_1}.\overline{C_0}+C_4.\overline{C_3}.\overline{C_2}.\overline{C_1}.C_0$

CE# = $\overline{C_4}.C_3.\overline{C_2}.\overline{C_1}.C_0+C_4.C_3.\overline{C_2}.C_1.\overline{C_0}$

Ovvero una rete logica cosi’ fatta:



Dove con Cx si indica il corrispondente bit del contatore. **Si tenga presente che nell’intero progetto si e’ fatto uso di 2 contatori , e precisamente “contatore” a 6 bits per eseguire le piccole decodifiche dentro gli stati, mentre il contatore “byte” a 22 bits per tenere sotto controllo il numero di byte scritti o letti.**

Erase: prima di essere scritto, ogni indirizzo della flash deve avere il valore FFH, cioe’ deve essere cancellato. Si e’ quindi inserito qui il ciclo di cancellazione che precede ogni scrittura. Per eseguire la cancellazione della flash sono necessarie due istruzioni: 06H (‘00000110’,write enable istruction) seguita dalla transizione 0→1 di CE# , e 60H (‘01100000’,chip erase). Allo stesso modo di prima si definisce la seguente tabella di verita’:

Contatore	Ciclo	SerialOut	CE#
00001	1	0	0
00010	2	0	0
00011	3	0	0
00100	4	0	0
00101	5	0	0
00110	6	1	0
00111	7	1	0
01000	8	0	0
01001	9	X	1

Contatore	Ciclo	Serialout	CE#
01010	10	0	0
01011	11	1	0
01100	12	1	0
01101	13	0	0
01110	14	0	0
01111	15	0	0
10000	16	0	0
10001	17	0	0

E quindi:

Serialout = $\overline{C_3} \cdot C_2 \cdot C_1 \cdot \overline{C_0} + \overline{C_3} \cdot C_2 \cdot C_1 \cdot C_0 + C_3 \cdot \overline{C_2} \cdot C_1 \cdot C_0 + C_3 \cdot C_2 \cdot \overline{C_1} \cdot \overline{C_0}$

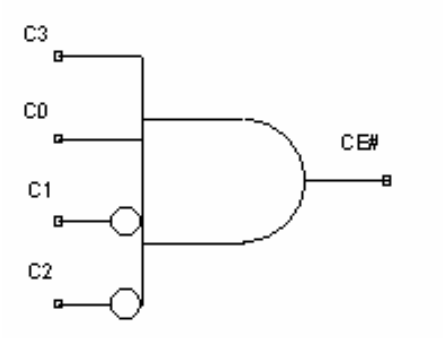
CE# = $C_3 \cdot \overline{C_2} \cdot \overline{C_1} \cdot C_0$

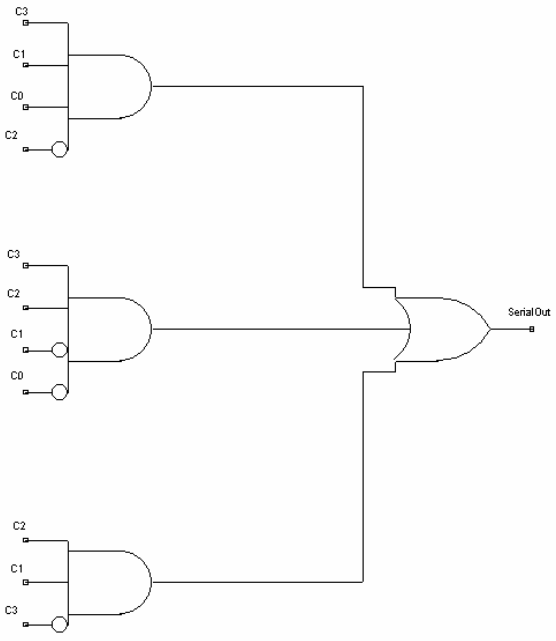
NOTA: tra il primo e secondo addendo di ‘’serialout’’ varia solo C0, quindi si puo’ togliere il riferimento a questo bit e usare un solo addendo a 3 componenti.

La forma semplificata diventa quindi:

Serialout = $\overline{C_3} \cdot C_2 \cdot C_1 + C_3 \cdot \overline{C_2} \cdot C_1 \cdot C_0 + C_3 \cdot C_2 \cdot \overline{C_1} \cdot \overline{C_0}$

La rete logica corrispondente risulta essere:





Attesa: per eseguire l'operazione di erase, CE# deve portarsi a livello '1' a starci finche' la procedura non sia terminata. Per non fare controlli sullo status register al fine di conoscere quando l'operazione e' terminata, semplicemente si attende il tempo massimo dichiarato per la cancellazione, cioe' 100ms.
Sperimentalmente si vede che la semplice pressione di un tasto assicura un contatto maggiore di 100 ms, quindi si considera che dalla pressione di "tasto_scrivi" al suo rilascio sia trascorso sicuramente questo tempo. Si impone l'uscita da questo stato quando "tasto_scrivi" viene rilasciato.
L'operazione di "erase" viene visualizzata oltre che dal led "busy", anche dal un terzo led "lederase" che permette una verifica visiva dell'operazione in corso.

InitWrite: dopo aver completato la procedura di erase, la flash torna allo stato di "read" e quindi per scriverci bisogna eseguire nuovamente il "write-enable" (06H) seguito da una transizione 0→1 di CE#. Successivamente si inizializza la procedura di AAI (auto address increment) in modo tale da scrivere l'intera area di memoria senza specificare l'indirizzo di volta in volta (comando AFH – '10101111'), seguito da 3 byte che rappresentano l'indirizzo di partenza (nel nostro caso tutti a 00H).

Tenendo presente che per semplificare la rete logica il contatore viene inizializzato al valore '07H', la tavola di verita' risulta essere la seguente:

Contatore	Ciclo	SerialOut	CE#
000111	1	0	0
001000	2	0	0
001001	3	0	0
001010	4	0	0
001011	5	0	0
001100	6	1	0
001101	7	1	0
001110	8	0	0
001111	9	X	1
010000	10	1	0
010001	11	0	0
010010	12	1	0
010011	13	0	0
010100	14	1	0
010101	15	1	0
010110	16	1	0
010111	17	1	0
011000	18	0	0
011001	19	0	0
011010	20	0	0
011011	21	0	0

Contatore	Ciclo	SerialOut	CE#
011100	22	0	0
011101	23	0	0
011110	24	0	0
011111	25	0	0
100000	26	0	0
100001	27	0	0
100010	28	0	0
100011	29	0	0
100100	30	0	0
100101	31	0	0
100110	32	0	0
100111	33	0	0
101000	34	0	0
101001	35	0	0
101010	36	0	0
101011	37	0	0
101100	38	0	0
101101	39	0	0
101110	40	0	0
101111	41	0	0

In questo stato ci si sofferma per 41 cicli di clock.

Risulta:

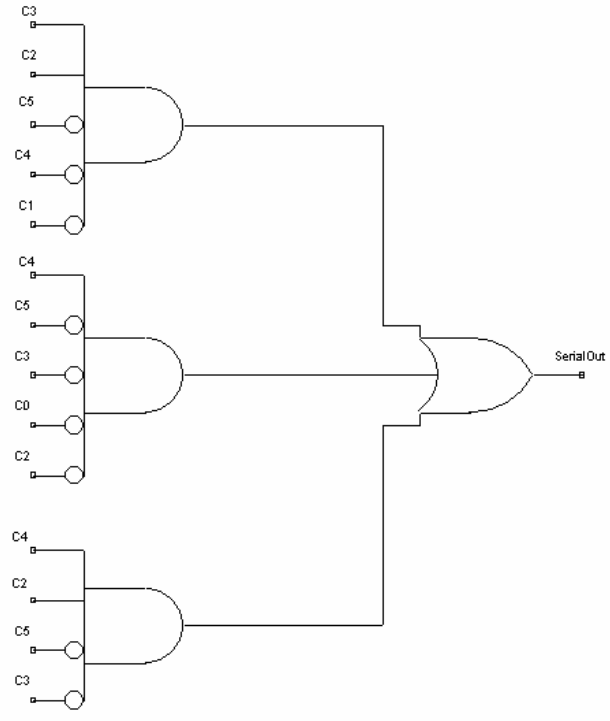
Serialout= $\overline{C_5}.\overline{C_4}.C_3.C_2.\overline{C_1}.(\overline{C_0}+C_0)+\overline{C_5}.C_4.\overline{C_3}.\overline{C_2}.\overline{C_0}.(\overline{C_1}+C_1)+$
$+\ \overline{C_5}.C_4.\overline{C_3}.C_2.(\overline{C_1}.\overline{C_0}+\overline{C_1}.C_0+C_1.\overline{C_0}+C_1.C_0)$

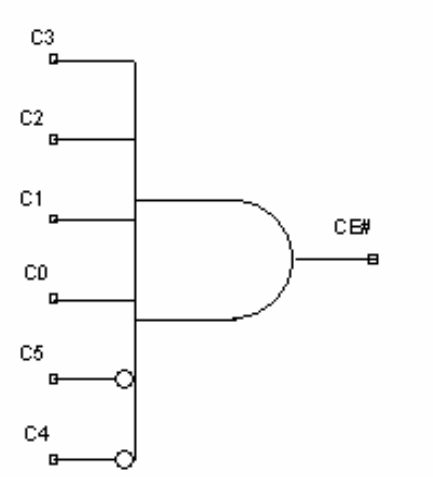
CE# = $\overline{C_5}.\overline{C_4}.C_3.C_2.C_1.C_0$

Quindi semplificando serialout risulta essere:

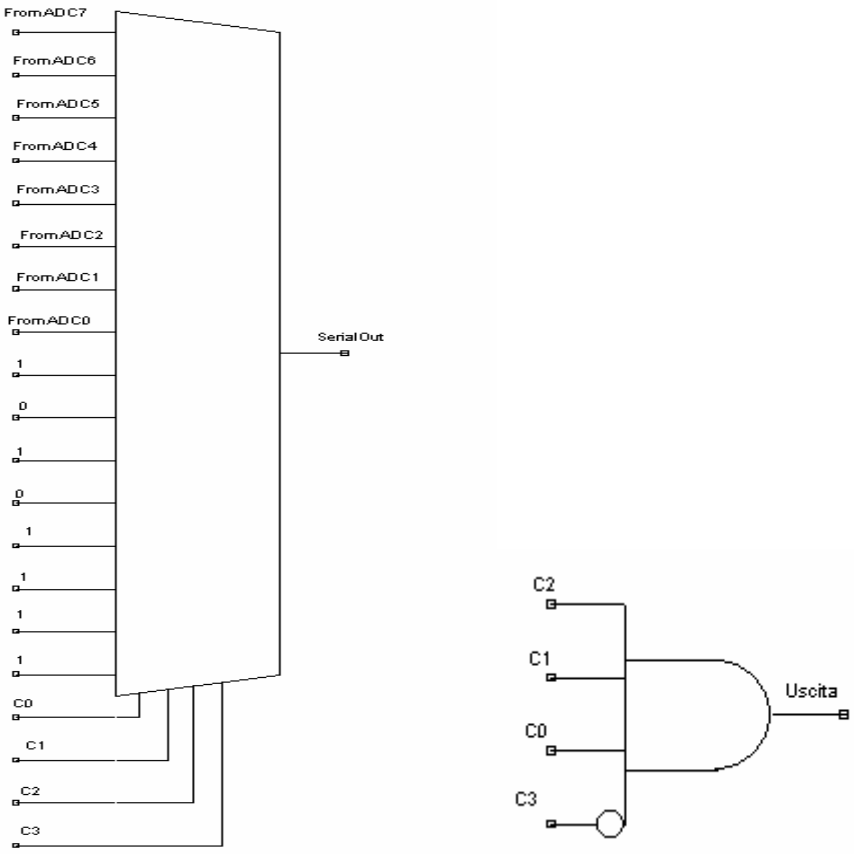
Serialout= $\overline{C_5}.\overline{C_4}.C_3.C_2.\overline{C_1}+\overline{C_5}.C_4.\overline{C_3}.\overline{C_2}.\overline{C_0}+\overline{C_5}.C_4.\overline{C_3}.C_2$

Ovvero risulta la seguente rete logica:





Scrittura & AttesaScrittura: questi due stati lavorano assieme creando un "loop" fino a che l'ultimo indirizzo di flash disponibile non e' stato raggiunto. Lo stato scrittura deve serializzare il byte letto in ingresso dall'ADC (la cui porta di comunicazione e' stata chiamata "fromadc") e mandarlo alla flash alternandolo al comando di AAI visto prima (che serve alla flash per sapere che la scrittura continua all'indirizzo successivo).
Per fare cio' si e' usato un multiplexer a 16 ingressi, i cui primi 8 bit sono fisicamente collegati a "fromadc" mentre i successivi otto rappresentano la costante AFH (AAI, '10101111').
I 4 bit che servono a decodificare i 16 ingressi del MUX sono i 4 bit meno significativi del contatore .
La variabile "uscita" fa uscire dallo stato ogni 16 cicli di clock.
A questo punto si devono inviare gli otto bit di "fromadc", uscire allo stato di attesa , aspettare che il byte sia scritto, tornare in questo stato e riprendere a far funzionare il MUX da dove si era fermato, cioe' dal bit8 al bit15 (AAI) e daccapo i primi 8 bit di "fromadc" .
La variabile "uscita" quindi dovra' identificare l'ottavo ciclo e riconoscerlo a multipli di 16. Cio' si ottiene semplicemente esprimendo in binario la cifra 7 (considero lo 0) e usando i soli 4 bit di contatore (prendo tutti i multipli di 16). Schematicamente ottengo:



Lo stato di “attesa scrittura” porta a “1” la linea CE# della flash memory per il tempo necessario alla scrittura del byte. Per assicurare cio’ senza monitorare lo status register, si attende il tempo massimo dichiarato dalla SST per la memorizzazione di un byte, ovvero 20us.
Lavorando a 200KHz cioe’ con T = 5us, saranno sufficienti 4 cicli di clock di sistema per assicurare la scrittura.
Contemporaneamente, lo stato di “attesa scrittura” porta a “0” la linea WR# che fa partire la conversione dell’ ADC.
L’ADC usato, tramite la linea WR# funziona come un LATCH, finche’ WR#=“1” in uscita si ha l’ultimo valore convertito, qualsiasi sia l’ingresso, mentre quando WR#=“0” parte la conversione e dopo poco piu’ di 1us e’ disponibile il codice in uscita.
In questo modo e’ assicurato che per quando il multiplexer ri-scansionera’ i primi suoi 8bit, il dato letto dal convertitore sara’ presente e stabile.
Nello stato di attesa viene inizializzato il contatore “byte” a 22 bit che ogni volta che si entra ed esce da questo stato risulta incrementato di 4 (essendo lo stato di attesa lungo 4 cicli di clock). Tramite esso si valuta quando l’ultimo byte e’ stato scritto.
Per questioni di comodita’ si e’ prevista la possibilita’ di terminare la registrazione anche prima dello scadere dei 26 secondi, tramite pressione del tasto_pause.
Dallo stato di “attesa scrittura” quindi si puo’ uscire in due direzioni diverse e mutuamente esclusive:

1) Torna allo stato scrittura (in automatico ogni 4 cicli di clock)
2) Esco allo stato “fine” nel caso in cui siano a “1” il tasto_pause oppure byte20. Quando byte20=’1’ infatti abbiamo contato esattamente (2^20)/4= 262.144 byte ovvero ho scritto l’intera flash da 2Mbits.

Fine: questo stato e’ necessario perche’ la procedura di scrittura della flash necessita di un comando che ne indichi il termine, altrimenti non sara’ possibile effettuare la lettura a meno di resettare la flash.
Il comando da inviare e’ 04H (‘0000100’) seguito dalla transizione 0→1 di CE#.
Visto che l’uscita avviene automaticamente verso lo stato di riposo che prevede CE# = ‘1’, si puo’ omettere portarlo a ‘1’ qua e inviare solo l’istruzione 04H: appena si passera’ allo stato di riposo verra’ eseguita.

La tabella di verita’ risulta essere:

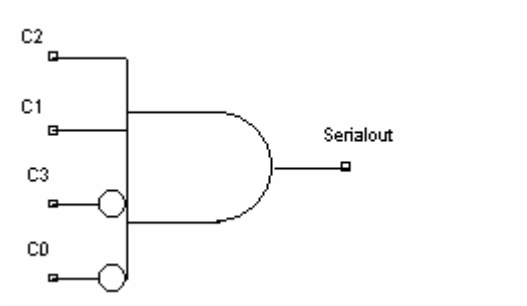
Contatore	Ciclo	Serialout	Uscita
0001	1	0	0
0010	2	0	0
0011	3	0	0
0100	4	0	0
0101	5	0	0
0110	6	1	0
0111	7	0	0
1000	8	0	1

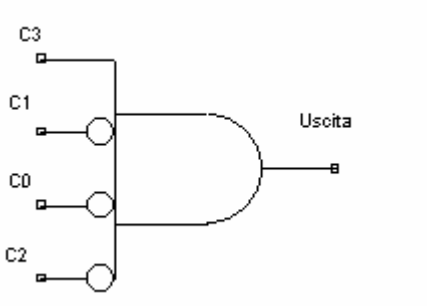
Ovvero valgono:

Serialout = $\overline{C_3} \cdot C_2 \cdot C_1 \cdot \overline{C_0}$

Uscita = $C_3 \cdot \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0}$

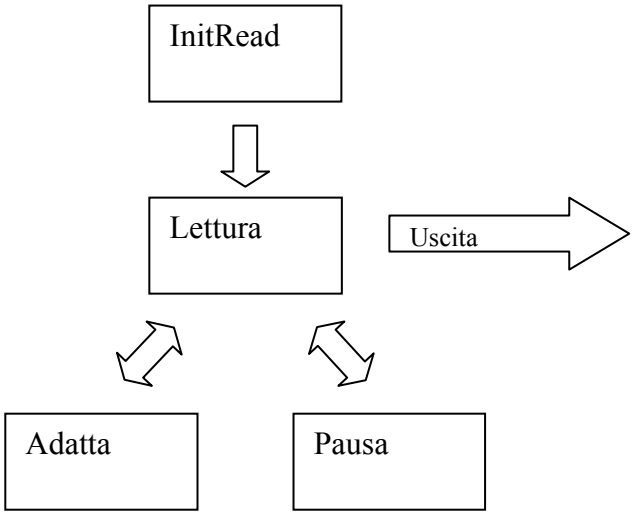
Ovvero i rispettivi schematici saranno:





.STATO “LETTURA”

Lo schema a blocchi della routine di lettura e’ la seguente:



InitRead: consiste nel portare CE# a livello basso ed inviare alla flash il comando di “read”, ovvero il valore 03H (‘00000011), seguito da 3 byte indicanti l’indirizzo di lettura di partenza (in questo caso tutti zeri).
Automaticamente dopo il ventiquattresimo bit di indirizzo, dall’uscita della flash cominciano ad uscire in flusso continuo i bit richiesti, senza bisogno di ulteriori comandi. Una transizione da 0→1 di CE# fa uscire la flash dallo stato di lettura.
Allo stesso modo di prima quindi definiamo la tabella di verita’ per serialout (per CE# non serve poiche’ e’ sempre a ‘0’):

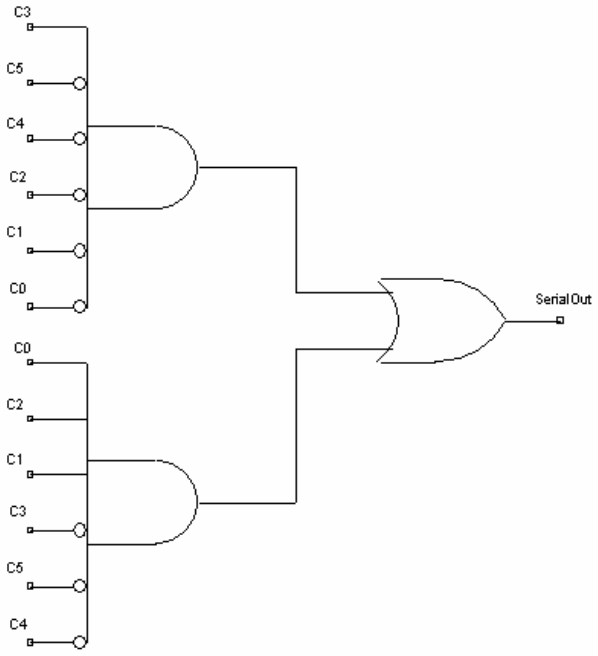
Contatore	Ciclo	SerialOut
000001	1	0
000010	2	0
000011	3	0
000100	4	0
000101	5	0
000110	6	0
000111	7	1
001000	8	1
001001	9	0
001010	10	0
001011	11	0
001100	12	0
001101	13	0
001110	14	0
001111	15	0
010000	16	0

Contatore	Ciclo	SerialOut
010001	17	0
010010	18	0
010011	19	0
010100	20	0
010101	21	0
010110	22	0
010111	23	0
011000	24	0
011001	25	0
011010	26	0
011011	27	0
011100	28	0
011101	29	0
011110	30	0
011111	31	0
111000	32	0

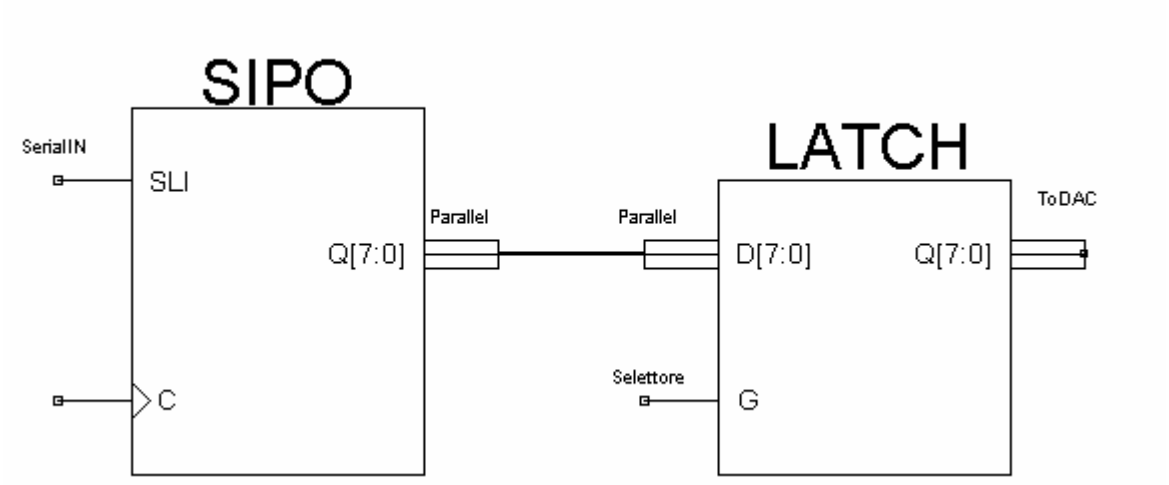
Cioe':

Serialout = $\overline{C_5} \cdot \overline{C_4} \cdot \overline{C_3} \cdot C_2 \cdot C_1 \cdot C_0 + \overline{C_5} \cdot \overline{C_4} \cdot C_3 \cdot \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0}$

Lo schema logico risulta essere il seguente:



Lettura/adatta: lo stato di lettura deve "parallelizzare" i bit provenienti dalla flash al fine di ricostruire il byte iniziale.
La flash fornisce di continuo e senza interruzioni i 2 Mbit presenti al suo interno.
A tal proposito si e' usato un SIPO (serial in parallel out) la cui uscita e' fisicamente collegata con gli 8 bits di ingresso del DAC attraverso la porta "todac".
Tra il SIPO e la porta d'uscita e' stato aggiunto un latch che sia in grado di "fotografare" al momento opportuno la sequenza di bit al fine di ricostruire il byte originale.
Lo schematico risulta essere il seguente:



Problema sulla riproduzione: scrivere un byte in memoria costa in termini di cicli di sistema 8+8+4 = 20 cicli di clock mentre la lettura ne occupa solamente 8; come risultato si ottiene un bit-rate in lettura molto maggiore di quello della registrazione.
Bisogna operare un 'adattamento' al fine di adeguare i due bit-rate.
Questo viene ottenuto sfruttando il pin "hold#" della flash, che permette di sospendere qualsiasi sequenza senza resettare la periferica.
Nella realizzazione proposta, lo stato 'lettura' dura 16 cicli di clock: durante i primi otto si legge il byte proveniente dalla flash, i successivi 8 si mette la flash in "hold" e si attiva "selettore" del latch fotografando il byte nella porta "todac".
Dopo il sedicesimo ciclo di clock si va nello stato "adatta" che dura 4 cicli (mantenendo 'hold' a '0'), al fine di arrivare ai 20 cicli totali per leggere ogni singolo byte; in tal modo si adatta la velocita' di lettura a quella di scrittura.

Nella realizzazione proposta, i segnali di controllo per lo stato "lettura" e "adatta" sono stati decodificati nel seguente modo:

LETTURA:

Contatore	Ciclo	HOLD#	Selettore	Uscita
00001	1	1	0	0
00010	2	1	0	0
00011	3	1	0	0
00100	4	1	0	0
00101	5	1	0	0
00110	6	1	0	0
00111	7	1	0	0
01000	8	1	0	0
01001	9	0	1	0
01010	10	0	0	0
01011	11	0	0	0
01100	12	0	0	0
01101	13	0	0	0
01110	14	0	0	0
01111	15	0	0	0
10000	16	0	0	1

ADATTA:

Contatore	Ciclo	HOLD#	Selettore	Uscita
001	1	0	0	0
010	2	0	0	0
011	3	0	0	0
100	4	0	0	1

Come si puo' notare, la variabile 'uscita' attiva sulla combinazione '0000' mi porta allo stato ''adatta'' prolungando di 4 cicli lo stato '0' di 'HOLD#'.
Al ritorno nello stato di lettura il contatore parte da dove si era fermato, cioe' riprende dall'inizio (0001) come indicato dalle frecce.

Per lo stato lettura risulta essere:

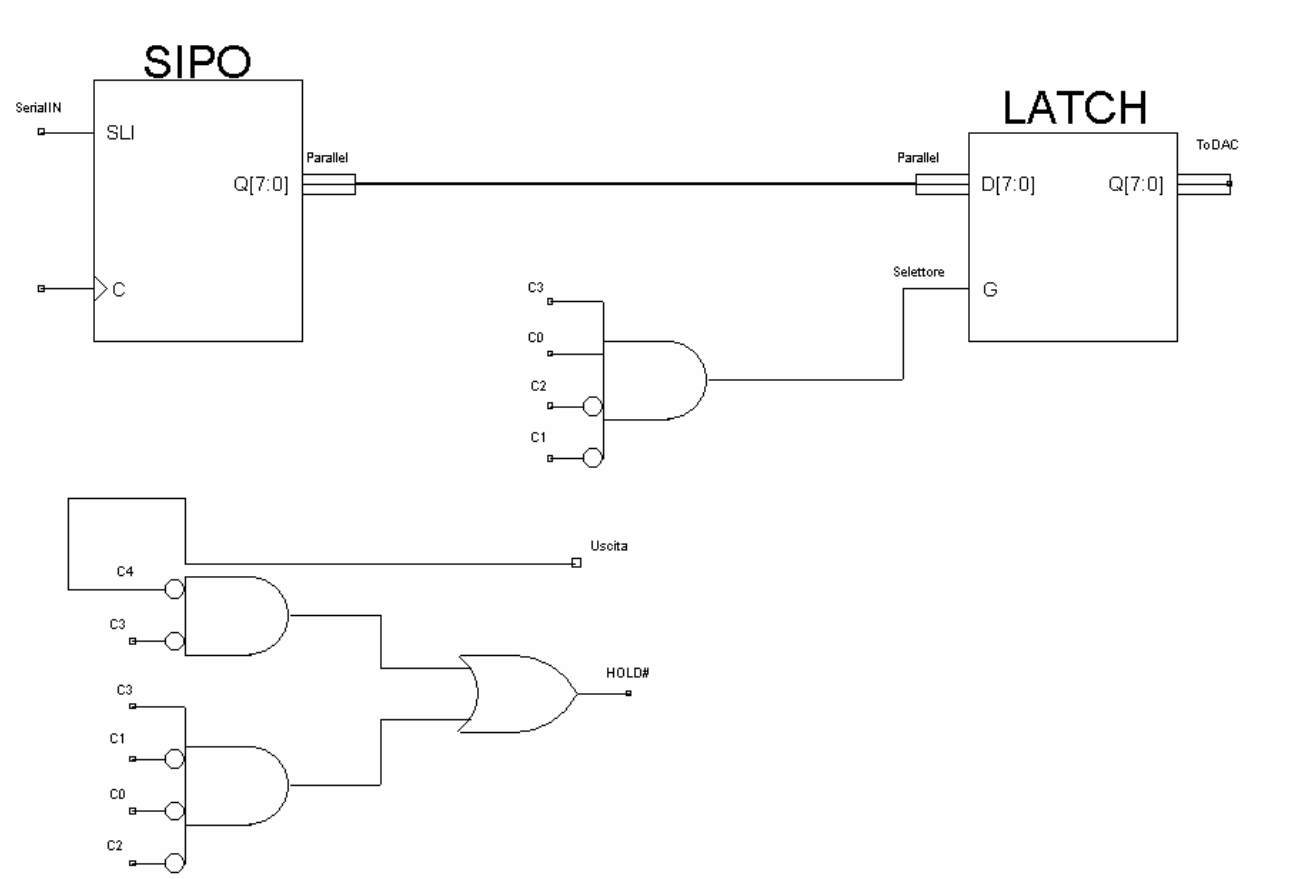
Hold# = $\overline{C_4} \cdot \overline{C_3} + C_3 \cdot \overline{C_2} \cdot \overline{C_1} \cdot \overline{C_0}$

Selettore = $C_3 \cdot \overline{C_2} \cdot \overline{C_1} \cdot C_0$

Uscita = C_4

Mentre lo stato ‘adatta’ si limita a portare HOLD# = ‘0’ e uscire dallo stato quando C_3 =’1’.

Per lo stato di ‘lettura’ si ha il seguente schematico:



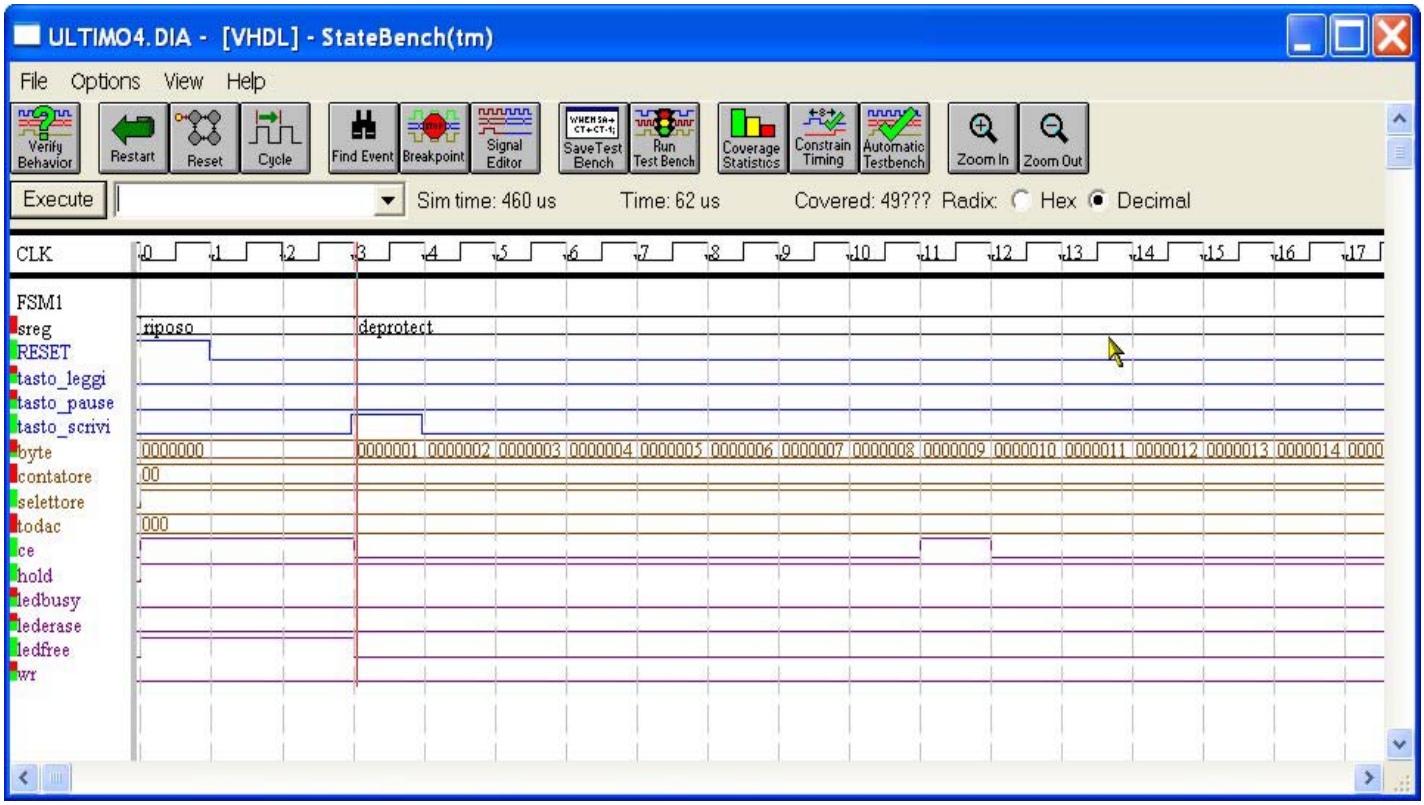
Nello stato di “lettura” si usa il contatore ‘byte’ a 22 bits al fine di contare i byte letti.
Per leggere ogni byte si impiegano 16 cicli di clock quindi l’uscita verso lo stato “riposo” avviene quando byte22=’1’ ovvero si sono contati

$$\frac{2^{22}}{16} = 262.144 \text{ byte}$$

cioe’ l’intero contenuto di memoria.

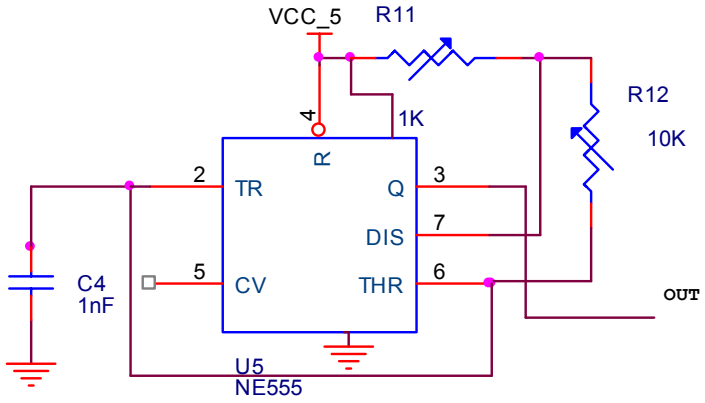
Pausa: Tramite pressione di “tasto_pause” si entra in questo stato che semplicemente porta il pin ‘HOLD#’ a ‘0’ sospendendo il flusso di dati letti. Si ritorna allo stato di lettura premendo il ‘tasto_leggi’.
A livello utente questo si traduce in una vera e propria pausa di riproduzione.

NB: Gli stati sopra descritti sono stati simulati interamente tramite il simulatore incorporato nello "statecad" di cui riportiamo una schermata a titolo esemplificativo:



PROGETTO DEL GENERATORE DI CLOCK

Per generare il segnale di clock di sistema a 200 KHz e' stato utilizzato l' integrato NE555 nella sua configurazione come multivibratore ASTABILE. Lo schema implementato e' quello suggerito dal datasheet e risulta essere il seguente:



La frequenza di oscillazione risulta definita da:

$$f = \frac{1.44}{(R_{11} + 2R_{12}) \cdot C4}$$

mentre il duty-cycle risulta essere definito da

$$\text{Duty} = \frac{R_{11} + R_{12}}{R_{11} + 2R_{12}}$$

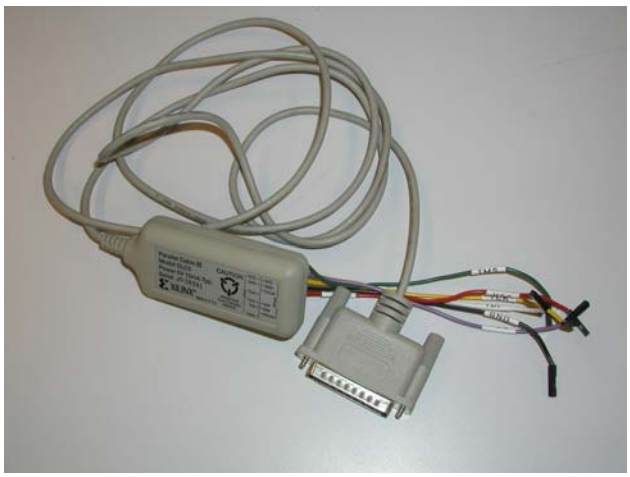
Per avere un duty cycle del 50% , basta fare in modo che R12>>R11.

Sono state fissate R11= 1KΩ e R12= 10KΩ

In questa modo il valore della capacita' C4 resta determinata dalla formula scritta sopra considerando una frequenza di 200 KHz.
Si e' cercato di utilizzare un valore di capacita' commerciale che ci facesse andare in prossimita' di questa frequenza, per poi regolarla precisamente tramite il trimmer R12.
Come valore di capacita' e' stato usato C4=1nF mentre la taratura del dispositivo e' stata ottenuta utilizzando l'oscilloscopio e agendo sul trimmer R12..

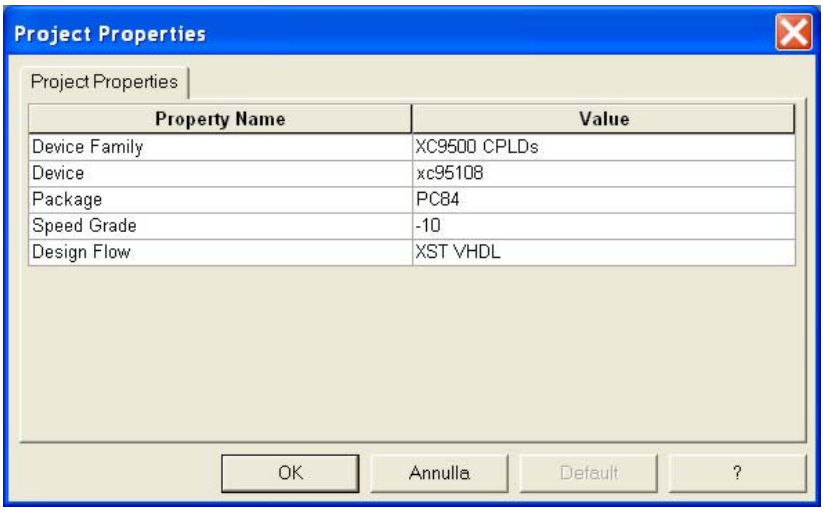
PROGRAMMAZIONE DELLA CPLD

La CPLD in questione e' stata programmata utilizzando l'interfaccia JTAG che prevede l'uso di un cavo parallelo (XILINX PARALLEL CABLE III) dotato di 6 connettori esterni, fornito direttamente dalla Xilinx :



Per arrivare al file di programmazione bisogna eseguire alcuni passi base con il software ISE WebPack :
Dal file 'definitivo.dia' creato con lo 'statecad' si genera il corrispondente listato VHDL agendo sul tasto "generate HDL".
Ora si puo' chiudere lo "statecad" e passare al programma "Xilinx Project Navigator" scegliendo "New project".
A questo punto verra' richiesto il nome del progetto, la directory di lavoro, il modello di FPGA/CPLD che si vuole utilizzare ed i suoi parametri principali.

Segue una schermata esempio :

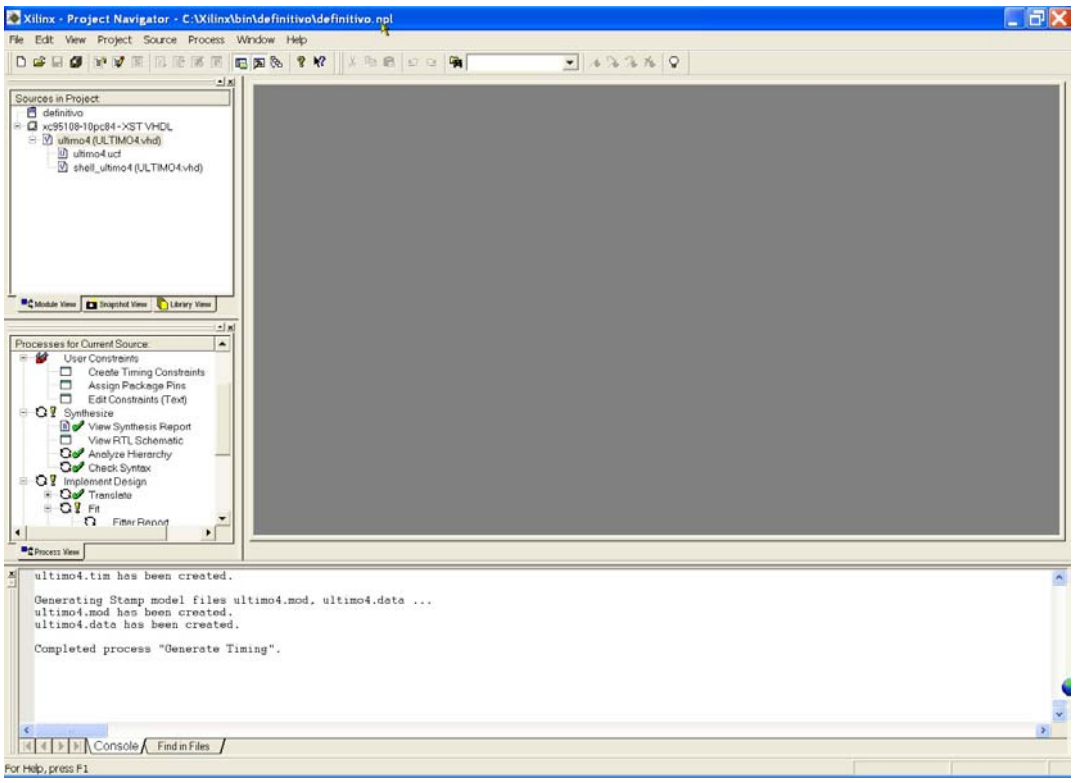


Per inserire il listato VHDL creato in precedenza si agisce su SOURCE→ADD SOURCE
E si indica il percorso di "definitivo.vhd".
In questa sezione e allo stesso modo si puo' aggiungere il file *.UCF che serve a definire la posizione dei pin desiderata.
Se non e' presente questo file, il compilatore ogni volta eseguira' un FIT nel modo e nel posto che a lui sembra piu' conveniente.
Segue un esempio di file *.UCF :

```
NET "ce" LOC = "P1";
NET "serialin" LOC = "P2";
…….
```

La struttura e' molto semplice e puo' essere cambiata andando sulla voce " assign package pins".

A questo punto si presenta una schermata del tipo seguente:



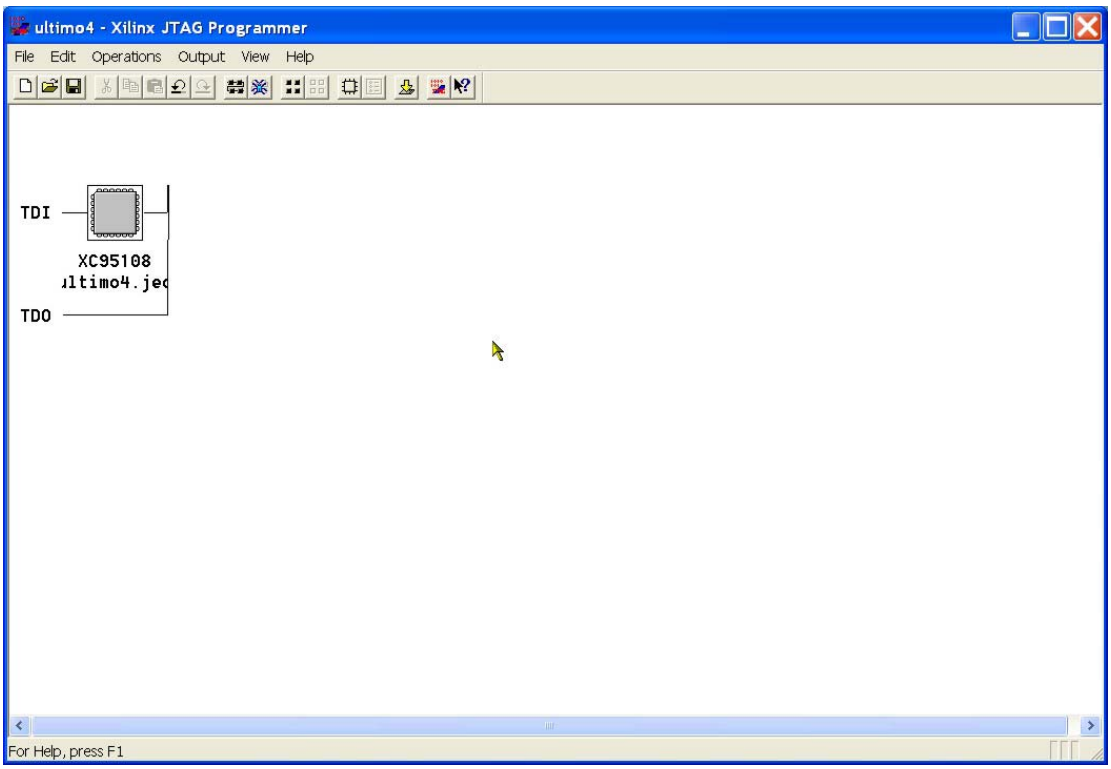
Ora e' sufficiente andare sulla voce "implemet design" → "run". In questo modo vengono eseguiti i seguenti passi partendo dal file VHDL unito al file UCF che descrive la posizione dei pins:

1) Sintesi
2) Implementazione (Traduzione + FIT + Generate timing)

Se tutto va a buon fine si puo' passare al passo successivo che e' la generazione del file di programmazione.
Questo file ha estensione *.JED e viene creato nella stessa directory e con lo stesso nome.

NB: a questo punto il FIT e' stato eseguito e cliccando su "view fitted design" si puo' vedere la "piantina" della nostra CPLD con le assegnazioni dei pins (post fit).

Per programmare la CPLD bisogna uscire e aprire il programma utility per la programmazione, ovvero "Xilinx JTAG programmer", sempre disponibile online sullo stesso sito.
Ci troveremo di fronte alla seguente schermata:



Per utilizzare il file appena creato, si va su EDIT →ADD DEVICE e si specifica il file *.JED desiderato.
Ora si va su OPERATION → PROGRAM e avra' luogo la programmazione della CPLD .

TEST DEL PROGETTO

Il circuito e' stato montato e testato prima su una basetta del tipo "millefori".
E' stato eseguita una piccola schedina adattatrice in vetronite dove far alloggiare la flash, in modo da prolungarne ed allargarne i contatti (essendo unicamente in formato smd, stagnarla ripetutamente sarebbe stato un problema).
Sulla basetta e' stato montato tutto l'hardware necessario, e' stato tarato come descritto in precedenza sezione per sezione, sono stati derivati i 6 pin di collegamento per la programmazione della CPLD (TDO, TCK, TDI, TMS, VCC, GND).

Il test e' risultato positivo, la qualita' dell'audio registrato e' risultata buona e il sistema in generale e' risultato stabile.

Si e' proceduto quindi allo sviluppo della scheda elettronica finale, usando il sofware ORCAD.

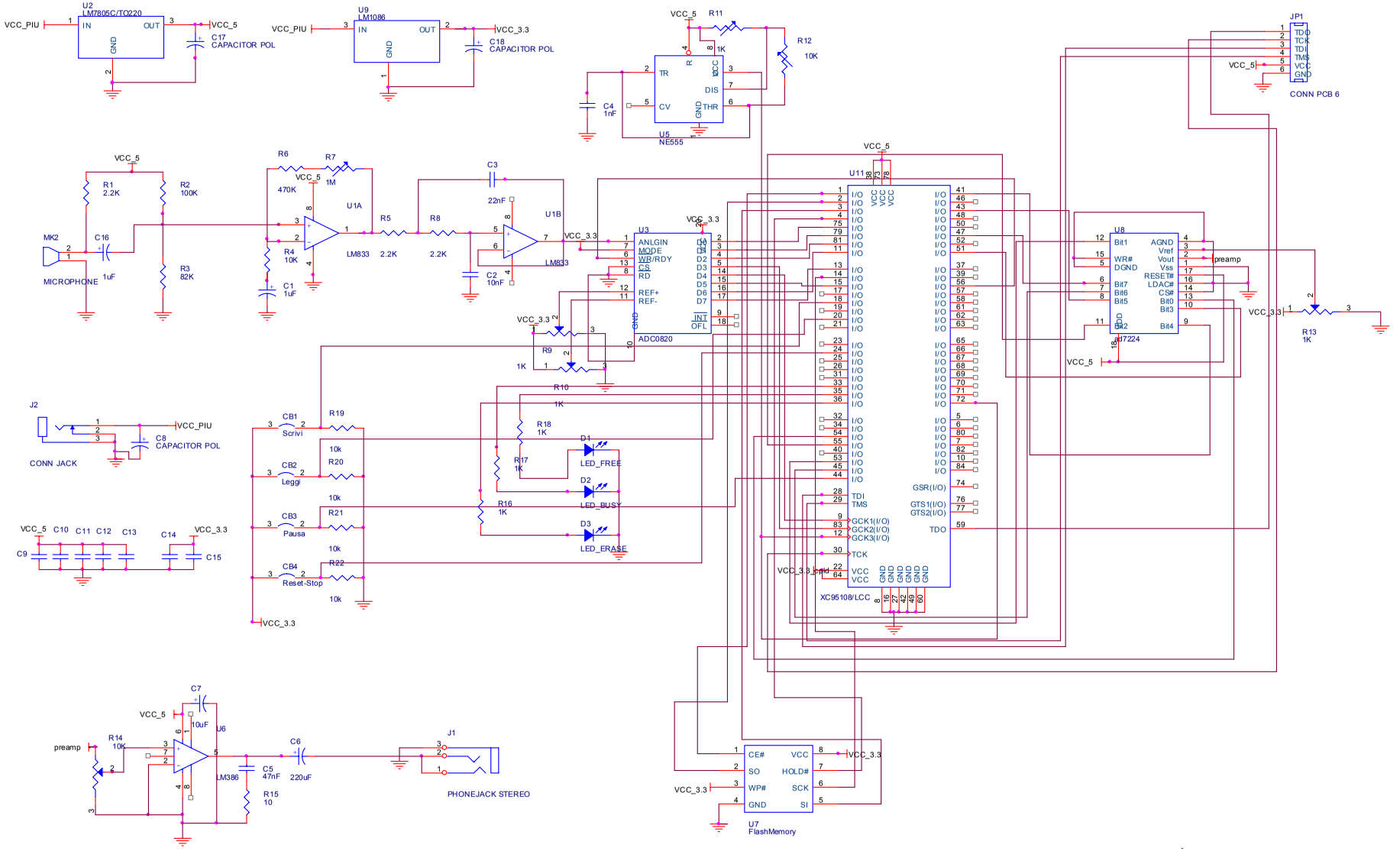
SVILUPPO DEL CIRCUITO CON ORCAD

Per implementare un circuito elettronico con Orcad (versione 9.1 quella usata da noi) e' necessario eseguire dei passi fondamentali usando diversi tools presenti nel pacchetto del programma.

1) Inserimento dello schema elettrico tramite programma CAPTURE.
2) Creazione di una netlist da poter usare nel programma LAYOUT.
3) Importazione della netlist in LAYOUT, posizionamento componenti, routing.

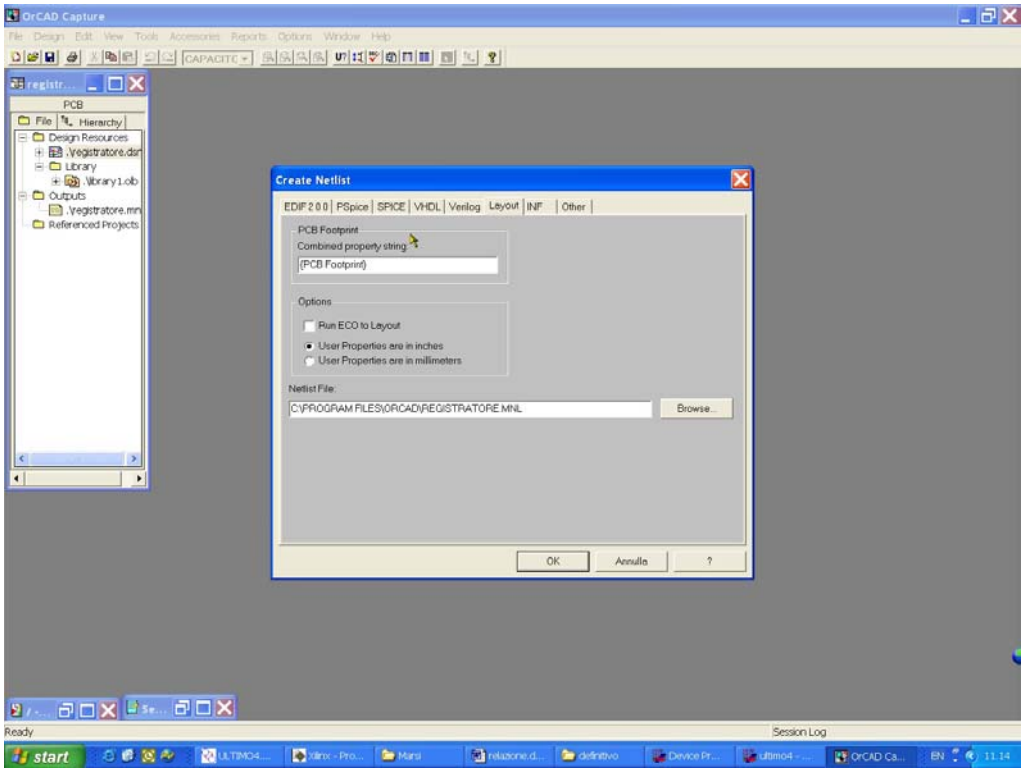
.CAPTURE : tramite questo programma e' possibile "disegnare" lo schematico, cioe' lo schema elettrico del circuito.
Nella pagina seguente e' stato riportato l' intero schema elettrico, dividendo il piu' possibile le varie sezioni.
Sono stati usati i riferimenti "VCC_5" e "VCC_3" per indicare le alimentazioni rispettivamente a 5V e 3.3V.
I condensatori 'isolati' C9÷C15 sono condensatori stabilizzatori di tensione che vanno piazzati nelle immediate vicinanze dei pin di alimentazione di ciascun integrato.
Il loro valore sperimentalmente puo' essere fissato a 100nF.

Nella pagina che segue e' stato riportato l'intero schematico del progetto, eseguito con ''capture'':



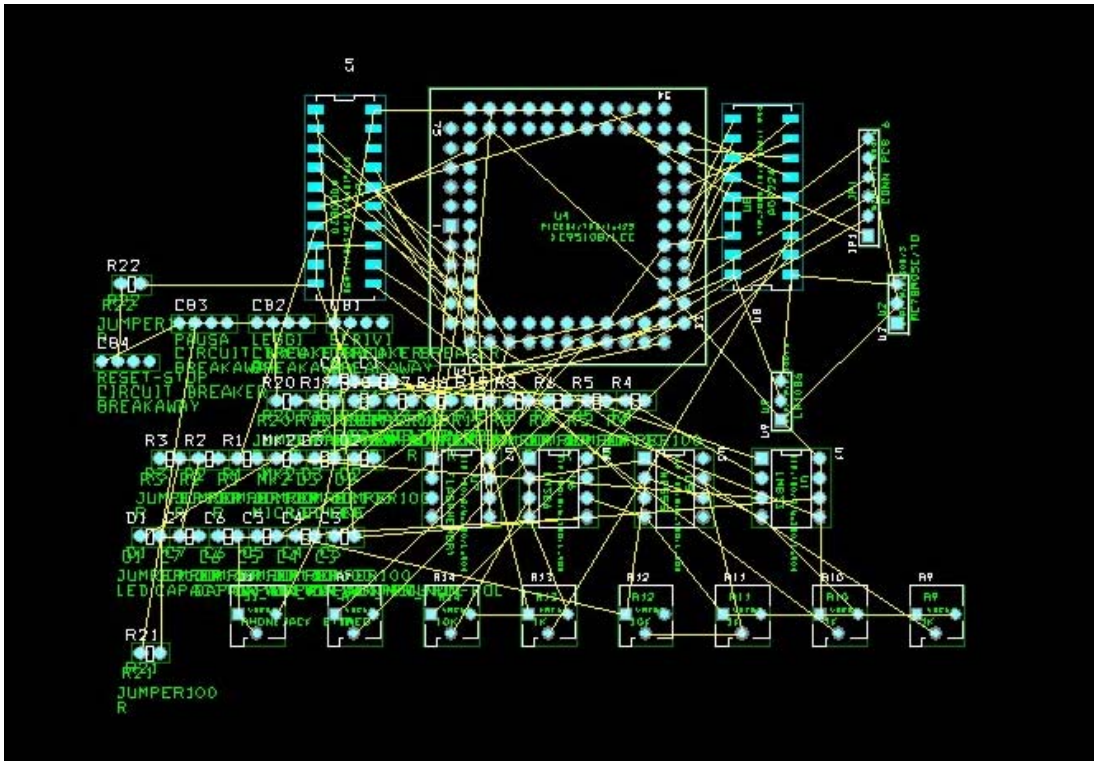
Title	<Title>	
Size: Custom	Document Number: <Doc>	Rev: <Rev Code>
Date: Tuesday, July 15, 2003	Sheet 1 of 1	

Il file della netlist si crea dallo schematico nel seguente modo:
Si va nella schermata principale di CAPTURE, si fa il seguente percorso: “TOOLS→CREATE NETLIST”, si seleziona il programma di destinazione, cioe’ LAYOUT e si preme OK.
Il file *.mnl viene cosi’ generato:



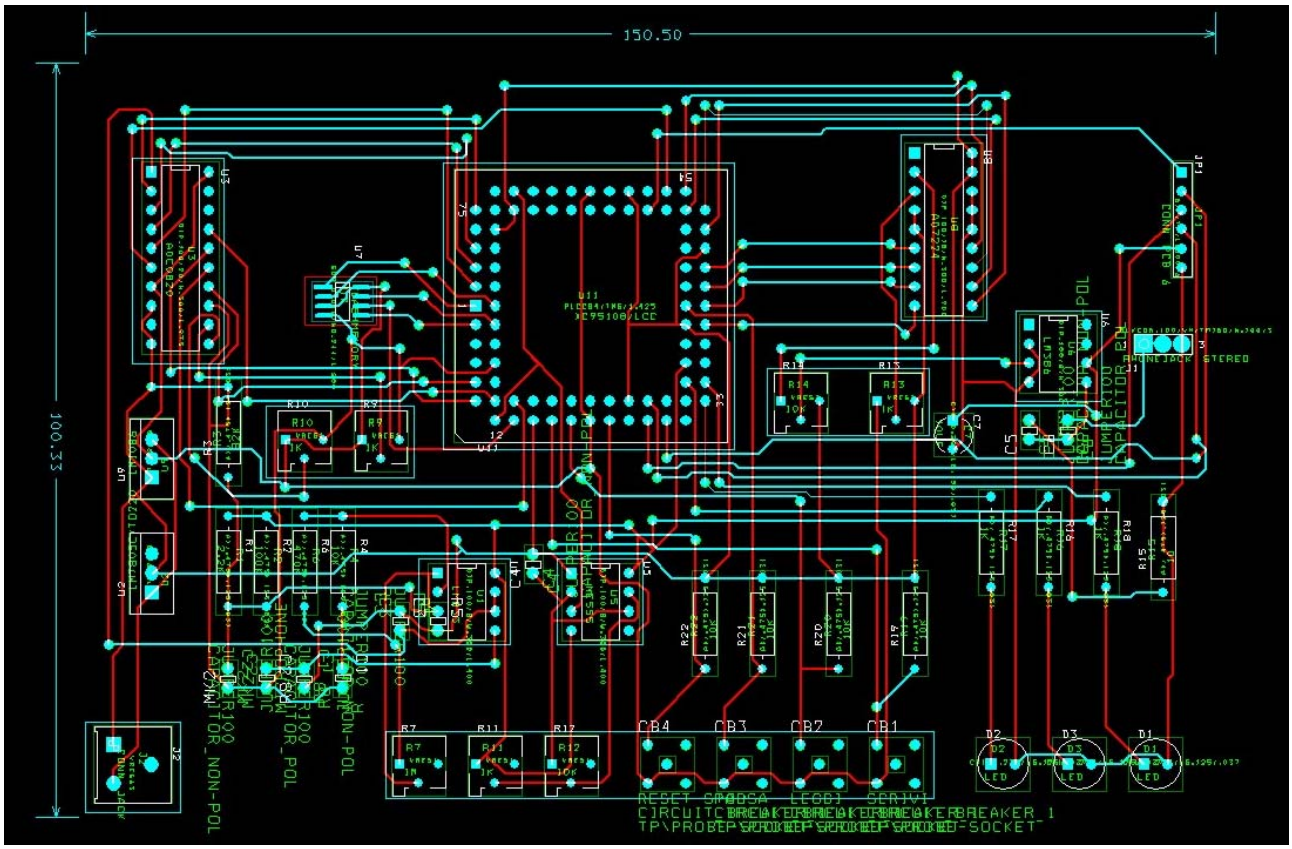
La netlist contiene la descrizione di tutte le connessioni tracciate nello schematico utilizzando i nomi delle reti, dei componenti e dei pin.
Usando l’opzione “Run ECO to layout” si permette al programma LAYOUT di intercomunicare con CAPTURE accorgendosi di eventuali modifiche al file *.mnl e apportando cosi’ automaticamente le dovute modifiche al suo interno.
Si passa ora al programma LAYOUT che permette lo sbroglio vero e proprio del circuito.
All’inizio di ogni progetto viene richiesta l’importazione del file *.mnl e i footprint dei singoli componenti.
I footprint sono dati ausiliari che descrivono per ogni componente il numero dei pin, la loro distanza e posizione, lo spazio occupato fisicamente dal package.

Quello che si presentera’ sara’ un “ammasso” di connessioni con componenti tutti vicini, come l’illustrazione che segue:



A questo punto si dispongono manualmente i componenti nel posto desiderato.
Dopo averlo fatto si specificano i dati principali della board, come per esempio il numero dei layer da trattare.
In questo caso e' stato impostato a 2 (TOP e BOTTOM).
Per fare cio' si va nel menu' "tools→layer→select from spreadsheet" e si imposta su "routing" i soli due layer interessati e annullando il resto.
Un'altra cosa importante e' definire gli ostacoli, ovvero le zone nei singoli layer dove non venga eseguito il ruoting.
Cio' e' importante perche' in questo caso risulterebbe impossibile (tramite saldatura a mano) eseguire saldature sul layer TOP sui piedini interni della CPLD.
Si e' scelto di non far eseguire routing superficialmente in tutta la CPLD e su ogni integrato, jack di connessione , alimentazione , cuffie e trimmer.
Per definire le zone proibite si usa l'apposito strumento "obstacle tool" delimitando l'area da proteggere e definendone il layer.
A questo punto si puo' fare una prova di autoroute, usando l'apposito comando "auto→autoroute→board".

Si otterra' una schermata del genere:

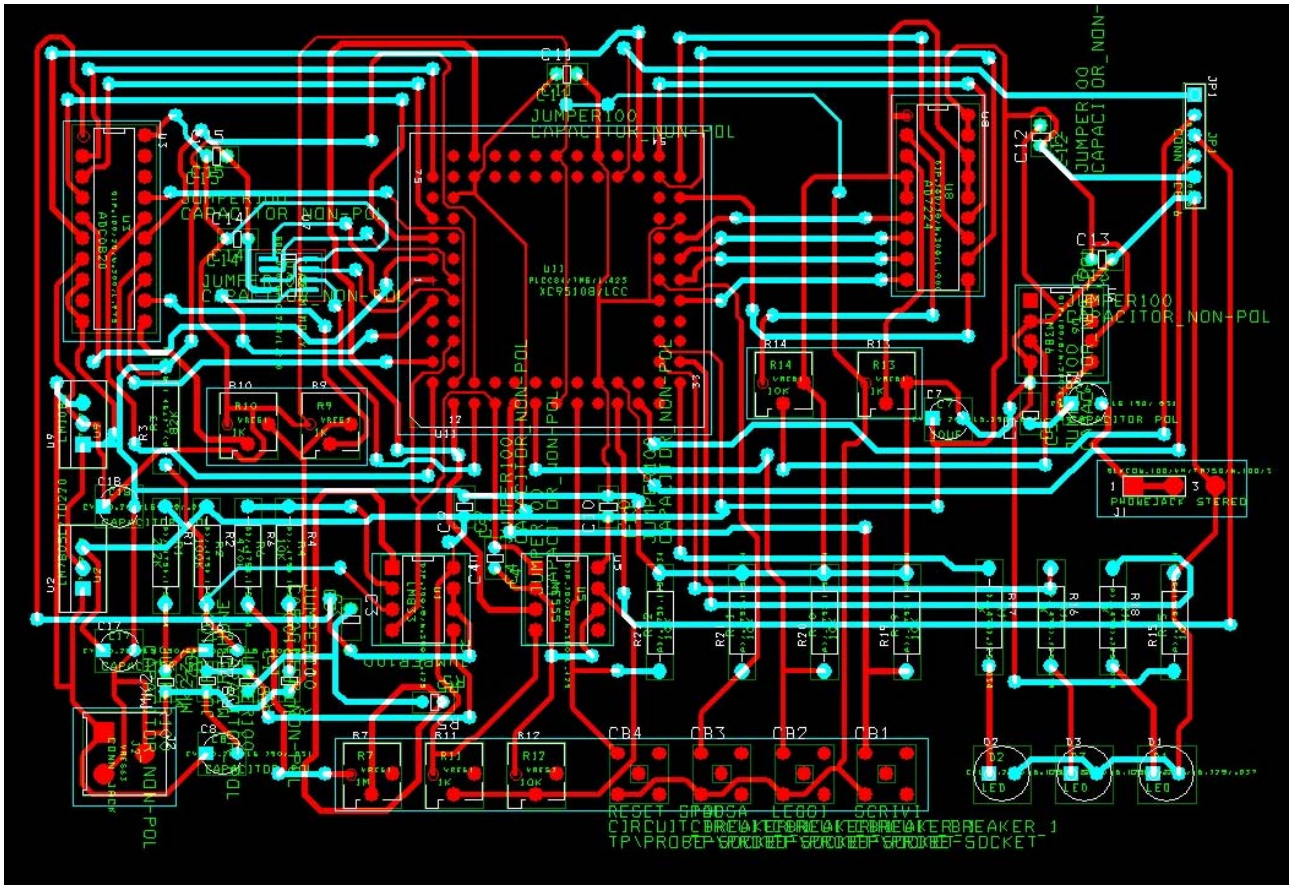


La board e' gia' stata dimensionata a 10x15cm usando il tool 'dimensions'.
Il layer TOP e' indicato in azzurro, quello BOTTOM in rosso.
Si puo' notare inoltre l'effetto di "obstacle tool".
Il layout ora e' quasi finito, ma si deve ancora tener conto delle dimensioni delle piste (nets) e delle piazzole (padstacks).
Di default sono impostate a valori molto piccoli, aumentarle troppo comporta un routing a volte impossibile (per esempio tra i pin degli integrati).
Si e' deciso di mantenere lo spessore di default delle piste per quelle passanti internamente alla CPLD, mentre di ingrossare le altre. Stesso discorso per le piazzole.

Per variare le dimensioni delle piazzole si fa riferimento allo spreadsheet localizzato in "tools→padstack→select from spreadsheet" dove si puo' scegliere la dimensione per ogni singolo padstack.

In modo simile si procede alla variazione delle dimensioni delle singole piste, usando l'utility "tools→net→select from spreadsheet" .
Per variare le dimensioni dei singoli segmenti si usa invece il tool "edit segment mode" che permette di ridefinire manualmente il ''routing'' dei singoli segmenti.
Si puo' inoltre procedere ad eliminare i padstack sul layer TOP della CPLD e degli integrati, visto che non sono soggetti a routine e quindi neppure a saldature.

Il routing “finale” risulta il seguente:

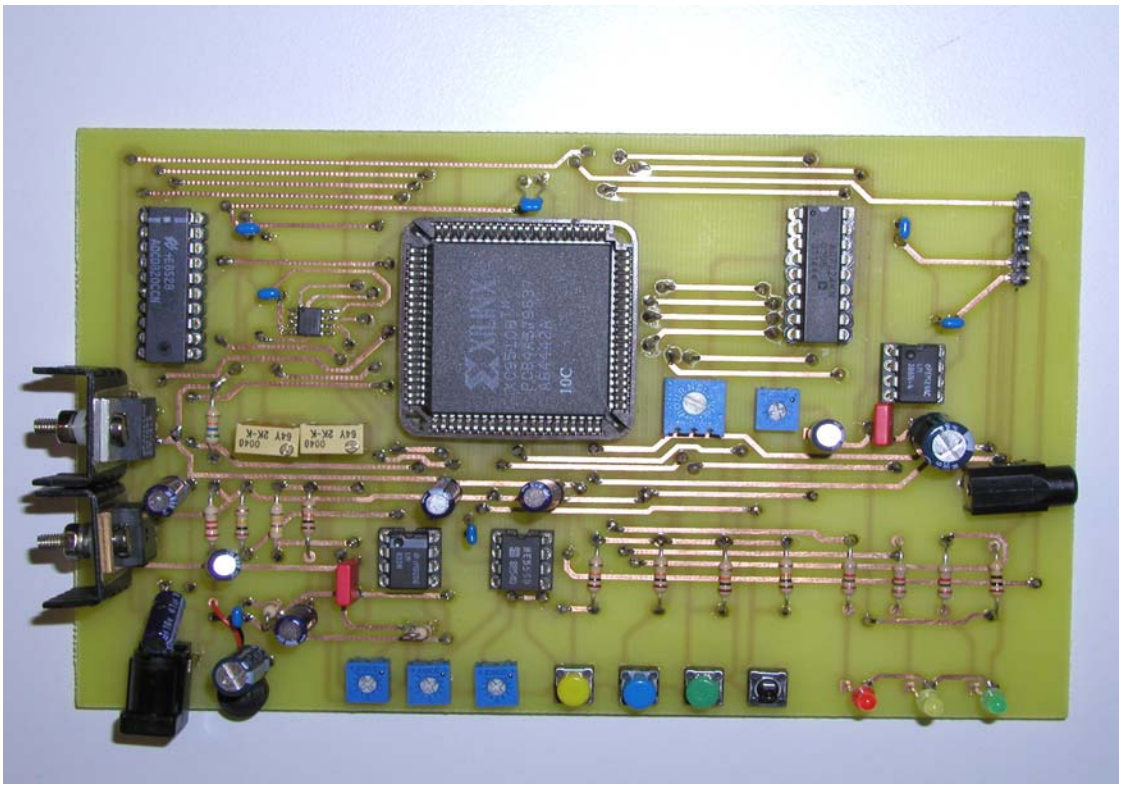


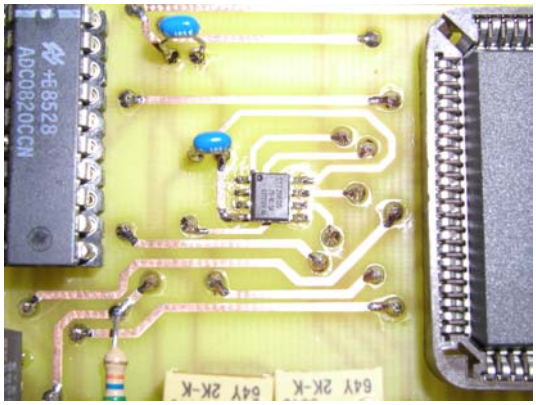
REALIZZAZIONE DELLO STAMPATO:

Una volta finito il layout bisogna provvedere a trasferirlo su basetta in vetronite.
Le basette a nostra disposizione sono quelle pre-sensibilizzate da utilizzare con la tecnica della fotoincisione.
Tale tecnica funziona nella seguente maniera: la basetta ha superficialmente uno strato di photoresist, un particolare prodotto che illuminato dai raggi UV diventa ‘’attaccabile’’ dal cloruro ferrico, l’acido comunemente usato per corrodere il rame. Nei punti dove invece i raggi UV non colpiscono il photoresist, l’acido non corrode nulla.
Si procede quindi a stampare su un lucido il layout precedentemente fatto (in questo caso sono stati stampati due lucidi, uno per ogni layer), lo si appoggia sopra la basetta, si espone il tutto ai raggi UV per alcuni minuti : i raggi colpiranno esclusivamente le zone dove non scorrono piste.
Successivamente si passa per alcuni minuti la basetta nella soda caustica diluita in acqua, che ha il compito di asportare lo strato di photoresist dalle zone colpite dalla luce.
Per finire si immerge la basetta nel cloruro ferrico finche’ il rame in eccesso non risulta completamente corroso.

A questo punto si lava la basetta in acqua e si possono praticare i fori, che risultano facilitati dalla presenza dei DRILL aggiunti in fase di stampa, ovvero dei ‘’forellini’’ centrali alle singole piazzole che permettono alla punta del trapano di forare esattamente nel punto voluto.
Fatto questo, con l’aiuto di una carta vetrata finissima si ‘’leviga’’ l’intera basetta, facendo comparire il rame che stava sotto lo strato di photoresist indurito.
A questo punto siamo pronti per effettuare tutte le saldature necessarie.

Si e’ cominciato con la saldatura dei ‘’vias’’ ovvero dei punti di interconnessione tra i due layer, successivamente sono stati saldati zoccoli e connettori, per poi finire con la saldatura dei componenti, lasciando per ultima la saldatura superficiale della flash, operazione molto delicata da effettuare a mano.
Si riporta qui di seguito l’immagine del circuito montato e in particolare l’ingrandimento della flash memory:





ANALISI DEI COSTI E CONCLUSIONI

Nella seguente tabella sono riportati i costi dei singoli componenti usati per la costruzione di questa scheda. I prezzi espressi sono in euro.

Descrizione	Quantita'	Prezzo unitario	Totale
Resistenze 1/4W 5%	15	0,012	0,18
Trimmer orizzontali 1K/10K	7	0,30	2,1
Condensatori policarbonato 100nF/1nF	10	0,17	1,7
Condensatore policarbonato 47nF	1	0,20	0,20
Condensatore elettrolitico 47uF/1uF	5	0,15	0,75
Condensatore elettrolitico 220uF	1	0,30	0,3
Regolatore tensione LM7805	1	0,60	0,60
Regolatore tensione LM1086 3.3V	1	1,56	1,56
Pulsanti normalmente aperti	4	0,54	2,16
Zoccolo 20 pin	1	0,78	0,78
Zoccolo 18 pin	1	0,65	0,65
Zoccolo 8 pin	3	0,36	1,44
Alette raffreddamento	2	0,43	0,86
Led rosso 0.3	1	0,15	0,15
Led verde 0.3	1	0,23	0,23
Led giallo 0.3	1	0,23	0,23
Jack cuffie	1	0,80	0,8
Jack alimentazione	1	0,57	0,57
Microfono preamplificato	1	0,85	0,85
Convertitore ADC0820	1	9,72	9,72
Converitore AD7224	1	9,82	9,82
Operazionale LM386	1	1,82	1,82
Integrato NE555	1	0,49	0,49
Operazionale TL082	1	1,51	1,51
Flash memory SST25VF020	1	1,57	1,57
CPLD XC95108 PC84	1	39,00	39,00
Basetta presensibilizzata 10x16 cm doppia faccia	1	3,48	3,48

TOTALE	€ 83,52

Il costo e' abbastanza sostenuto se paragonato alle prestazioni svolte dall'apparecchiatura, soprattutto pensando che al giorno d'oggi esistono gia' integrati dei chip che per pochi euro e con solo un paio di componenti esterni offrono registrazioni audio di qualità superiore, per cui lo scopo di questo progetto vuole essere solo didattico, per entrare nel mondo delle CPLD/FPGA, imparando ad usare i software opportuni e ottenendo un'infarinatura di VHDL, nonche' sviluppando la circuiteria analogica necessaria.
L'apparecchiatura comunque presenta possibilita' di espansione semplicemente cambiando il modello di flash memory (attualmente la SST produce modelli fino a 4 Mbits), aumentando la frequenza di campionamento e magari implementando una compressione dei dati audio.

BIBLIOGRAFIA

. "Dispositivi e circuiti elettronici" vol. II, M. Gasparini – D. Mirri -CALDERINI-

DATASHEET

. Regolatore di tensione LM7805 (5V) : http://www.fairchildsemi.com/ds/LM/LM7805.pdf
. Regolatore di tensione LM1086 (3.3V) : http://www.national.com/ds/LM/LM1086.pdf
. Amplificatore operazionale TL082 : http://www.national.com/ds/TL/TL082.pdf
. Amplificatore audio LM386 : http://www.national.com/ds/LM/LM386.pdf
. Convertitore ADC0820 : http://www.national.com/ds/AD/ADC0820.pdf
. Convertitore AD7224 : http://www.analog.com/UploadedFiles/Data_Sheets/303660920ad7224.pdf
. CPLD Xilinx XC95108 : http://www.cse.cuhk.edu.hk/~khwong/www2/ceg3480/95108.pdf
. Memoria flash SST25VF020 : http://www.sst.com/downloads/datasheet/S71231.pdf
. Timer NE555 : www.fairchildsemi.com/ds/SA/SA555.pdf

SITI INTERNET

. Xilinx ISE Webpack : http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+WebPack