SVILUPPO SCHEDE DI CONTROLLO MEDIANTE MICROPROCESSORI AVR/ARM

RELATORE: *Prof.* Gaudenzio Meneghesso

LAUREANDO: Maurizio Bottaro

A.A. 2009 – 2010

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

RELAZIONE DI TIROCINIO

SVILUPPO SCHEDE DI CONTROLLO MEDIANTE MICROPROCESSORI AVR/ARM

RELATORE: *Prof.* Gaudenzio Meneghesso

LAUREANDO: Maurizio Bottaro

Padova, 20 Novembre 2009

Ai miei genitori, a mia sorella, a Valentina e agli amici tutti

SOMMARIO

Questa relazione nasce da un tirocinio svolto all'interno della ditta Micronova s.r.l., un'azienda che si occupa di elettronica industriale e sistemi a microprocessore. Le argomentazioni si concentrano sullo sviluppo di software per due diverse architetture di microcontrollore: AVR (RISC a 8 bit) e ARM (RISC a 32 bit).
Per rappresentare l'architettura AVR è stato scelto un ATMega32 utilizzato per la gestione di una scheda di controllo per stufe a pellet. Per l'architettura ARM un AT91SAM9263 (meglio conosciuto come ARM9) utilizzato per il controllo di un casco termo stimolatore per capelli.
Sono architetture simili a quella MIPS, studiata durante questi anni universitari. L'unica differenza è che è stato possibile aver un riscontro pratico della funzionalità e delle prestazioni dei microcontrollori a livello industriale.

Si procederà con una panoramica sui microprocessori, microcontrollori e sistemi embedded. Sarà analizzata l'esperienza lavorativa, sviluppando approfondimenti sulle architetture AVR e ARM.
Per quanto riguarda il progetto su ATMega32, saranno esposti i punti più interessanti, omettendo il codice in C; in quanto poco interessante e difficilmente comprensibile.
Per quanto concerne l'AT91, dopo una descrizione di esso, saranno spiegate le potenzialità di questo versatile integrato, lo sviluppo di esso e la realizzazione in Qt dell'interfaccia grafica mediante l'inserimento di parti salienti di codice in C++.

Capitolo 1
I MICROPROCESSORI

Un microprocessore, all'interno di un circuito elettrico, ha lo scopo di gestire tutte le sue periferiche funzioni tramite la CPU (Central Processing Unit) che è al suo interno.
Oramai i μP[1] sono utilizzati in moltissime applicazioni, sia domestiche che industriali.
Applicazioni molto comuni possono essere cellulari, palmari, personal computer, autovetture, catene di produzione industriale e molto altro ancora.
Molti anni fa, i calcolatori occupavano interi hangar, date le loro dimensioni enormi.
Infatti per realizzare una piccola parte di esso, per esempio una ALU, erano necessari moltissimi componenti discreti, come resistenze, condensatori, diodi e transistor.
Con l'avvento della tecnologia LSI (large scale integration) è stato possibile integrare una intera CPU all'interno di un μP.

1.1 ARCHITETTURA LSI – LARGE SCALE INTEGRATION

Con tale innovazione si è potuto ridurre il costo di produzione e, fatto non di poco conto, le dimensioni finali delle schede da mettere in produzione per un'eventuale distribuzione di massa.
I μP sono costruiti con transistor MOSFET, e tali transistor costituiscono ogni singola parte di un circuito integrato; per esempio registri, ALU, memorie flash addirittura integrate all'interno di uno stesso μP e moltissimi altri circuiti digitali.
La densità di integrazione e le prestazioni dei circuiti integrati hanno avuto una crescita stupefacente negli ultimi decenni.
Nel 1960 Gordon Moore, cofondatore di INTEL, predisse che il numero di transistor sarebbe aumentato esponenzialmente con il passare del tempo.
Tale osservazione è chiamata "LEGGE DI MOORE", osservazione che poi è diventata legge data la sua sincerità e applicabilità anche al giorno d'oggi, basti pensare che a partire dal 1970 ad oggi la densità delle memorie è aumentata di più di 1000 volte.
I μP sono cresciuti a livello di complessità e prestazioni ad un passo costante e prevedibile. Il loro continuo incremento di prestazioni non sta dando segni di rallentamento.
I processori più conosciuti sono gli INTEL, MOTOROLA, TEXAS INSTRUMENTS, AMD, FREESCALE/MOTOROLA e PHILIPS.
In base all'uso richiesto e alla mole dei segnali d'ingresso da gestire dobbiamo ponderare la scelta del nostro μP.
Per assurdo se si dovesse far lampeggiare un led non andremmo ad utilizzare un μP AMD a 64 bit con una frequenza di clock di 3.00 GHz, e magari anche dual core.
La scelta di un μP, per la produzione di schede su larga scala, va effettuata considerando i rapporti prezzo/prestazioni e prezzo/qualità.
I processori che costano meno non è detto che siano affidabili quanto un processore costoso e prestante di fascia alta.
Per applicazioni diverse dalla gestione complessa e pesante di una motherboard di un pc, sono utilizzati dei μP di tipo RISC e mips.
Tali processori sono utilizzati nella maggior parte dei casi su varianti del sistema operativo UNIX e su sistemi embedded.

[1] Altro modo per dire microprocessore. Per semplificare la lettura del manoscritto d'ora in poi sarà sempre usata questa sigla.

Un esempio sono i μP della ARM, all'inizio vennero utilizzati nei pc, ma col passare del tempo l'azienda decise di utilizzarli su sistemi embedded; dove erano richiesti μP a basso consumo, ma comunque con elevate prestazioni.

1.2 ARCHITETTURA CISC – COMPLEX INSTRUCTION SET COMPUTER

È l'architettura tradizionale, dove un aumento delle prestazioni corrisponde ad un aumento della complessità. Esempi tipici sono i μP utilizzati nel PC. Hanno però due limitazioni:

- Efficienza della ALU non è possibile migliorarla
- Velocità di accesso ai dati e/o istruzioni risolta solo con l'introduzione di cache di primo e secondo livello

Le istruzioni sono sempre più complicate, in modo da realizzare un μP che realizzi istruzioni paragonabili ai linguaggi di alto livello. Solo il 10% del set di istruzioni completo è effettivamente utilizzato. La realizzazione pratica, richiede molto più silicio rispetto ad architetture MIPS o RISC, ma rispetto a queste ultime diminuisce la difficoltà di progetto del compilatore e del sistema operativo, in quanto il μP utilizza istruzioni complesse (pseudo-istruzioni).
Ma una cosa in particolare accomuna la due architetture CISC e RISC. I μP CISC traducono le istruzioni in formato RISC in modo che la CPU le elabori come un RISC classico. Infatti non esiste alcun μP in grado di gestire e tradurre istruzioni complesse scritte in codice ad alto livello, l'istruzione è necessariamente scomposta in istruzioni a loro volta più semplici. La struttura di elaborazione è sempre più semplice man mano che ci si avvicina al core del μP, quindi sezioni che permettono elaborazioni sempre più rapide.

1.3 ARCHITETTURA RISC – REDUCED INSTRUCTION SET COMPUTER

È un set di istruzioni ridotto per eseguire operazioni semplici in tempi simili.
Questo approccio di programmazione fa si che vengano messe da parte le istruzioni più complesse e vengano sostituite da istruzioni molto più semplici e basilari.
In una macchina di questo tipo le uniche operazioni che permettono di accedere alla memoria sono quelle di load e di store, tutte le altre utilizzano registri.
Infatti questo tipo di programmazione viene definito "programmazione load/store".
Un codice di questo tipo, step to step, permette di realizzare programmi molto veloci e soprattutto ottimizzati.
L'unica pecca è la lunghezza del codice, confrontando una somma con Java o con C salta subito all'occhio la differenza della sintassi.
Java e C sono molto più sintattici, mentre in linguaggio RISC (tipo assembly) per eseguire una somma deve fare svariati passaggi tra registri utilizzando appunto le istruzioni di load/store.
All'inizio dell'industria dell'informatica i compilatori non esistevano e i programmatori scrivevano direttamente in codice macchina o in assembly.
Fu così che si decise di inserire nell'instruction set del μP istruzioni anche molto complesse per simulare le funzioni ad alto livello dei linguaggi di programmazione direttamente nei processori.
Facendo così si potevano realizzare programmi molto compatti, riducendo così l'occupazione della memoria programma. Ulteriore conseguenza era la riduzione dei tempi di caricamento dei programmi e quindi vi era un incremento di prestazioni.
L'architettura RISC ha dalla sua parte la velocità di esecuzione del programma ma lo svantaggio è l'occupazione di memoria da parte del codice, pur avendo integrato istruzioni complesse all'interno della CPU.

1.4 ARCHITETTURA MIPS – MICROPROCESSOR WITHOUT INTERLOCKED PIPELINE STAGES

MIPS è l'acronimo di Microprocessor without Interlocked Pipeline[2] Stages. È una architettura di tipo RISC di tipo ISA (Instruction Set Architecture). Questo tipo di μP è utilizzato per molte applicazioni di tipo embedded, per esempio router e videogiochi portatili. Sono disponibili versioni sia a 32 che a 64 bit, con estensioni per calcoli avanzati come applicazioni 3D che permettono di eseguire calcoli a 32 e a 64 bit in virgola mobile.

Lo sviluppo di questa architettura è incentrato sul potenziamento delle performance tramite l'ottimizzazione dell'utilizzo delle pipeline. Un'operazione gravosa era quella della divisione, in quanto si doveva attendere diverso tempo per far scorrere i dati all'interno della pipeline. Una soluzione era quella di usare una serie di connessioni (interlocks) che permettevano agli stadi della pipeline di indicare che erano impegnati nell'elaborazione, in modo da stoppare il flusso dei dati verso gli altri stadi, mettendoli così in uno stato di hide. Facendo così si perdeva del tempo, in quanto bisognava comunicare a tutti i moduli della CPU la situazione di elaborazione della pipeline. Per risolvere il problema, ogni istruzione del set è stata realizzata in modo da essere eseguita in un solo ciclo, eliminando la necessità di introdurre interlocks.
Il risultato di questa modifica comportò l'incremento della potenza di calcolo, anche in caso di istruzioni di divisione e moltiplicazione. L'architettura MIPS può comunque essere considerata una semplificazione dell'architettura RISC.
I primi esemplari risalgono al 1985, con la produzione dell'R2000. Strutture a 64 bit risalgono al 1991, con la produzione dell'R4000.

[2] Tecnologia utilizzata dai microprocessori per incrementare la quantità di istruzioni eseguite in una data quantità di tempo. Vi è un aumento della complessità circuitale ma il tutto è compensato da una maggiore velocità di esecuzione.

Capitolo 2
SISTEMI EMBEDDED

Un sistema embedded come dice la parola stessa è un sistema inglobato. Tali μP sono utilizzati per realizzare delle piattaforme ad hoc.
Sono integrati nel sistema e permettono di gestirne tutte le sua funzionalità; si parte dall'acquisizione dei segnali in ingresso, alla loro elaborazione e si arriva a generare l'uscita che gestirà eventuali carichi collegati al sistema.
Rispetto a sistemi di uso comune come i pc, i sistemi embedded sono progettati per eseguire solo un determinato tipo di applicazione; in accordo con l'hardware connesso alla scheda per la trattazione dei segnali.
In questo modo si riduce considerevolmente l'hardware occupato sulla scheda e anche il costo di fabbricazione.
Infatti l'essere μP per applicazioni abbastanza limitate, li rende molto economici e molto più semplici da inserire all'interno di un circuito elettrico rispetto a sistemi general purpose tipo i μP dei pc. L'esecuzione del software avviene in tempo reale per permettere un controllo deterministico dei tempo di esecuzione.

Di seguito si citano alcuni esempi di sistemi embedded:

- Sportelli Bancomat e apparecchi POS.
- Telefoni cellulari.
- Sistemi di automazione casalinghi come termostati, condizionatori e altri sistemi di monitoraggio della sicurezza.
- Elettrodomestici come forni a microonde, lavatrici, apparecchi televisivi, lettori o scrittori di DVD.
- I PLC (Programmable Logic Controller) utilizzati per l'automazione industriale.
- Consolle per videogiochi fisse e portatili.

Da tali esempi si può intuire che i sistemi embedded spaziano dai più comuni apparecchi come lettori MP3 e cellulari ai controlli in tempo reale.
Tali sistemi sono molto utilizzati per eseguire azioni ripetitive ma ad un costo contenuto.
Talvolta è necessario che i sistemi embedded abbiano delle prestazioni minime per l'esecuzione in tempo reale di alcune funzioni. Un esempio molto pratico è quello di fare in modo che siano eseguite certe istruzioni ad una data frequenza di clock finché il sistema è impegnato a fare delle altre operazioni critiche.
Queste funzionalità sono garantite con una apposita combinazione hardware/software.
Utilizzando algoritmi poco complessi e snelli si può riuscire molto bene ad impegnare al minimo le risorse del μP del nostro sistema.
Un esempio abbastanza tangibile può essere una comunissima scheda di acquisizione di flusso audio/video. Infatti il μP si interessa solamente di catturare gli interrupt e di indirizzare al posto giusto le informazioni; mentre l'acquisizione audio/video (campionamento degli ingressi), viene effettuata da appositi circuiti integrati dedicati.
Per questo motivo l'architettura di un sistema embedded risulta essere molto più semplificata rispetto ad un comune pc che deve eseguire le stesse operazioni ma utilizzando una unica CPU.
La progettazione dei sistemi dedicati dipende anche dalla loro distribuzione alla collettività.
Se la tiratura è limitata, come nel caso di macchinari specifici e costosissimi come apparati elettromedicali che si possono permettere solo i migliori ospedali, questi sistemi sono realizzati

con l'impiego del miglior hardware presente nel mercato; rinunciando al risparmio e garantendo una qualità senza confronti.
Per tirature elevate come cellulari e walkman, anche la scelta del più comune condensatore elettrolitico, un modulo di memoria RAM e il μP stesso, sono di fondamentale importanza.
Il risparmio anche di un solo centesimo di euro è importante.
Si pensi alla produzione di un milione di walkman. Se si risparmia un centesimo di Euro per ogni memoria ram integrata nell'apparecchio, si arrivano a risparmiare diecimila euro in tutto; rispetto ad un memoria che se la usiamo ci fa spendere diecimila euro in più.
Per la grande maggioranza dei sistemi embedded non si parla di software di gestione, bensì di firmware.
Tali firmware sono progettati per durare nel tempo e non causare errori, quindi durante la fase di stesura del codice si presta molta attenzione.
Nei sistemi embedded gli hard disk non vengono quasi mai usati, al loro posto ci sono delle memorie flash ad architettura NAND o NOR. Esse sono insensibili alle vibrazioni e a possibili movimenti molesti durante l'uso della macchina.
C'è la necessità di costruire macchine robuste che, anche nel caso di ricezione di dati sbagliati (satelliti mandati nello spazio), sappiano ricostruire il codice errato ricevuto con l'utilizzo di appositi algoritmi ed eventualmente, nella peggiore delle ipotesi, resettarsi.
Per il soddisfacimento di questa specifica i μP sono dotati di un **WATCHDOG TIMER**, che ripristina il processore se il programma scritto in essi non azzera con una stabilita periodicità un timer interno del componente.

2.1 PROGETTO DI SISTEMI EMBEDDED

Per i sistemi embedded ad alto volume di produzione ci si sta spostando sempre più verso i sistem on a chip (SOC). I SOC racchiudono, in un singolo circuito integrato di tipo ASIC, tutte le periferiche (USB, RS232/RS458, DSP, FLASH, CONVERTITORI AD/DA e viceversa ecc.) e la CPU stessa.
Talvolta il produttore di μP fornisce un board support package (BSP) per semplificare il supporto e integrazione tra il software sviluppato ad hoc, l'ambiente operativo sottostante e l'hardware.

2.2 INTERFACCE UTENTE

Un buon sistema integrato necessita di una semplice, ma non per questo poco efficace, interazione con l'utente finale.
Il sistema deve poter comunicare il suo stato operativo, saper ricevere informazioni direttamente dall'utente e comunicare eventuali errori.
Per fare queste operazioni si passa dai più comuni pulsanti per selezione dei vari menù e led che rendono un'informazione visiva delle operazioni svolte, ai sistemi più evoluti quali i touch screen.
È ovvio che un sistema economico non avrà mai a bordo un touch screen, ma basti pensare che se non esistessero, non si potrebbe fare tutto ciò che si fa ora nel campo delle apparecchiature elettromedicali.
Per quanto riguarda le soluzioni più minimaliste si utilizzano dei comunissimi led.
Si passa dai classici led rosso, giallo e verde alle più moderne tonalità blu, bianco ecc.
Le colorazioni classiche permettono di rappresentare, per esempio, delle situazioni standard nel funzionamento di una macchina (il classico semaforo):

- LED ROSSO => stato di pericolo
- LED GIALLO => stato di allerta
- LED VERDE => stato di funzionamento regolare

Rispetto alle nuove tonalità di colore introdotte da poco grazie agli innovativi drogaggi del silicio, le colorazioni standard costano e consumano meno.
Ai capi di un led normale si ha una caduta di tensione di circa 1,5 V mentre sulle nuove giunzioni si arriva anche ai 3 V.
Può essere però molto comodo visualizzare la fase di starting della macchina.
Un esempio molto palpabile ci viene fornito dal pc, esso infatti avviandosi ci fa vedere lo stato della RAM e lo stato delle periferiche ad esso connesso, come hard disk, lettori ottici e anche periferiche USB.
Nel caso di un qualsiasi malfunzionamento, l'utente è avvisato con un messaggio visivo o da un messaggio sonoro con una precisa "melodia" suonata dalla scheda madre dipendentemente dall'errore verificatosi.
Nei sistemi embedded potrebbe essere molto utile una funzionalità simile.
Se abbiamo a che fare con un macchinario di notevoli dimensioni, potremmo avere un vantaggio smisurato nel sapere già cosa non funziona nella macchina; potendo così intervenire prima che la stessa venga avviata e possa creare danni anche irreversibili a cose e/o persone.
Resta il fatto che un'interfaccia grafica, oltre che essere semplice ed intuitiva, comporta uno svantaggio considerevole a livello progettuale.
La progettazione richiede molto tempo, a volte anche un anno in più. Quindi anche se sicuramente procura un altro impatto rispetto ad una macchina con un LCD[3] spartano con solo il testo visualizzato, bisogna fare i conti anche con il tempo/denaro e soprattutto con la concorrenza.
Bisogna stare anche molto attenti alla completezza dei menù. Si ricorda la semplicità e l'immediatezza, per realizzare le interfacce utente bisogna sempre immedesimarsi al livello dell'utente che è molto probabile che non abbia la conoscenza tecnica del progettista; quindi mai dare niente per scontato.
Se si rendono il più chiaro possibile i metodi di funzionamento del macchinario, per un'eventuale traduzione dei menù (dato il nostro attuale contesto multietnico), il tutto risulterà più semplice e sicuramente meno macchinoso e allo stesso tempo più sicuro e puntualizzato.
Normalmente il progetto finito viene fatto provare al cliente, che deciderà eventualmente se cambiare qualcosa.
Talvolta però si manifestano dei ritardi, causati dall'incapacità del cliente di sindacare eventuali parti ottimizzabili della macchina. Ciò causa ritardi e perdita di tempo/denaro.

2.3 STRUMENTI DI SVILUPPO

Come per altri software, i progettisti di sistemi embedded utilizzano compilatori, assembler e debugger per sviluppare i software relativi al sistema. Tuttavia possono usare anche alcuni programmi più specifici.
Un in-circuit emulator (ICE) è un dispositivo hardware che sostituisce o si interfaccia con il microprocessore, ed offre funzionalità per caricare velocemente e effettuare il debugging di codice di prova all'interno del sistema.
Per velocizzare e rendere economica la diagnostica e il debugging, specie su sistemi prodotti su larga scala, viene integrata, a livello di SoC[4], microcontroller o CPU, l'interfaccia JTAG (alias IEEE 1149.1), uno standard di interfacciamento semplice ed economico atto a sospendere il normale funzionamento del processo e interrogarne le fasi tramite collegamento a un personal computer.

[3] Liquid Crystal Display
[4] System on Chip.

Alcune utility aggiungono un controllo di ridondanza (checksum) o un Cyclic redundancy check (CRC) al programma, in modo da permettere al sistema embedded di controllare la validità del programma.
Per sistemi che utilizzano Digital Signal Processor (DSP), i progettisti possono usare uno strumento algebrico come MathCad o Mathematica per simularne la matematica.
Compilatori e linker specifici possono essere utilizzati per migliorare l'ottimizzazione di hardware particolari.
Un sistema embedded può avere un proprio linguaggio specifico o programma di sviluppo, oppure offrire miglioramenti ad un linguaggio esistente.
Data la complessità crescente, non è raro che il produttore dell'hardware fornisca un BSP (o Board Support Package) per semplificare il supporto e integrazione tra il software sviluppato ad-hoc, l'ambiente operativo sottostante e l'hardware.
I programmi per la generazione del software possono avere provenienza diversa:

- Compagnie produttrici di software specializzate nel mercato dei sistemi embedded
- Possono essere tool provenienti dal progetto GNU (si veda anche cross compiler)
- Talvolta possono essere utilizzati tool di sviluppo per personal computer se il processore embedded è molto simile ad un comune processore per PC.

2.4 AUTO-TESTING INTERNO

Gran parte dei sistemi embedded hanno capacità native di auto-verifica delle proprie funzionalità. Tipicamente in caso di procedure in loop o subentrando in situazioni di deadlock[5], intervengono dei meccanismi di 'protezione' chiamati watchdog. Ci sono diversi tipi principali di verifiche, divise in base alla funzione o componente controllati:

- ***Verifica Calcolatore:*** CPU, RAM e memoria programmabile. Spesso si effettua una volta all'accensione. Nei sistemi di importanza critica si effettua anche periodicamente o continuamente.
- ***Verifica Periferiche:*** Simula ingressi e misura uscite. Un sorprendente numero di sistemi di comunicazione, analogici o di controllo possono fare queste verifiche, spesso a costo molto basso.
- ***Verifica Alimentazione:*** Solitamente misura ogni linea di potenza, e può controllare anche qual è l'ingresso (batterie o rete). Le alimentazioni sono spesso sfruttate al massimo, con poco margine di scarto.
- ***Verifica Comunicazione:*** Verifica la ricezione di semplici messaggi da parte delle altre unità connesse. Internet, ad esempio, usa il messaggio "ping".
- ***Verifica Cablaggi:*** Solitamente usano un cavo sistemato a serpentina tra punti rappresentativi dei cavi che devono essere collegati. I sistemi di comunicazione sincroni, come la telefonia, spesso usano "sync test" a questo scopo. Le verifiche dei cavi sono economiche, e estremamente utili quando l'unità ha dei connettori.
- ***Verifica Attrezzaggio:*** Spesso un sistema deve essere regolato quando viene installato. Questa verifica fornisce indicazioni alla persona che installa il sistema.
- ***Verifica Consumi:*** Misura le risorse che il sistema utilizza, e avvisa quando le quantità sono basse. L'esempio più comune è la lancetta della benzina di un'auto. Gli esempi più complessi possono essere i sistemi di analisi medica automatica che gestiscono inventari di reagenti chimici.

[5] È una situazione in cui due o più processi si bloccano a vicenda aspettando cje uno esegua una certa azione che serve all'altro e viceversa.

- ***Verifica Operativa:*** Misura varie cose che interessano all'utente lavorando sul sistema. Notare che si effettua mentre il sistema è in funzione. Esempi sono gli strumenti di navigazione aeronautici, il contachilometri di un'auto, le lucine di un disk-drive.
- ***Verifica di Sicurezza:*** Eseguita periodicamente secondo un' intervallo di sicurezza, assicura che il sistema sia ancora affidabile. La durata dell'intervallo è in genere appena inferiore al tempo minimo entro cui un malfunzionamento può causare danni.

2.5 COLLAUDI AUTOMATICI

I collaudi automatici costituiscono una parte molto importante nella realizzazione di un sistema a μP.
Tali collaudi sono detti automatici, poiché c'è una macchina che si occupa di testare il corretto funzionamento delle schede elettriche.
I sistemi di collaudo, in generale, si interessano di sollecitare gli ingressi della scheda con opportuni segnali. Facendo così, in base al segnale applicato in ingresso, è possibile verificare se l'uscita è corretta o meno.
Un tipico esempio possono essere delle luci che si accendono, all'arrivo di un evento, integrate nell'apparato di collaudo.
Per realizzare i collaudi automatici di solito vengono usati dei pc che, interfacciati appositamente con la scheda in test, offrono in tempo reale informazioni sullo stato della scheda in analisi.
Per schede più semplici, talvolta non è richiesto nemmeno l'uso del pc; infatti può essere sufficiente un collaudo indipendente da interfacce seriali, parallele o usb.
Questi controlli sono d'obbligo, in quanto tutte le schede devono rispondere alle specifiche ben precise stipulate durante la progettazione con il cliente.
L'hardware che gestisce il collaudo ha una parte molto importante nel test di affidabilità di una scheda, ma altrettanto importante è il contributo umano.
L'uomo, a differenza delle macchine, è in grado di rilevare particolari fondamentali durante questi collaudi.
Può capitare che ci siano dei componenti discreti o smd non montati in modo consono; per sistemare tale situazione l'unica soluzione è l'intervento di un buon saldatore combinato a dell'altrettanto buon stagno per la saldatura dei componenti.
È buona abitudine fare sempre un rapido controllo visivo di piste e componenti prima del collaudo, per evitare danni alla scheda, al collaudatore, al collaudo automatico stesso e all'hardware esterno collegato.
Quando una scheda passa il test, solitamente viene imballata per la consegna al cliente; in questo passo bisogna prestare molta attenzione all'adagiamento della scheda nell'imballo.
Può capitare che gestendo male queste operazioni che sembrano semplici, possa accadere che qualche componente si danneggi o addirittura si stacchi dalla scheda in seguito a un urto.
Bisogna quindi prestare molta attenzione e fare questi passaggi con molta calma, in modo da non vanificare il tempo perduto nel test della scheda.

Capitolo 3
ESPERIENZA LAVORATIVA

Vengono di seguito riportati due progetti eseguiti durante il tirocinio presso la Micronova s.r.l. di Peraga di Vigonza (PD).

3.1 PRESENTAZIONE AZIENDA

MICRONOVA viene fondata nel 1981 da Renzo Benetti, specializzandosi fin dall'inizio nella realizzazione di schede elettroniche.
Le prime produzioni riguardavano soprattutto schede elettroniche per semafori (settore di provenienza del fondatore, in cui raggiunsero ottimi risultati), per organi musicali (l'organo a 60.000 canne della Chrystal Cathedral di Los Angeles resta una delle realizzazioni più memorabili e prestigiose dei primi anni di attività) e per sistemi di sicurezza (adottati da circa 220 banche nel Triveneto).
Forte di questi primi successi, MICRONOVA inizia a specializzarsi in schede elettroniche per macchine da caffè (commissionate dalla San Marco), per distributori automatici per il settore della refrigerazione (distributori di bibite in lattina e gelati, banchi frigo, refrigeratori, frigoriferi da supermercato) e per impianti di condizionamento (ad esempio regolatori di velocità per impianti di condizionamento per hotel).
L'eterogeneità dei campi di applicazione delle schede elettroniche MICRONOVA ha consolidato nel tempo la grande esperienza nella ricerca di soluzione specifiche per la realizzazione di schede elettroniche completamente personalizzate.
MICRONOVA è oggi punto di riferimento per le principali aziende italiane ed europee che hanno fatto dell'innovazione il proprio punto di forza.

3.2 DIVISIONE AZIENDALE

- **RICERCA E SVILUPPO -** Le schede elettroniche progettate da MICRONOVA si basano sull'utilizzo di microprocessori prodotti da altre aziende: lo staff RICERCA & SVILUPPO è quindi costantemente impegnato nella ricerca di dispositivi con le più alte prestazioni tecnologiche e la massima affidabilità. Per garantire i migliori risultati i partner (Atmel, Freescale, Arm Core, Linux) sono scelti su scala internazionale tra i marchi più prestigiosi.

- **UFFICIO TECNICO -** Lo staff dell'Ufficio Tecnico si occupa di studiare risposte personalizzate tenendo sotto controllo i costi. L'ingegnerizzazione dei nuovi prodotti viene eseguita con CAD appropriati e nel rispetto delle normative internazionali vigenti in materia di sicurezza, IMQ, VDE, UL, ecc. Tutte le fasi di progettazione sono monitorabili attraverso una serie di documenti dettagliati che le accompagnano. Particolare attenzione viene riservata a quelle destinate all'installatore e all'utente finale.

- **CONTROLLO QUALITA' -**Per garantire un coefficiente di qualità sempre più elevato, MICRONOVA concentra la propria attenzione su tre argomenti chiave:
  1) componenti utilizzati
  2) sistemi di produzione e colladi
  3) post vendita

- **LABORATORIO EMC -**Il laboratorio è fornito di strumentazioni elettroniche e relative procedure necessarie per eseguire i test necessari all'autocertificazione CE delle varie apparecchiature.

- **UFFICIO ACQUISTI -** L'Ufficio Acquisti di MICRONOVA si occupa direttamente dell'acquisto di tutti i materiali: ottiene così i migliori prezzi, si mantiene aggiornato sugli avvicendamenti del mercato e intrattiene rapporti sempre più stretti con i maggiori distributori nazionali ed internazionali.

- **PRODUZIONE -** L'azienda ha attrezzato un reparto dedicato al montaggio dei componenti SMD. Le lavorazioni speciali per il finissaggio di alcuni prodotti vengono eseguite da tecnici specializzati.

- **REPARTO COLLAUDI -** Tutte le schede elettroniche realizzate da MICRONOVA sono sottoposte a severi controlli svolti durante tutte le fasi di produzione. In base alla tipologia di prodotto vengono eseguiti test specifici quali:
  1) BURN-IN-TEST, in apposita cella climatica
  2) TEST-IN-CIRCUIT
  3) controlli visivi mediante strumentazioni ottiche digitali
  4) prove di vita elettriche e meccaniche

  Tutto il processo di collaudo è svolto con modalità automatizzate attraverso apparecchiature progettate appositamente per ogni prodotto MICRONOVA che forniscono report precisi direttamente sui PC degli analisti per l'elaborazione finale.

3.3 FILOSOFIA AZIENDALE

MICRONOVA si propone di crescere continuando a produrre schede elettroniche semplici, per le quali non vengano aggiunte complicazioni inutili e superflue rispetto al compito che devono svolgere, facilmente gestibili dal cliente finale attraverso un'elettronica morbida e aperta, cioè in grado di risolvere esigenze anche nuove rispetto a quelle di partenza, trasformando i costi in investimenti di lungo periodo.

Il primo progetto constatava nella modifica del software di una scheda di gestione potenza di una stufa a pellet.
La scheda era controllata dal μC ATMEL MEGA32. Si richiedeva di gestire la memorizzazione della temperatura misurata da una sonda NTC all'interno dell' EEPROM del μC.
Il secondo progetto constatava nella realizzazione di un'interfaccia grafica per un casco termostimolatore professionale per parrucchieri e nella realizzazione di un collaudo automatico per lo stesso. La scheda è controllata da un μC ATMEL AT91.

Capitolo 4
PRIMO PROGETTO

Il primo progetto affrontato è servito per prendere mano con la programmazione in C nei sistemi embedded, quindi non più una programmazione con verifica diretta su piattaforma x86 (vedi MIPS e Java). La difficoltà stava essenzialmente nella comprensione e modifica del codice scritto per il funzionamento di potenza e interfaccia utente/macchina di una scheda di gestione per stufe a pellet. La problematica evidenziata dal tutor riguardava un problema di salvataggio delle impostazione dell'offset di temperatura al momento del collaudo prima della vendita della scheda. Era quindi richiesto di modificare il programma affinché i dati venissero salvati correttamente, in modo che alla riaccensione della scheda tutto risultasse salvato ancora in EEPROM. Questa modifica ha comportato anche l'inserimento di una procedura di verifica visiva nel software di collaudo.

4.1 SCHEDA IN ANALISI

Si tratta di un controllore per stufa a pellet, codificata in azienda come I023.
Il controllore è costituito da una scheda elettronica provvista di una serie di connettori che permettono il collegamento della scheda ai vari dispositivi principalmente costituiti da:

- Consolle di comando realizzata in varie versioni e con elevato grado di personalizzazione
- Sensori di temperatura
- Ventilatori
- Coclea
- Candeletta
- Allarmi
- Interfacce di comunicazione (RS232, Bluetooth)

Per la gestione a livello utente della scheda si collega alla scheda una consolle LCD o una consolle con display a sette segmenti.
C'è la possibilità di comandare la I023 attraverso un telecomando IR tramite un sensore posto sulla consolle grafica.

4.2 ATMEL ATMEGA32

Il controllore utilizzato per gestire la scheda è un ATMega32 a 8 bit prodotto da Atmel, con 32 kbytes di memoria programma.
Esso è basato su architettura RISC (Reduced Instruction Set Computer). Eseguendo istruzioni complesse, il μC[6] raggiunge prestazioni di 1 MIPS per MHz permettendo al progettista di ottimizzare il consumo di potenza nei confronti della velocità del μC.

[6] Altro modo per dire microcontrollore. Per semplificare la lettura del manoscritto d'ora in poi sarà sempre usata questa sigla.

4.3 SCHEMA A BLOCCHI

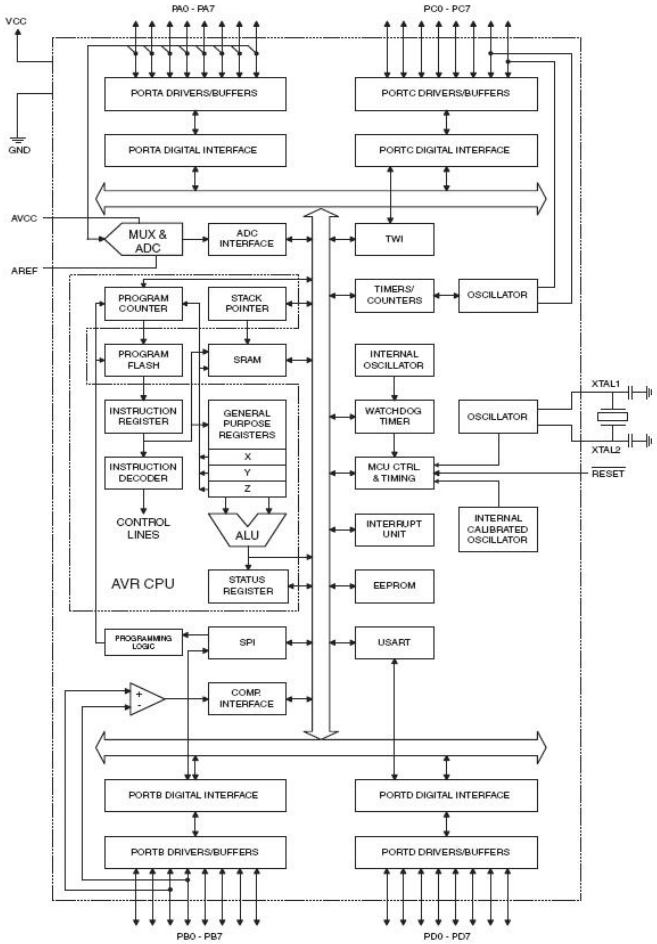

Figura 1 – Schema a blocchi architettura AVR.

4.4 PACKAGE ESTERNO

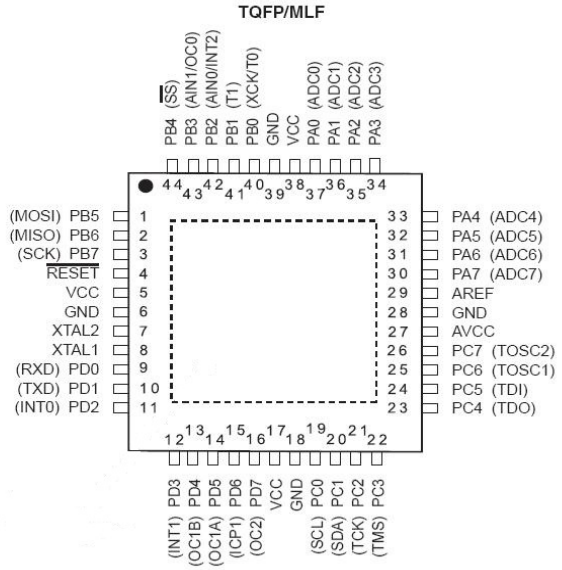

Figura 2 – Package esterno ATMega32.

Il nucleo AVR racchiude un ricco set di istruzioni e 32 registri per uso generico, i quali sono direttamente connessi alla ALU; permettendo a 2 registri indipendenti di essere utilizzati in ogni istruzione eseguita ad ogni ciclo di clock.
L'architettura utilizzata dall'ATMega32 è molto più veloce di una architettura CISC (x86).

Il μC è caratterizzato da:

- 32 Kb di memoria Flash programmabile, quindi per contenere il codice sorgente del programma di gestione della scheda elettronica
- 1024 byte di memoria interna EEPROM, molto comoda per salvare registri in modo permanente
- 2 Kb di memoria ram statica
- 32 linee di I/O ad uso generico
- 32 registri ad uso generico
- Interfaccia JTAG
- Supporto si debugging on-chip
- Seriale USART programmabile
- Tre timer/contatori interni
- Interrupt interni ed esterni
- Interfaccia seriale a due fili
- Convertitore ADC a 10 bit
- Watchdog Timer programmabile con oscillatore interno
- Porta seriale SPI
- Sei modi selezionabili via softaware per il risparmio energetico

L'ATMega32 può essere programmato in linguaggio C, con simulatori e con debugger, con emulatori "in-circuit" e con kit di valutazione.

4.5 ARCHITETTURA AVR

Per ottimizzare le prestazioni è implementata una architettura di tipo Harvard nella quale memoria dati e programma sono distinte. Quando un'istruzione sta per essere eseguita, l'istruzione seguente entra già in fase di fetch. Questo significa che il program counter viene incrementato ad ogni esecuzione di istruzione e l'occorrente per l'esecuzione dell'istruzione successiva viene preparato per ottimizzare i tempi di esecuzione e quindi ogni istruzione può essere eseguita ad ogni ciclo di clock.
Per spostare il contenuto dei registri all'interno degli accumulatori della ALU ed eseguire l'operazione richiesta è necessario attendere solo un ciclo di clock.
La ALU del μC permette di eseguire operazioni tra registri e anche tra un registro e un valore costante (immediate).
Sono contemplate istruzioni di salto condizionato o incondizionato.

La memoria programma, che è una memoria di FLASH, è divisa in due sezioni:

- Area di Boot
- Area di Applicazione

Entrambe queste aree possono essere protette da scrittura o lettura.

In caso di interrupt e chiamate a subroutine il valore del program counter viene salvato nello stack, che è mappato nella SRAM.

La gestione degli interrupt si abilita nel registro di stato (STATUS REGISTER) configurando il bit "Global Interrupt Enable". Ad ogni interrupt corrisponde un vettore di interrupt nella rispettiva tabella dei vettori. Gli interrupt hanno una priorità in funzione dell'indirizzo nei quali sono mappati, quindi più basso sarà l'indirizzo del vettore più la priorità sarà alta.

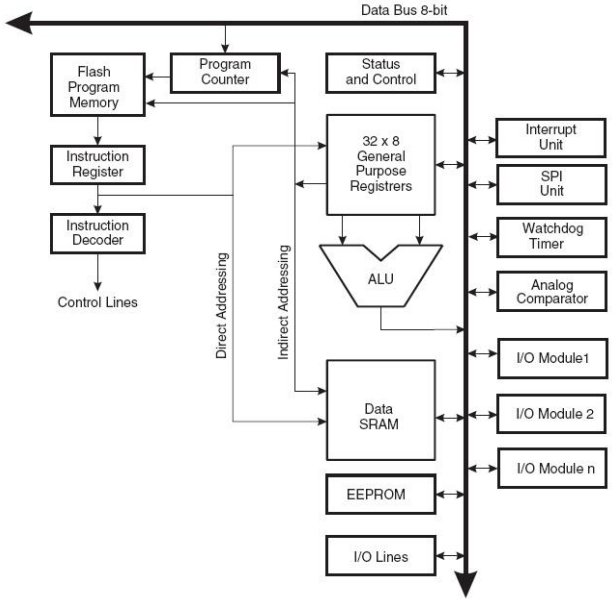

Figura 3 – Architettura AVR.

4.6 NOTE PER LA PROGRAMMAZIONE IN C

La creazione del sorgente può essere fatta utilizzando un qualsiasi editor di testo, ricordandosi però poi di salvare il listato con l'estensione ".c". I listati così creati però hanno bisogno anche di un altro file con estensione ".h" che contiene tutte le firme dei metodi utilizzati nel ".c"; in pratica è come un file di definizione che si occupa di dare una descrizione molto rapida del funzionamento del listato.
Quando tutti i listati saranno pronti si può passare alla loro compilazione. I passi della compilazione sono i seguenti:

- ***Controllo del codice sorgente*** da parte del **PREPROCESSORE**, che si occuperà della gestione dei "#define" e "#include", costruendo quindi ciò che è una mappa del programma.
- ***Compilazione del listato*** per mezzo del **COMPILATORE**, che si occupa di trovare eventuali errori di sintassi nei listati e di eliminare i commenti (fondamentali a parere personale per la comprensione delle operazioni svolte, al fine che possano essere modificate in futuro da un altro progettista). Il risultato del PREPROCESSORE è un listato in assembly, che è un linguaggio di programmazione a basso livello del tipo load/store, quindi molto scarno ed essenziale.
- ***Assemblaggio*** o **ASSEMBLER**, quindi si crea un file oggetto che dipenderà dal sistema operativo del pc che si sta utilizzando. Allora se si sta usando un SO fondato su UNIX l'uscita del codice oggetto avrà estensione ".o", altrimenti se si usa WINDOWS l'estensione sarà ".obj".
- ***Gestione delle dipendenze*** o **LINKER**, si gestiscono dipendenze tra i listati. Infatti è possible implementare all'interno di un listato altre funzioni provenienti da altri programmi oppure

importare semplicemente delle librerie. Il risultato di questa operazione è un file eseguibile, nel caso di WINDOWS un ".exe".

4.7 PROGRAMMAZIONE DEL μC

La programmazione dell'ATMega32 può essere effettuata in due modi:

- Programmazione diretta in linguaggio assembly
- Programmazione in linguaggio C

In questo caso si è optato per la seconda scelta, in quanto il C è ormai considerato un linguaggio standard sia per quanto concerne la progettazione di software che la progettazione di hardware.

Per programmare la scheda è necessario il kit di sviluppo proprietario della ATMEL che è il JTAGICEMKII.
È un tool di debugging on-chip che può "debuggare" in due diverse modalità:

- JTAG
- debugWIRE

La connessione al pc può essere eseguita mediante interfaccia USB o RS232.
Per lo sviluppo della I023 si è scelto di usare l'interfaccia d'ingresso USB e l'interfaccia di uscita alla scheda JTAG.
Per la programmazione e il debug è necessario fornire la tensione di alimentazione dall'esterno, quindi è sufficiente collegare la scheda alla rete di alimentazione 220Vac in quanto la I023 è dotata di un alimentatore integrato che si occupa di gestire l'alimentazione delle varie parti della scheda mediante opportuni adattamenti.

AVR studio permette di compilare il sorgente preparato per la scheda creando quindi dei file di output che saranno poi caricati nel μC per l'esecuzione del programma.
Questo tool di sviluppo permette di fare debug in tempo reale, è possibile visualizzare tutte le operazioni che fa il μC con esecuzioni "step to step".
Tutti i registri del μC sono accessibili nella modalità read/write.
Quindi se si necessità di cambiare un bit di configurazione al volo è possibile farlo visualizzando real-time le modifiche apportate.
Questo approccio risulta molto costruttivo se si stanno effettuando delle operazioni dubbie oppure rischiose.
Un esempio molto pratico si rivela quando la scheda non si comporta come il programmatore ha deciso, quindi l'esecuzione step to step permette di accorgersi di eventuali errori di sviluppo oppure anche di comportamenti anomali della circuiteria gestita dal μC.
Una esecuzione di questo tipo è altrettanto importante quando si sta facendo una esecuzione pericolosa, basti pensare che sulla scheda elettrica sono montati tutti i componenti necessari per il funzionamento della stessa.
Una gestione poco accurata della scheda, accompagnata da un'esecuzione immediata del programma, può causare seri danni alla circuiteria; talvolta danni irreparabili che possono mettere a rischio persino l'incolumità del progettista.
Non essendo possibile una esecuzione su una macchina virtuale (vedi JAVA), tutti i comandi predefiniti in fase di progettazione vengono eseguiti alla lettera dal μC, incurante dei circuiti periferici ad esso connesso.

4.8 INTERFACCIA JTAG E SISTEMA DI DEBUG ON-CHIP

L'interfaccia JTAG di ATMEL è compatibile con lo standard IEEE 1149.1.
L'unità di debug ha accesso a:

- tutte le periferiche interne del μC
- ram interne ed esterne
- registri interni
- program counter
- EEPROM e memorie FLASH

Il debug è consentito anche in condizioni di arresto dell'esecuzione del programma, di seguito le modalità di arresto di esecuzione:

- Istruzioni di interruzione AVR
- Interruzione al cambiamento della memoria di programma
- Esecuzione step to step
- Possibilità di inserire break points su indirizzi singoli o multipli
- Possibilità di inserire break points in memoria dati su indirizzi singoli o multipli

È possibile programmare la memoria FLASH, EEPROM, i fusibili e i lock bits.
È inoltre presente il supporto di debug on-chip.
Per configurare la modalità JTAG si configurano 4 pin, definiti come TAP (Test Access Port).
Questi pin sono:

- TMS (Test Mode Select). Serve per la navigazione attraverso il controllore di stato TAP.
- TCK (Test Clock). Si usa per sincronizzare le operazioni del JTAG.
- TDI (Test Data In). Trasferisce i dati in ingresso all'Instruction Register o al Data Register.
- TDO (Test Data Out). Preleva i dati dall'Instruction Register o dal Data Register.
- TRST (Test Reset). Resetta il μC.

Ecco un esempio di debug:

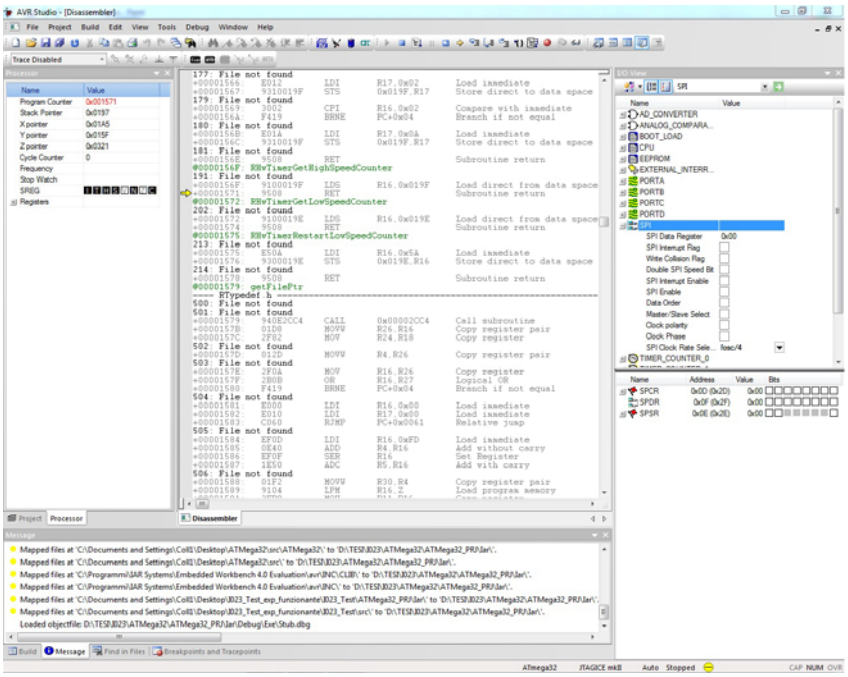

Figura 5 – Finestra di Debug AVR Studio.

Da questo screenshot si intuisce la gestione completa che offre questo tool di sviluppo.
Nel fianco sinistro è presente la voce “Processor”.

Si possono notare i seguenti componenti del μP effettivamente:

- Program Counter
- Stack Pointer
- X, Y, Z Pointer
- Cycle Counter
- Frequency
- Stop Watch
- SREG
- Registers

Nel fianco destro superiore appare la sottofinestra degli I/O, “I/O View”:

- AD_Converter
- Analog_Comparator
- Boot_Load
- CPU
- EEPROM
- External_Interrupt
- JTAG
- PORTA, PORTB, PORTC, PORTD
- SPI
- Timer_Counter_0, Timer_Counter_1, Timer_Counter_2
- TWI
- USART0
- Watchdog

Nel fianco destro inferiore appaiono invece tutte le sottovoci dell’I/O View. In base alla voce selezionata sarà possibile apportare modifiche o semplicemente leggere tutto ciò che accade ai registri specializzati.

Ogni qualvolta si vada ad interrompere un processo di debug appare una schermata di "disassembler", che mostra tutte le ultime operazioni fatte dal μC. Di seguito viene fornito un esempio:

```
@000009BF: RAdcInit
---- RAdc.c ------------------------------------------------------------------
+000009BF:        D013        RCALL       PC+0x0014    Relative call subroutine
+000009C0:        9100007B    LDS         R16,0x007B   Load direct from d.s.
+000009C2:        7F08        ANDI        R16,0xF8     Logical AND with immed.
+000009C3:        9300007B    STS         0x007B,R16   Store direct to d.s.
+000009C5:        9100007C    LDS         R16,0x007C   Load direct from d.s.
+000009C7:        7E00        ANDI        R16,0xE0     Logical AND with immed.
+000009C8:        9300007C    STS         0x007C,R16   Store direct to d.s.
+000009CA:        E001        LDI         R16,0x01     Load immediate
+000009CB:        9300026F    STS         0x026F,R16   Store direct to d.s.
+000009CD:        9100007A    LDS         R16,0x007A   Load direct from d.s.
+000009CF:        6400        ORI         R16,0x40     Logical OR with immediate
+000009D0:        9300007A    STS         0x007A,R16   Store direct to d.s.
+000009D2:        9508        RET                      Subroutine return
@000009D3: RAdcRefresh
```

Queste sono le tipiche istruzioni di un linguaggio di programmazione di tipo RISC.
La prima colonna identica il valore del program counter (PC). In questo breve segmento di codice è stato invocato il metodo RAdcInit della classe RAdc.c. In caso di assembly si parla di chiamata a subroutine.
Si noti che ad ogni istruzione eseguita dalla CPU il valore del PC non aumenta sempre di una unità, può aumentare anche di due.
Questo è possibile per il fatto che non tutte le istruzioni dell'ATMega32 non richiedono tutte il tempo di un ciclo macchina, pur essendo un μC di tipo RISC e quindi ad istruzioni veloci e semplificate per velocizzare l'esecuzione del sorgente.
Un esempio tangibile lo si ha nel passaggio dalla seconda alla terza istruzione:

- Il valore del PC nella prima istruzione vale Hex000009C0=Dec2496, il μC sta caricando informazioni dal registro R16 (data space)
- Terminata questa istruzione il PC vale Hex000009C2=Dec2498, allora 2498-2496=2.

Da questa considerazione si può capire che ogni istruzione, che va a caricare informazioni dal nostro data space, impiega due cicli macchina.
La chiamata a subroutine con l'istruzione RET.

Per meglio capire il linguaggio assembly sopra riportato si fornisce di seguito il sorgente in linguaggio C.

```
void RAdcInit(void)
{
  RAdcRefresh();
  ADCSRB.Bits.ADTS = 0; //Setto in free running mode
  ADMUX.Bits.MUX = 0; //Setto l'ADC0 nel mux
  iIsSampling = TRUE;
  ADCSRA.Bits.ADSC = 1; //Faccio partire l'ADC
}
```

Il sorgente sopra riportato è costituito da 4 istruzioni di set e da una chiamata a subroutine.

- `RAdcRefresh();`

La chiamata a subroutine viene gestita dall'assembler con un'istruzione di salto, quindi con un aumento del PC.

- `ADCSRB.Bits.ADTS = 0;`

L'ADCSRB è il registro di controllo e di stato B.

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 6 – ADCSRB.

Con questa istruzione tutti i bit di tipo ADTS vengono portati immediatamente a 0. Le seguenti istruzioni si interessano di portare a termine la procedura:

```
+000009C0:   9100007B   LDS         R16,0x007B       Load direct from data space
+000009C2:   7F08       ANDI        R16,0xF8         Logical AND with immediate
+000009C3:   9300007B   STS         0x007B,R16       Store direct to data space
```

Viene quindi prelevato il contenuto di ADCSRB dalla memoria nel registro R16, viene fatta una AND con Bin11111000=Hex0xF8 e viene salvato il nuovo contenuto del registro nella stessa locazione di memoria dove era stato precedentemente prelevato (0x007B).

- `ADMUX.Bits.MUX = 0;`

L'ADMUX è il registro che permette di selezionare l'ADC desiderato mediante un multiplexer.

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 7 – ADMUX.

Tutti i bit di tipo MUX vengono portati a 0 mentre tutti gli altri rimangono invariati. Le istruzioni interessate sono le seguenti:

```
+000009C5:   9100007C   LDS         R16,0x007C       Load direct from data space
+000009C7:   7E00       ANDI        R16,0xE0         Logical AND with immediate
+000009C8:   9300007C   STS         0x007C,R16       Store direct to data space
```

Viene caricato il contenuto del registro ADMUX (0x007C) in R16, vengono portati a 0 i bit interessati con una and logica tra R16 e Bin11100000=Hex0xE0 e viene ristabilito il nuovo contenuto del registro ADMUX in memoria.

- `iIsSampling = TRUE;`

È una variabile locale della classe RAdc.

```
+000009CA:   E001       LDI         R16,0x01         Load immediate
+000009CB:   9300026F   STS         0x026F,R16       Store direct to data space
```

È utilizzata come flag di tipo BOOLEAN per la gestione della concorrenza con l'interrupt. Ogni qualvolta si va a leggere l'ingresso analogico la variabile va a 1 per indicare che si sta effettuando un campionamento degli ingressi. In R16 è quindi caricato il valore immediato Hex0x01=Dec1 ed è salvato subito nella locazione di memoria dedicata alla variabile che in questo caso è 0x026F, portandola quindi a 1=TRUE.

- `ADCSRA.Bits.ADSC = 1;`

L'ADCSRA è il registro di controllo e di stato A.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 8 – ADCSRA.

Il bit ADSC viene portato a 1, mentre tutti gli altri rimangono invariati. Di seguito le istruzioni interessate:

```
+000009CD:   9100007A   LDS         R16,0x007A       Load direct from data space
+000009CF:   6400       ORI         R16,0x40         Logical OR with immediate
+000009D0:   9300007A   STS         0x007A,R16       Store direct to data space
```

Viene caricato in R16 il contenuto di ADCSRA (0x007A). ADSC è portato a 1 mediante una or logica con Bin01000000=Hex0x040, allora il nuovo contenuto del registro viene rimappato in memoria.

4.9 STRUTTURA MEMORIA E MODALITÀ DI ACCESSO I023

La scheda è stata progettata in modo da fornire all'utente la possibilità di gestire il contenuto della EEPROM interna del μC dall'esterno.
I modi per accedere alla consolle sono due:

- Accesso diretto da plancia di comando LCD a sette segmenti.
- Accesso tramite software di gestione seriale: SERAMI copyright Micronova.

Per quanto riguarda invece la gestione dei parametri intrinseci del firmware della I023, si possono modificare unicamente utilizzando l'interfaccia seriale sviluppata da Micronova in combinazione con SERAMI.
Di seguito viene riportato uno schema che descrive in modo sintetico la struttura della memoria e la
modalità di accesso dall'esterno.

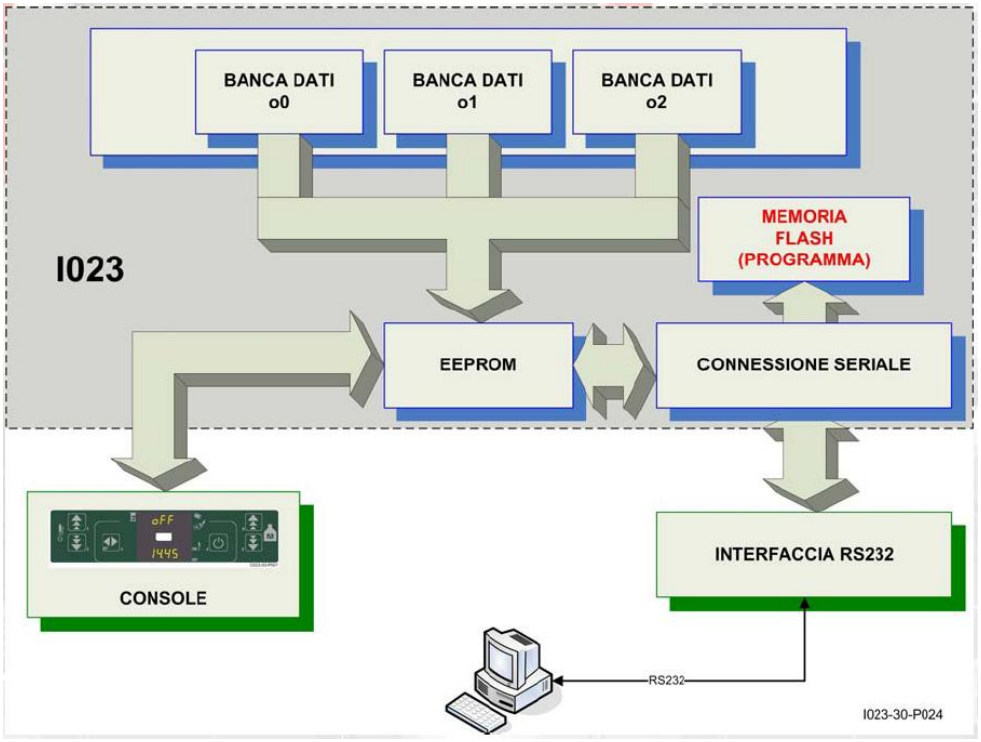

Figura 4 – Struttura memoria e modalità di accesso I023.

Per quanto concerne la descrizione della piedinatura del μC, le specifiche tecniche, lo schema di connessione e la tabella di I/O della IO23 si veda l'***APPENDICE***.

4.10 GESTIONE SALVATAGGIO OFFSET TEMPERATURA TERMOCOPPIA IN EEPROM

La termocoppia è un sensore di temperatura abbastanza preciso ed economico, che è utilizzato nel caso specifico per misurare la temperatura dei fumi della stufa a pellet.
Come tutte le sonde, essa infatti necessita di essere tarata, per poter rilevare correttamente la grandezza fisica per la quale è stata progettata. Una termocoppia è costituita da due conduttori di natura metallica differente; una variazione di temperatura comporta una differenza di potenziale tra gli estremi dei due filamenti. Nel caso della I023, la termocoppia non è collegata direttamente al μC, è necessario collegarla nel modo figurato nella prossima immagine. Il diodo D6 e la resistenza R32 servono per simulare una termocoppia che misura la temperatura ambiente (necessaria per la taratura dello zero) e IC7 (TLC271, ovvero un amplificatore operazionale), in connessione integrativa, in modo da fornire un valore di tensione opportuno all'ingresso del Mega32.

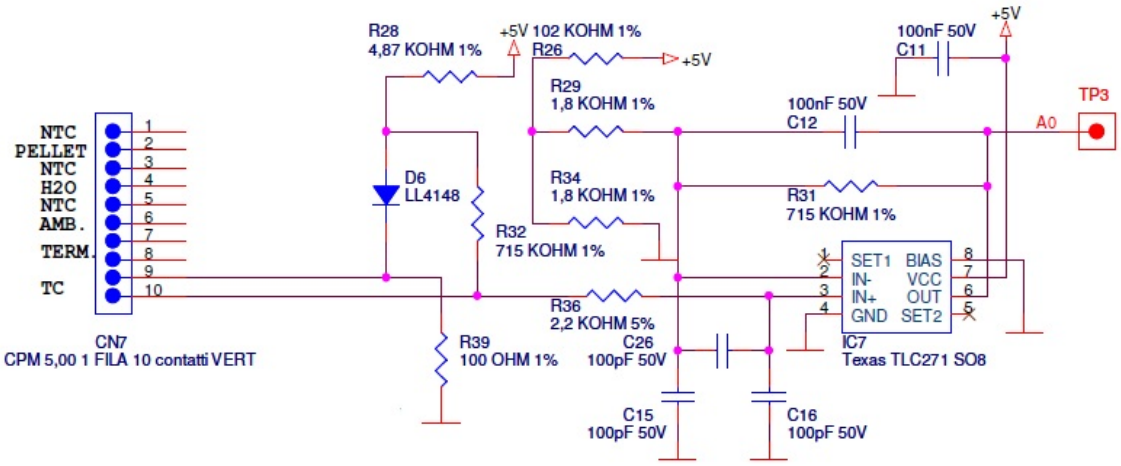

Figura 9 – Schema elettrico gestione termocoppia.

La taratura segue questo schema di principio:

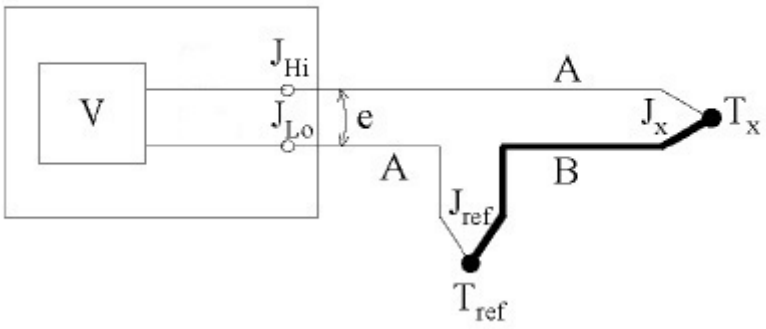

Figura 10 – Schema di taratura termocoppia.

La tensione V ai terminali della termocoppia è proporzionale alla differenza di temperatura tra quella misurata e quella ambiente e ad un coefficiente di proporzionalità K:

$$e = K(T_x - T_{ref}) = K(Offset) \qquad \textbf{Relazione 1}$$

Per la tarare lo zero della termocoppia vengono messi in corto circuito gli estremi (A e B), non viene usata la termocoppia che misura la temperatura T_x. Per la taratura dello zero ci si affida alla misura di temperatura del diodo. Ai capi del diodo è presente una tensione che, convertita da un algoritmo interno al Mega32, dovrebbe corrispondere alla temperatura ambiente.
Il collaudatore, tramite l'apposita consolle grafica collegata via flat alla I023, imposta un valore sulla consolle pari alla temperatura ambiente. Tale valore viene salvato contemporaneamente sia in RAM che in EEPROM.
Il valore salvato, corrisponde alla differenza delle due precedenti misurazioni:

TEMPERATURA RILEVATA – TEMPERATURA AMBIENTE = OFFSET

Al momento della taratura, l'offset dovrebbe tendere a zero, in quanto la temperatura ambiente e quella rilevata dal diodo dovrebbero coincidere. Questo però può non avvenire sempre per errori di rilevazione delle temperatura generati dalla sonda e per approssimazioni nella conversione del valore impostato via consolle. Viene considerato attendibile un offset compreso tra +/- 5 °C.

L'offset impostato viene utilizzato come valore campione per stabilire tutti gli altri valori di temperatura misurati dalla termocoppia. Questo si ottiene girando la ***Relazione 1***, conoscendo la tensione d'uscita e l'offset si ricava il coefficiente di proporzionalità, in mV/°C.
Stabilito il K, di volta in volta l'offset salvato in memoria dipende dalla tensione ai capi della termocoppia e da K, il tutto si fa utilizzando sempre la ***Relazione 1***.

Si richiedeva inoltre che venisse fatta una modifica a livello grafico che dimostrasse all'utente l'effettiva memorizzazione della temperatura in EEPROM.
È stata quindi introdotta una nuova schermata che evidenzia l'effettiva modifica in questo modo:

- Primo passaggio: IMPOSTAZIONE DELLA TEMPERATURA AMBIENTE

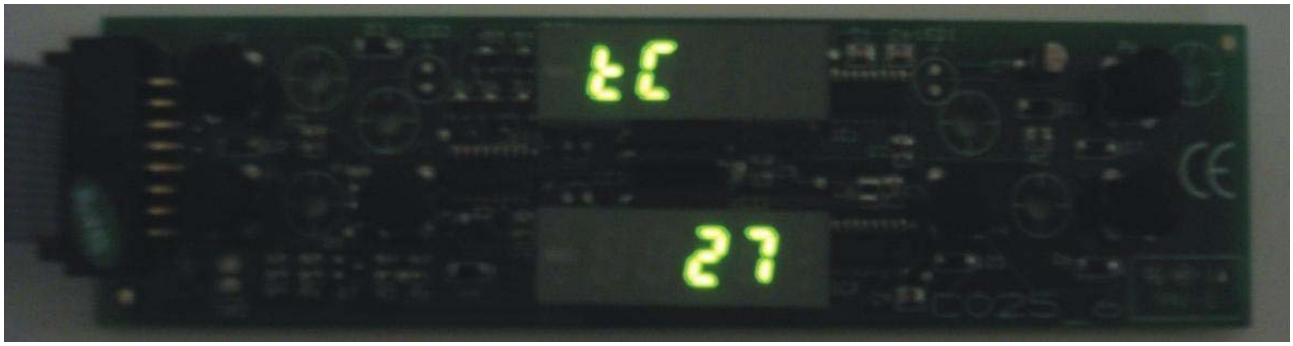

Figura 11 – Impostazione temperatura ambiente.

Nel display a 7 segmenti superiore viene indicato lo step del collaudo, in quello inferiore la temperatura impostata tramite consolle.

- Secondo passaggio (ultimo step del collaudo): VERIFICA MEMORIZZAZIONE TEMPERATURA IN EEPROM

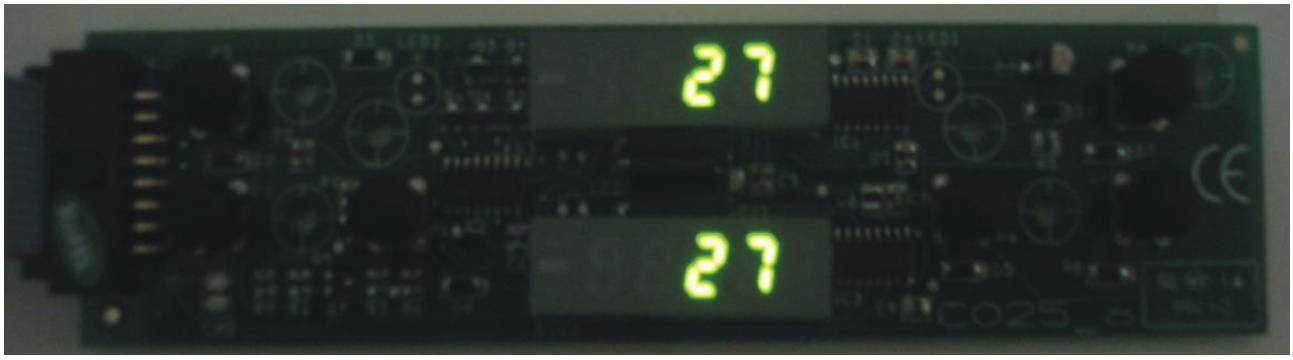

Figura 12 – Verifica memorizzazione temperatura in EEPROM (contenuto RAM = contenuto EEPROM)

A questo punto viene fatta una verifica dei dati salvati in RAM e in EEPROM (il display sopra visualizza il contenuto della RAM, quello sotto la EEPROM).
Se il contenuto delle due locazioni è identico allora il salvataggio delle impostazioni di temperatura è avvenuto correttamente.
Allo spegnimento della scheda il μC non avrà difficoltà a caricare le impostazioni preimpostate, non ci sarà rischio quindi di compromettere il funzionamento della stufa.
Se non si impostasse questo offset, la temperatura dei fumi risulterebbe sempre più alta di quello che effettivamente è; ciò comporterebbe ad uno spegnimento della stufa immediato per motivi di sicurezza e anche per non danneggiare gli apparati elettrico/meccanici della stufa stessa.
Per realizzare questa funzionalità è stato introdotto il refresh della temperatura in EEPROM ogni qualvolta la si vada a incrementare o decrementare.
Infatti ogni volta che la temperatura subisce variazioni il suo valore è salvato solamente in RAM e successivamente copiato da RAM a EEPROM alla fine del setting.
Ora la temperatura viene scritta contemporaneamente nelle due memorie, garantendo quindi un maggior livello di affidabilità.

Capitolo 5
SECONDO PROGETTO

Questo secondo progetto riguarda la progettazione dell'interfaccia grafica per un casco termostimolatore per capelli per la Ceriotti s.r.l. di Milano. Il casco in questione presenta numerose funzionalità. Le prestazioni e lavori tecnici forniti dal termo stimolatore sono: Meches, Decolorazioni, Colorazioni, Sostegno e Asciugature. Ciascuna funzione ha un programma preimpostato, contenuto in memoria. È comunque possibile salvare in memoria programmi personalizzati, e in ogni caso è possibile una gestione completamente manuale del casco. La gestione manuale comporta la gestione delle tempistiche di trattamento, velocità di della ventola, temperatura dei riscaldatori e lampada all'ozono per l'eliminazione di batteri, acari e odori di trattamento (acidi per piega). La macchina può essere utilizzata in tutta la Comunità Europea, grazie al supporto multilingue e la costruzione e il collaudo di esso seguendo le Normative Europee. Tutte le configurazioni e i trattamenti sono gestiti da un'interfaccia utente/macchina molto comoda e intuitiva per mezzo di un touch screen. La macchina però non vuole puntare solamente alla qualità del trattamento, un occhio di riguardo è stato dedicato anche al confort. Il cliente ha la possibilità, tramite supporto portatile USB, di ascoltare MP3 durante tutta la durata del trattamento; funzionalità unica nel suo genere e per la prima volta nel mondo. Di seguito sono trattati gli aspetti più importanti del progetto, con un approfondimento relativamente alla progettazione dell'interfaccia utente/macchina, attività seguita durante il tirocinio.

5.1 SCHEDE IN ANALISI

Si tratta essenzialmente di due schede. Una scheda viene utilizzata come master ed una come slave.
Esse gestiscono il funzionamento di un casco stimolatore per capelli. È un casco molto particolare, infatti oltre ad occuparsi del normale funzionamento di stimolatore per capelli, offre al cliente la possibilità di ascoltare della musica durante al trattamento e allo stesso tempo offre all'acconciatore un'interfaccia immediata e semplice per merito di un lcd/touch panel.
La scheda master si occupa della gestione di basso livello, di tutte le funzionalità sia avanzate che di base della scheda e della gestione della scheda slave via seriale half-duplex. La scheda slave si occupa dell'attivazione dei carichi.

5.2 ARCHITETTURA ARM

5.2.1 CENNI STORICI

Storicamente l'acronimo ARM significava Acorn RISC Machine, modificato successivamente in Advanced RISC Machine. È una famiglia di μP RISC a 32 bit, utilizzata prevalentemente nei sistemi embedded, dove i bassi consumi sono all'ordine del giorno.
La progettazione dell'ARM risale al 1983, introdotta dell'area di ricerca e sviluppo della Acorn Computers. I capi progetto Roger Wilson e Steve Furber pensavano di realizzare una nuova versione del MOS Technology 6502. Infatti, Acorn utilizzava il MOS 6502 per la costruzione dei suoi computer; riuscire a realizzare dei μP proprietari avrebbe comportato vantaggi competitivi all'azienda.

La prima versione dell'ARM, l'ARM1, nacque nel 1985, ma la prima versione commerciale ARM2, venne introdotta nel mercato nel 1986.
L'ARM2 era un μP con un bus dati da 32 bit, un bus indirizza da 26 bit per indirizzare fino a 64 Mbyte e dei registri da 16 e 32 bit. Le prestazioni dell'ARM2 erano paragonabili a quelle del Motorola 68000, pur avendo solo 30000 transistor rispetto ai 68000 del Motorola. L'ARM2 doveva la sua semplicità all'assenza di microcodice, che nel 68000 occupava circa il 25% dei transistor totali, e all'assenza della memoria cache.

Alla fine degli anni 80, Apple e Acorn cooperarono per sviluppare una nuova versione dell'ARM. Venne quindi creata una nuova compagnia, la Advanced RISC Machine che divenne solo ARM con la quotazione, nell'indice tecnologico NASDAQ nel 1998, della società madre ARM Holdings. La collaborazione con Apple portò alla nascita del core ARM6, utilizzato all'interno dell'Apple Newton. Il nuovo core aveva solo 5000 transistor in più rispetto all'ARM2. era una scelta di progettazione innovativa, in quanto permetteva la combinazione del core con vari componenti opzionali, con lo scopo di ottenere una CPU completa, a basso consumo e soprattutto ottimizzata.
Il primo successo della ARM, fu la produzione della settima versione del core; utilizzato su sistemi portatili quali telefonini e console portatili.

Di seguito lo schema di principio dell'architettura.

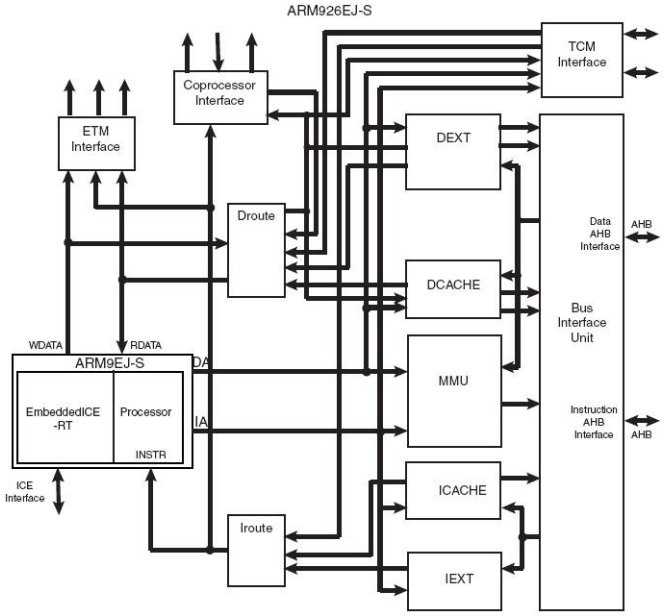

Figura 13 – Architettura ARM.

5.2.2 NOTE DI PROGETTO

L'architettura ARM è un'architettura RISC che include:

- Architettura load/store
- Mancato supporto ad accessi alla memoria non allineati
- Set di istruzioni ortogonale
- Registri a 16/32 bit
- Operation code a lunghezza fissa per semplificare la decodifica e l'esecuzione a costo di diminuire la densità del codice
- Esecuzione in un ciclo di clock per la maggior parte delle istruzioni

Il progetto semplice dell'ARM si distingueva rispetto ai più complessi Intel 80286 e Motorola 68020 per:

- Esecuzione condizionata di molte istruzioni per ridurre i salti e compensare gli stalli della pipeline
- Le operazioni aritmetiche sono le uniche che possono alterare i registri delle esecuzioni condizionate
- Shifter a 32 bit che può essere utilizzato in contemporanea con la maggior parte delle istruzioni senza penalizzazioni di tempo
- Metodo di indirizzamento a indice molto potente.
- Interrupt a 2 livelli molto veloce e semplice con un sottosistema di registi collegati che commutano

La peculiarietà più interessante sta nella struttura delle istruzioni. Le istruzioni contengono 4 bit addizionali per la realizzazione di codici condizionali per ogni istruzione.
Tali codici hanno ridotto le possibilità di indirizzo, data la scarsità di bit dedicati all'indirizzamento.
Il grande vantaggio però è che questi codici evitano di compiere salti nel caso di semplici if. Un esempio è la ricerca del massimo comune divisore.

In C scriveremo:

```
int gcd (int i, int j)
{
   while (i != j)
      if (i > j)
          i -= j;
      else
          j -= i;
   return i;
}
```

Che in assembly ARM diventa:

```
loop   CMP    Ri, Rj       ; set condition "NE" if (i != j)
                            ; "GT" if (i > j),
                            ; or  "LT" if (i < j)
       SUBGT  Ri, Ri, Rj   ; if "GT", i = i-j;
       SUBLT  Rj, Rj, Ri   ; if "LT", j = j-i;
       BNE    loop         ; if "NE", then loop
```

Evitando così i rami del then e dell'else. Ogni istruzione viene eseguita in un solo ciclo macchina. Il risultato è che i programmi, scritti in linguaggio assembly ARM, ottimizzano il riempimento delle pipeline, riducendo anche la frequenza del core nell'esecuzione delle istruzioni.
Istruzioni ottimizzate = riduzione frequenza core + riduzione dei consumi.
Altre caratteristiche degne di nota sono l'indirizzamento relativo al Program Counter e l'indirizzamento con il pre e post incremento, in aggiunta ogni nuova generazione di core viene arricchita con nuove istruzioni.
L'incremento delle prestazioni è raggiunto con la gestione multi stadio delle pipeline, per esempio un ARM7 aveva pipeline a tre stadi, mentre l'ARM9 aveva pipeline a cinque stadi.

5.2.3 SET DI ISTRUZIONI

- ***THUMB***

È un set di istruzioni a 16 bit che utilizza 4 bit per ogni istruzione. È un codice leggero in cui solo i salti possono essere condizionati e alcuni operation code non possono essere utilizzati da tutte le istruzioni. In sistemi con ridotta larghezza di banda fornisce ottime prestazioni rispetto al set di istruzioni completo. Conviene utilizzarlo quando il bus degli indirizzi verso la memoria è limitato, per esempio sistemi nei quali gli indirizzi possono essere mappati a 16 bit. In questo particolare caso la maggior parte del codice del programma conviene scriverla in THUMB e col il set completo di istruzioni per trattare le parti che richiedono una potenza di calcolo maggiore. THUMB è disponibile per architetture posteriori all'ARM6.

- ***JAZELLE***

Alcuni esemplari di ARM possono eseguire nativamente il Java Bytecode. Il primo μP dotato di questa struttura è l'ARM926J-S. È utilizzato nei telefoni cellulari per migliorare le prestazioni delle applicazioni JAVA.

- ***THUMB 2***

È un'estensione del THUMB, infatti possiede tutte le istruzioni a 16 bit delle versione precedente più delle istruzioni a 32 bit per aumentare la potenza di calcolo, avvicinandosi così alle prestazioni di un ARM a 32 bit. Sono state introdotte nuove istruzioni che permettono:

- Modifica di singoli bit
- Esecuzioni condizionate
- Gestioni tabelle con salti

- ***THUMB 2EE***

È una tecnologia implementata per la prima volta nel μP Cortex-A8. È l'evoluzione del THUMB 2, progettato per gestire il codice generato in tempo reale, durante le esecuzioni "Just in time". È una tecnologia progettata per linguaggi come Java e C, in modo da generare codice compilato di dimensioni ridotte riducendo l'impatto negativo sulle prestazioni.
Le nuove funzionalità ridotte sono le seguenti:

- Controllo automatico dei puntatori nulli prima di ogni istruzione di load/store
- Gestione di eventuali overflow negli array
- Istanziamento memoria per nuovi oggetti (caratteristica dei linguaggi ad alto livello)

- ***NEON***

È una combinazione di istruzioni a 64 e 128 bit di tipo SIMD (Single Instruction Multiple Data), in modo da accelerare e standardizzare il trattamento e l'elaborazione dei segnali multimediali.
La tecnologia si fonda su un set di istruzioni separato, registri indipendenti ed esecuzione del codice separata. Possono essere gestite operazioni vettoriali, operazioni che gestiscono molti dati con lo stesso programma. Possono essere gestite 16 operazioni contemporaneamente.

- ***VFP***

È un'estensione dell'architettura ARM con l'inserimento di un coprocessore matematico. Lo scopo dell'architettura è quello di fornire operazioni per la trattazione dei dati in virgola mobile a singola e doppia precisione in modo economico e compatibile con lo standard *ANSI/IEEE Std 754-1985 Standard for Binary Floating-Point Arithmetic*.
Sono presenti istruzioni per eseguire compressioni, decompressioni, grafica 3-D, analisi audio e molto altro ancora. Applicazioni tipiche sono i classici telefonini e smart-phone.

5.3 MICROCONTROLLORE AT91SAM9263 (ARM9)

Questo tipo di μC è utilizzato in moltissimi sistemi embedded di larga distribuzione. Applicazioni comuni sono i telefoni cellulari e i navigatori satellitari. Un esempio dell'utilizzo della tecnologia ARM è l'iPhone di Apple che monta un ARM11.
È un μC con 324 pin di I/O e per far funzionare correttamente le varie periferiche richiede specifiche tensioni di alimentazione:

- 1.2V (da 1.08V a 1.32V) nominali per la CPU, le memorie e le periferiche
- 1.8V (da 1.65V a 1.95V) o 3.3V (da 3.0V a 3.6V) nominale per EBI0/EBI1 che sono i due bus esterni
- 3.3V (da 2.7V a 3.6V) nominali per le linee I/O e le linee USB
- 1.8 – 2.5 – 3 – 3.3V per le linee I/O corrispondenti all'interfaccia dei sensori per l'immagine
- 1.2V (da 1.08V a 1.32V) nominali per l'oscillatore interno del clock e per il controller del sistema
- 3.3V (da 3.0V a 3.6V) nominali per le celle del PLL
- 3.3V (da 3.0 a 3.6V) nominali per l'oscillatore principale

L'AT91 è un μC di tipo RISC con architettura Harvard, quindi la memoria del programma è distinta dalla memoria dati. Questo implica che tutti gli indirizzi delle memorie partono da 0HEX.
Le istruzioni sono a 32 bit, le istruzioni sono molto più complesse rispetto ad un Mega32; infatti nelle istruzioni dell'ARM si possono passare fino a 5 parametri contro i 2 di un Mega32.
Sono presenti canali per la gestione di:

- USART
- unità di debug
- controllori seriali sincroni
- interfacce seriali
- controller audio AC97
- lettori di card multimediali

EBI0 e EBI1 forniscono banda passante al sistema e evitano colli di bottiglia durante l'accesso alle memorie esterne. Questi due bus gestiscono 4 tipi di memoria:

- Memorie statiche
- SDRAM
- Controllori ECC
- NAND Flash

L'ARM9 gestisce periferiche embedded:

- Interfaccia seriale
- Seriale a due fili
- USART

- Controller seriali sincroni
- Controller AC97
- Timer
- Controller PWM
- Interfaccia per Multimedia Card
- Controller CAN
- Controller USB
- Controller LCD
- Accelerazione grafica 2D
- Ethernet 10/100
- Interfaccia Sensore per immagini

L'ARM9 gestisce applicazioni multitasking mantenendo alte performance nell'uso delle risorse di memoria, consumi e prestazioni. Supporta istruzioni a 32 bit ARM, a 16 bit THUMB e a 8 bit JAVA.
È possibile utilizzare 37 registri:

- 31 registri per uso generico a 32 bit
- 6 registri di stato a 32 bit

5.4 INSTRUCTION SET ARM

L'ARM9 ha un instruction set suddiviso nelle seguenti categorie:

- Istruzioni di salto (branch)
- Istruzioni di condizionamento dati
- Istruzioni classiche del tipo LOAD & STORE
- Istruzioni di trasferimento per i registri di stato
- Istruzioni per il coprocessore
- Istruzioni per la generazione di eccezioni

Di seguito l'instruction set completo.

Mnemonic	Operation
BXJ	Branch and exchange to Java
BLX (1)	Branch, Link and exchange
SMLAxy	Signed Multiply Accumulate 16 * 16 bit
SMLAL	Signed Multiply Accumulate Long
SMLAWy	Signed Multiply Accumulate 32 * 16 bit
SMULxy	Signed Multiply 16 * 16 bit
SMULWy	Signed Multiply 32 * 16 bit
QADD	Saturated Add
QDADD	Saturated Add with Double
QSUB	Saturated subtract
QDSUB	Saturated Subtract with double

Mnemonic	Operation
MRRC	Move double from coprocessor
MCR2	Alternative move of ARM reg to coprocessor
MCRR	Move double to coprocessor
CDP2	Alternative Coprocessor Data Processing
BKPT	Breakpoint
PLD	Soft Preload, Memory prepare to load from address
STRD	Store Double
STC2	Alternative Store from Coprocessor
LDRD	Load Double
LDC2	Alternative Load to Coprocessor
CLZ	Count Leading Zeroes

Mnemonic	Operation
MOV	Move
ADD	Add
SUB	Subtract
RSB	Reverse Subtract
CMP	Compare
TST	Test
AND	Logical AND
EOR	Logical Exclusive OR
MUL	Multiply
SMULL	Sign Long Multiply
SMLAL	Signed Long Multiply Accumulate
MSR	Move to Status Register
B	Branch
BX	Branch and Exchange
LDR	Load Word
LDRSH	Load Signed Halfword
LDRSB	Load Signed Byte
LDRH	Load Half Word
LDRB	Load Byte
LDRBT	Load Register Byte with Translation
LDRT	Load Register with Translation
LDM	Load Multiple
SWP	Swap Word
MCR	Move To Coprocessor
LDC	Load To Coprocessor
CDP	Coprocessor Data Processing

Mnemonic	Operation
MVN	Move Not
ADC	Add with Carry
SBC	Subtract with Carry
RSC	Reverse Subtract with Carry
CMN	Compare Negated
TEQ	Test Equivalence
BIC	Bit Clear
ORR	Logical (inclusive) OR
MLA	Multiply Accumulate
UMULL	Unsigned Long Multiply
UMLAL	Unsigned Long Multiply Accumulate
MRS	Move From Status Register
BL	Branch and Link
SWI	Software Interrupt
STR	Store Word
STRH	Store Half Word
STRB	Store Byte
STRBT	Store Register Byte with Translation
STRT	Store Register with Translation
STM	Store Multiple
SWPB	Swap Byte
MRC	Move From Coprocessor
STC	Store From Coprocessor

Figura 14 – Instruction set ARM completo.

5.5 STRUTTURA DEL PROGETTO

Come precedentemente descritto vi sono due schede che gestiscono il funzionamento del casco. La scheda master J018 è controllata da una scheda montata su di essa che contiene il modulo AT91SAM9263. Tale concezione di progettazione è possibile paragonarla all'ottica dei personal computer, nei quali il μP non è saldato direttamente sulla scheda madre (come del resto avveniva alla fine degli anni 80 con gli Intel 80286). Dato l'elevato quantitativo di pin del μC, e quindi l'elevato quantitativo di piste da far passare nel PCB, è necessario realizzare un PCB a livelli sovrapposti: un PCB multistrato. In questo modo la gestione più grossa e complicata delle piste che partono dal μC è svolta in una scheda a parte, evitando di sovraccaricare di piste il PCB della scheda madre, rendendola ancora più fragile.
La J018 gestisce la scheda slave J017 via seriale. Questa scheda è controllata da un ATMega32 come la scheda I023 del primo progetto. La J017 gestisce le uscite delle lampade (che producono calore per l'asciugatura), una lampada ad ozono e una ventola con le rispettive velocità di funzionamento.

La scheda J017 è alimentata da una scheda di alimentazione: J026. La J026 converte la tensione di rete (115 o 230 Vac) in 12 Vcc. La scheda master è alimentata sfruttando un'uscita della J017 sempre a 12 Vcc, tramite un cavo a 4 fili intestato con connettore MODU II.

5.6 SCHEMA DI PRINCIPIO

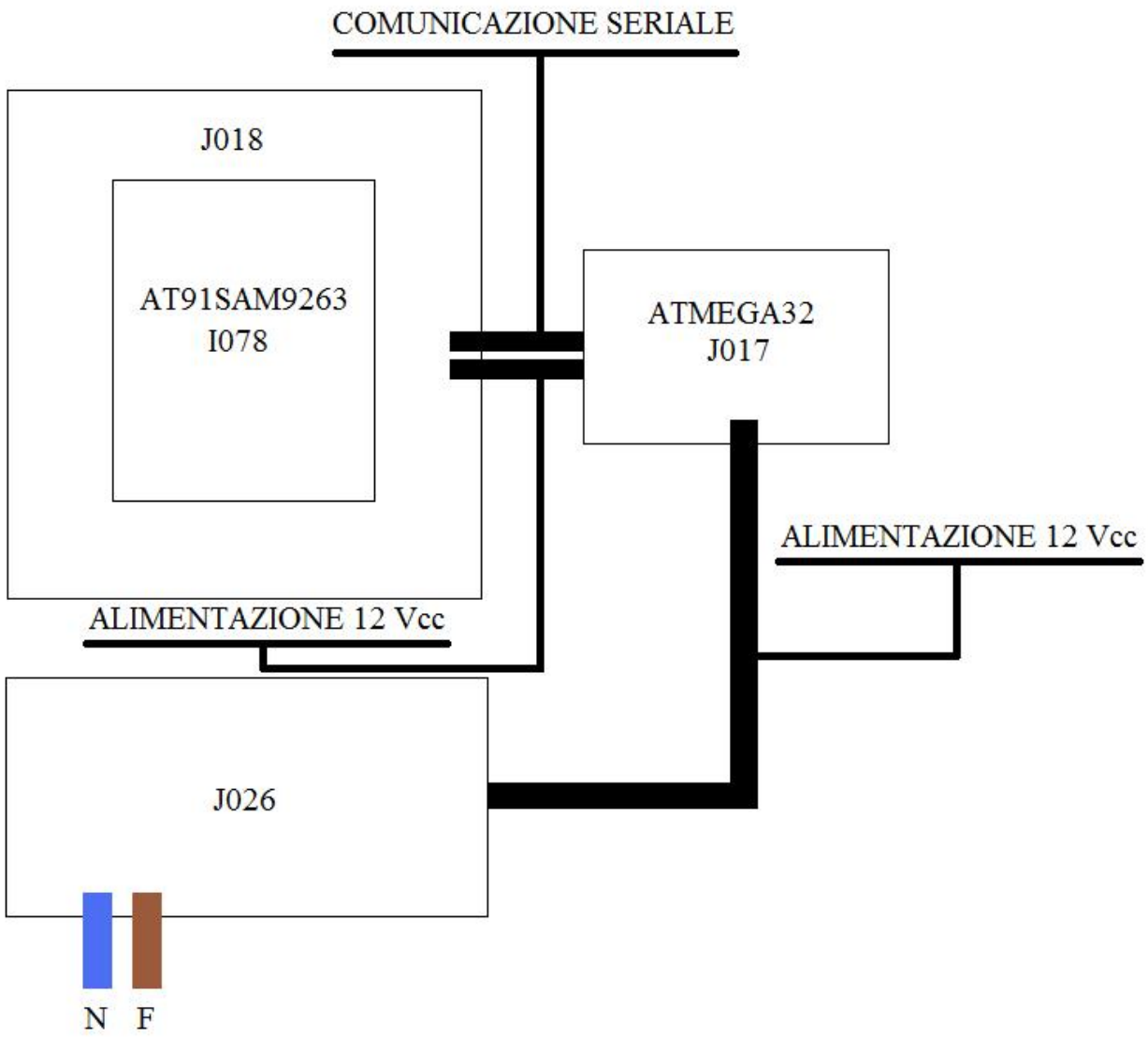

Figura 15 – Schema di principio progetto Ceriotti.

5.7 DESCRIZIONE FUNZIONALE DEI BLOCCHI

5.7.1 SCHEDA MASTER J018

I blocchi principali della scheda sono:

- Controllore USB 2.0 per collegamento di supporti rimovibili
- Gestione lcd/touch panel
- Controller switching da 12 Vcc a 5 Vcc e da 12 Vcc a 3.3 Vcc
- Gestione audio AC97

5.7.2 CONTROLLER USB

Sulla scheda master sono presenti due ingressi USB di Tipo A. Un ingresso “DEVICE” è gestito direttamente dall’AT91, fornendo i due ingressi D+, D- e la tensione di 5 Vcc @ 500 mA. Il secondo ingresso “HOST” è gestito dall’AT91 solo per quanto riguarda gli ingressi D+ e D-. I 5 Vcc @ 500 mA sono forniti dall’integrato SP2525A. Di seguito lo schema di principio inerente l’uso dell’SP2525A:

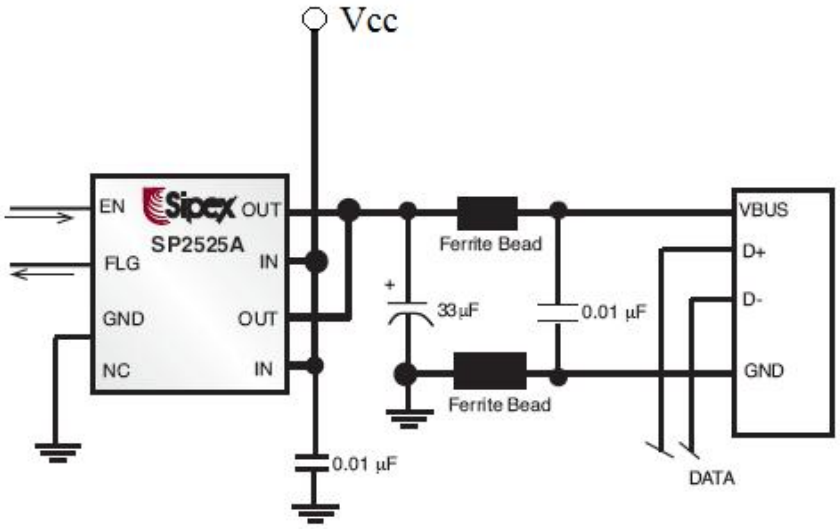

Figura 16 – Gestione controller USB.

5.7.3 GESTIONE LCD/TOUCH SCREEN

La gestione di questo blocco è affidata a un ADS7843E, un ADC con interfaccia seriale sincrona. È un registro ad approssimazioni successive con funzione di sample-hold. Il touch panel può essere immaginato come una mappa resistiva, ad ogni punto sullo schermo corrisponde un diverso valore resistivo. Il touch panel è gestito dal convertitore interno come un piano cartesiano, dove la coordinata (0;0) è il centro del pannello. L’ ADS7843E possiede un ingresso di interrupt dedicato per la modalità in basso consumo. Per la linea di interrupt risulta abilitata, allora alla fine di ogni conversione ADC, il convertitore interno viene spento. Al contrario se la linea risulta disabilitata, il convertitore sarà sempre acceso. La logica di funzionamento è di seguito riportata:

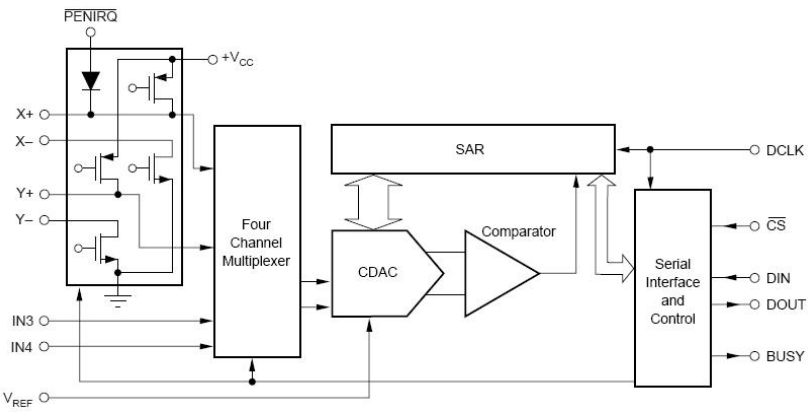

Figura 17 – Gestione LCD/Touch Screen.

5.7.4 CONTROLLER SWITCHING

Per generare le tensioni filtrate 5 Vcc e 3.3 Vcc sono stati impiegati due regolatori switching step-down (convertitori di tipo buck). È gestita anche la funzione di sleep per ridurre i consumi durante lo stand-by del casco. Per il dimensionamento del circuito è stato utilizzato un software della NATIONAL SEMICONDUCTOR. In base a tensione e corrente di uscita desiderate, il software calcola i componenti da collegare all'integrato. In questo modo è possibile realizzare dei dimensionamenti ottimi, garantendo la stabilità degli output, assolutamente necessaria per il corretto funzionamento, per esempio, delle USB e dell'AT91, i quali sono componenti molto delicati che richiedono soprattutto specifiche in corrente molto restrittive.

5.7.5 GESTIONE AUDIO AC97

L'integrato interessato per l'elaborazione audio è un AD1981B, prodotto da ANALOG DEVICES. È un codec audio AC97 SoundMAX con le seguenti caratteristiche:

- Amplificatore stereo integrato per ascolto con cuffie stereo
- Quantizza i segnali a 20 bit in formato PCM
- 3 ingressi stereo
- 1 ingresso per microfono
- Uscita mono per altoparlanti
- Uscita S/PDIF a 20 bit con bit rate di 48 e 44.1 kHz

I segnali elaborati dall'AD1981B vengono trasmessi a tre amplificatori di potenza da 1.5 W: SSM2211. I tre amplificatori sono utilizzati rispettivamente per:

- Amplificare canale destro
- Amplificare canale sinistro
- Amplificare l'uscita mono

Di seguito il blocco funzionale degli amplificatori:

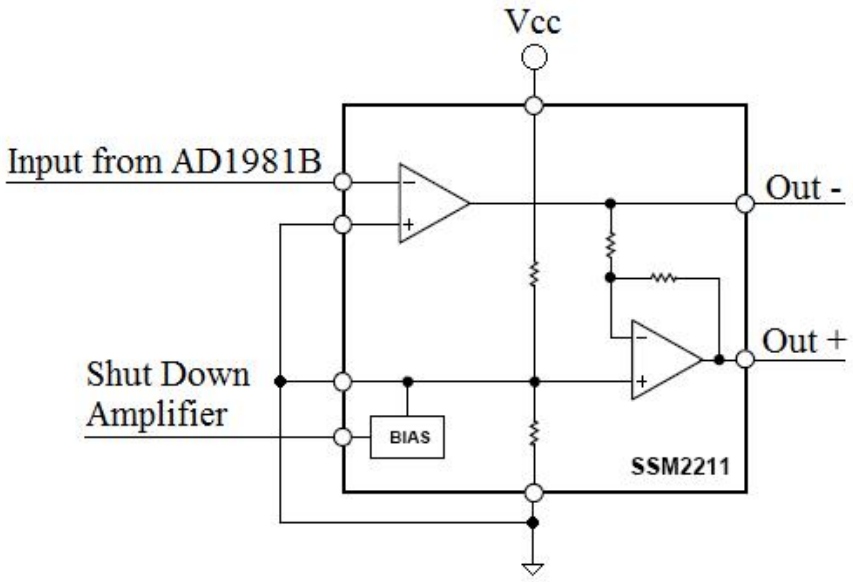

Figura 18 – Gestione audio AC97.

5.8 REALIZZAZIONE INTERFACCIA GRAFICA

Per la realizzazione dell'interfaccia grafica sono stati utilizzati software di sviluppo specifici e librerie grafiche specifiche. Per quanto concerne la parte di programmazione grafica è stato utilizzato l'ambiente di sviluppo ECLIPSE, sfruttando le librerie grafiche QT Trolltech by Nokia.
Il compilatore utilizzato è il GCC in ambiente Unix, il MinGW in ambiente Microsoft.

5.9 PROGRAMMARE IN Qt

La programmazione in Qt è un'estensione delle librerie del C++ e fornisce un sistema di programmazione multi piattaforma per la costruzione di UI (User Interface). Il software viene scritto una volta ed è possibile compilarlo per qualsiasi tipo di piattaforma che lo supporti. La prima pubblicazione risale al 1995, sviluppata da Haavard Nord e da Eirik Chambe-Eng, due norvegesi esperti di scienze informatiche.
Qt è supportato sia da Windows che da sistemi Unix, ed è dotato di stesse funzionalità e librerie per entrambe le piattaforme.
Qt è stato utilizzato per progettare il KDE di Linux, una delle tante interfacce grafiche disponibili.
La prima versione di KDE risale al 1997; e così il Qt diviene la base per lo sviluppo delle applicazioni KDE su Linux.
Nel 2000 nasce Qt/Embedded (QTopia Core), con lo scopo di caricarlo nei dispositivi embedded con il kernel di Linux. Già dalla versione 3.0, la piattaforma di sviluppo è disponibile per Windows, Unix, Linux, Embedded Linux e Mac OS X.
La piattaforma è stata utilizzata per sviluppare applicazione molto famose come Google Earth e Skype.

Con Qt è possibile realizzare questi tipi di progetto:

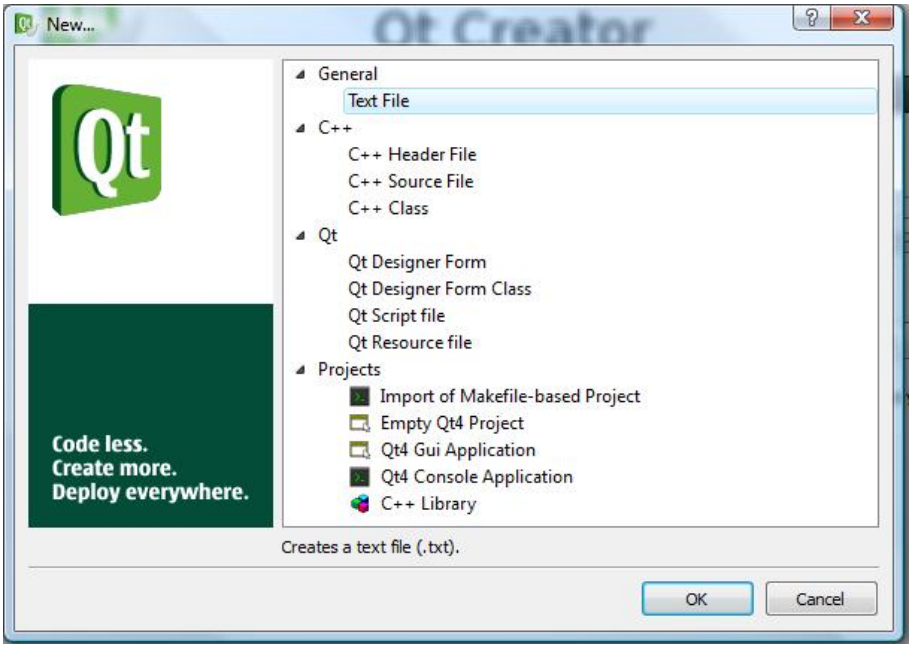

Figura 19 – Qt.

Si parte dal più comune file di testo e si arriva ai progetti Qt. Come dice il motto: ***"Code less. Create more. Deploy everywhere."***, è possibile realizzare interfacce grafiche in modo molto semplice, senza utilizzare codice, solo spostando i widget necessari per realizzare l'interfaccia grafica. Questo comporta una maggior semplicità realizzativa, tempi di consegna più ridotti e maggior intuitività.
Di seguito si può vedere l'interfaccia grafica fornita dal team di Qt, semplice ma non per questo poco professionale.

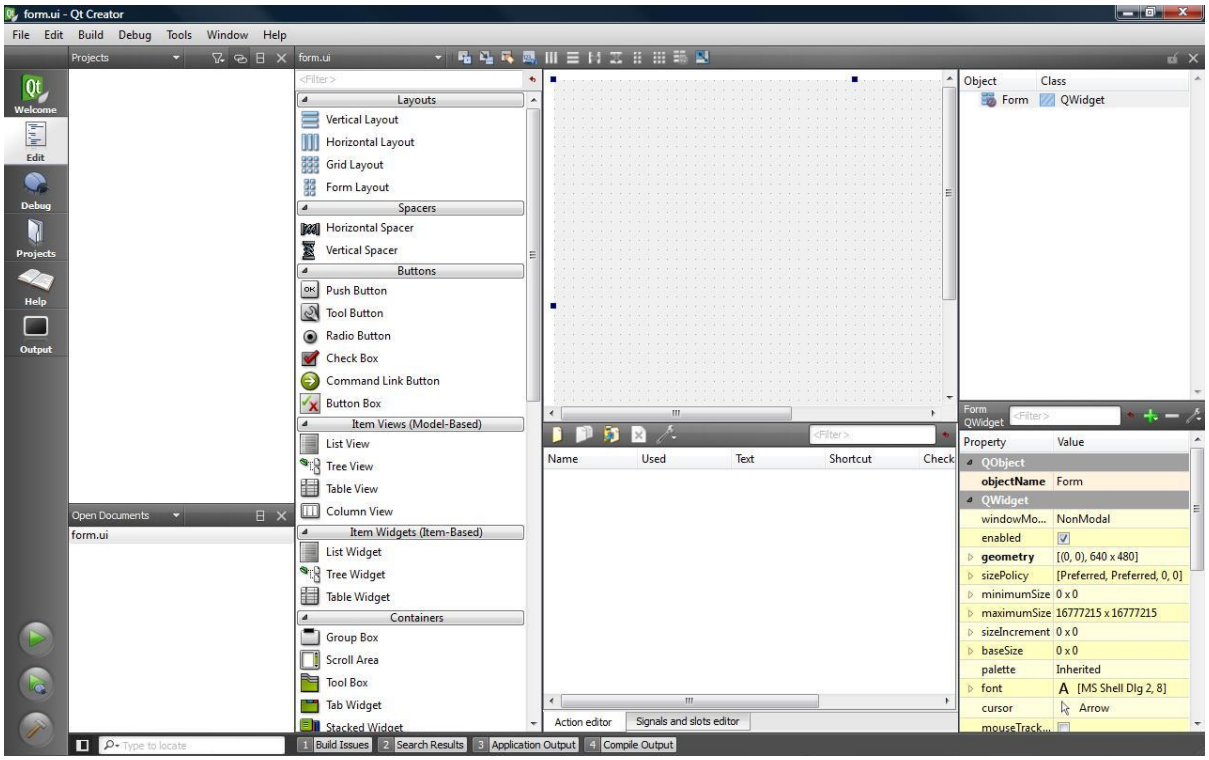

Figura 20 – Finestra di lavoro Qt.

5.10 LINUX EMBEDDED

Per lo sviluppo dei sistemi embedded sono state concepite delle distribuzioni fondate sul kernel di Linux. Questi sistemi impongono vincoli severi per quanto riguarda le prestazioni (velocità di avvio), l'occupazione della memoria di programma e anche un eventuale quantitativo di ram per il salvataggio di eventuali applicazioni aperte, e quindi il rapido utilizzo di esse senza passare per una memoria di archiviazione molto più lenta. Per la progettazione dei sistemi embedded conviene utilizzare il kernel di Linux, per ammortizzare i costi di sviluppo, i costi di licenza e perché comunque Linux, di sua natura, è un sistema portabile su qualsiasi tipo di piattaforma compilando il sorgente con i cross-compiler più adatti.
I sistemi embedded nati con questo kernel, ereditano tutte le proprietà del file system di Linux; in pratica si realizza un microcomputer dedicato per lo svolgimento di certe applicazioni, minimizzando i consumi e l'utilizzo delle risorse hardware.
Per l'integrazione del sistema Linux in una applicazione embedded vengono utilizzati dei supporti di archiviazione opportuni come ad esempio NAND Flash; che vanno dalla dimensione di qualche mega a qualche decina di mega. È infatti possibile ridurre il kernel di Linux a qualche centinaio di kilobytes, selezionando così solo le applicazioni e le librerie strettamente necessarie.

È possibile realizzare questo tipo di sistemi anche utilizzando delle versioni GNU di Linux (Gnu is Not Unix), versioni completamente gratuite costruite su quattro cardini fondamentali:

- Libertà di eseguire il programma per qualsiasi scopo
- Libertà di studiare il programma e modificarlo
- Libertà di copiare il programma in modo da aiutare il prossimo
- Libertà di migliorare il programma e di pubblicarne i suoi miglioramenti, in modo che tutta la comunità degli utenti possa trarne beneficio

Queste quattro libertà vengono garantite dalla licenza GPL (Gnu Public License).
Queste libertà non significano che i programmatori non possano chiedere dei compensi per le loro pubblicazioni intellettuali, purché rispettino le regole sopracitate.

5.11 UTILIZZO DI UN SISTEMA OPERATIVO SU SISTEMI EMBEDDED

Quali vantaggi comporta l'utilizzo di un kernel Linux, utilizzato solitamente in ambiente x86, in un sistema embedded?
Come in ogni approccio di progettazione sono presenti vantaggi e svantaggi, non esiste una soluzione assoluta nella progettazione di sistemi embedded; tutto deve essere calibrato in base a quello che bisogna realizzare, garantendo stabilità ed allo stesso tempo semplicità costruttiva minimizzando l'utilizzo delle risorse.
Vantaggi:

- ***MULTITASKING:*** Si possono far girare più applicazioni contemporaneamente, in modo che per eventuali modifiche si possa intervenire esclusivamente sui blocchi interessati.
- ***DRIVERS:*** Ogni kernel contiene al suo interno questi "software". Sono applicazioni (o meglio istruzioni) che si occupano di interfacciare l'hardware collegato al sistema. Sebbene all'interno di un kernel non siano presenti proprio i driver necessari, essi possono essere modificati, e quindi i tempi di sviluppo degli stessi sono ridotti.
- ***SVILUPPO:*** Tutte le prove necessarie del software possono essere eseguite su piattaforma x86 per poi essere scaricate su sistema embedded tramite cross-compiler. Questo garantisce tempi di sviluppo e di collaudo inferiori.
- ***STANDARDADIZZAZIONE:*** Utilizzando in pratica le stesse istruzioni per le piattaforme x86, l'apprendimento per la realizzazione di strutture embedded risulta essere molto veloce, non discostandosi molto dai sistemi x86.
- ***IMPORTAZIONE DELLE LIBRERIE:*** In base a ciò che si deve eseguire sarà necessario importare opportune librerie. Per esempio, se si vogliono utilizzare le librerie Qt si possono importare, senza doverle riscrivere da zero.

Svantaggi:

- ***REAL-TIME:*** Linux non è adatto per l'esecuzione di applicazioni real-time, a meno che non siano implementate delle opportune estensioni.
- ***RISORSE HARDWARE:*** Bisogna ricordarsi che Linux nasce per essere utilizzato su x86, quindi risulta indispensabile essere forniti di opportune risorse hardware; come RAM e memorie di massa per il caricamento del sistema operativo e rispettive librerie importate. È anche vero però che tali risorse sono sempre più accessibili a livello economico, ma questo non significa che ci si può prendere la libertà di esagerare nel dimensionamento delle struttura di memorizzazione di contorno. Il tutto deve essere fatto sempre considerando il fatto di utilizzare il minimo quantitativo di risorse.

5.12 KERNEL DI LINUX

Il kernel, in italiano nucleo, non è definibile propriamente come un software. È un gestore globale di sistema, gestisce tutto il basso livello che ha il compito di interfacciare l'esterno (hardware), con l'interno (software, o meglio interfaccia utente). Linux si appoggia su di un kernel monolitico, un nucleo costituito da un singolo programma eseguibile che gestisce tutte le componenti necessarie per il funzionamento del sistema: usb, schede di rete, schede grafiche, ecc.
È possibile caricare dei driver in runtime tramite i moduli, quindi operare direttamente sulla gestione delle periferiche senza modificare il kernel.
Parte importante di un kernel è il file system. Il supporto di memorizzazione di massa, quindi hard-disk per i PC o memoria flash (NANDFlash) per sistemi embedded, è una periferica hardware e quindi necessita di driver per il funzionamento. Nei sistemi basati su Linux esistono vari tipi di file system:

- EXT2, EXT3, RAISERFS, MINIX, JFFS, JFFS2 supportati solo da sistemi Unix
- FAT, NTFS standard supportati da Unix però nativi di Microsoft

5.13 COMPILATORE GCC

L'acronimo GCC sta per GNU Compiler Collection. È una distribuzione di compilatori che include C, C++, Objective-C, Objective-C++, Java, Fortran e Ada.
Con GCC è possibile generare codice macchina per diversi μP o μC. Il compilatore supporta tre versioni di C standard:

- ANSI C Standard (ISO/IEC 9899:1990) => C89 o C90
- (ISO/IEC 9899:1994) => C94 o C95
- (ISO/IEC 9899:1999) => C99 (non supportato pienamente da GCC)

Per quanto concerne il C++ sono supportate le seguenti versioni:

- ISO C++ standard (ISO/IEC 14882:1998)
- Le correzioni tecniche apportate nel 2003 (ISO/IEC 14882:1998)

In ogni caso è possibile selezionare la versione di C, o C++, che si usa per programmare, in modo da evitare errori o warnings.
Come precedentemente detto il GCC è in grado di compilare per diversi tipi di μP o μC, nel caso in oggetto si è utilizzata una compilazione per AVR e per ARM.

I file da processare e processati hanno le seguenti estensioni:

- .c = file da passare al preprocessore (C)
- .h = file di header (C e C++)
- .cpp = file da passare al preprocessore (C++)
- .o = file oggetto compilati (C e C++)

5.14 ECLIPSE

Eclipse è una piattaforma di sviluppo, basata sullo stile dell'open-source. È un ambiente di sviluppo integrato (IDE – Integrated Development Environment) utilizzato per produzioni software di qualunque genere. Nel caso particolare è stato utilizzato lo strumento di sviluppo inerente al Qt. Non è nemmeno necessaria l'installazione del programma, in quanto viene eseguito appoggiandosi sulla JVM (Java Virtual Machine).

Analizzando più approfonditamente Eclipse, si nota che è una collezione di plug-in. Ogni plug-in (codice) fornisce funzionalità al prodotto finale. Quindi il codice ed i plug-in utilizzati saranno installati nel computer dove si intende utilizzare il prodotto finito. Tutti i plug-in del prodotto finito sono raggruppati in una classe che viene definita come *caratteristiche del programma*.
L'integrazione di tutte queste funzionalità aggiuntive sono semplici da integrare, e una volta integrate nel prodotto possono essere aggiornate costantemente, senza la necessità di modificare parti del codice scritto. L'organizzazione delle applicazioni scritte con Eclipse è ad albero, quindi c'è una *root* principale, e le *leaves* (foglie) saranno tutti i plug-in.

Per la programmazione in Qt è stata necessaria l'installazione di un'opportuna integrazione Qt disegnata appositamente per Eclipse. Chi volesse sviluppare il codice e l'interfaccia grafica in ambiente Windows deve installare il pacchetto MinGW, una versione di GCC per l'OS di Microsoft. È fondamentale impostare l'OS in modo che nella PATH di sistema siano presenti tutte le utilità di compilazione sia del MinGW o GCC, a seconda dell'OS e le utilità di compilazione di Qt.

Con Eclipse è possibile creare le interfacce grafiche, omettendo così l'utilizzo dell'ambiente proprietario di Qt, il Qt Creator. A parer personale è vantaggioso poter modificare codice e interfacce grafiche all'interno dello stesso applicativo di sviluppo; per semplici ragioni di comodità e ordine nella gestione del progetto.

L'ambiente di lavoro Eclipse con le librerie Qt integrate si presenta all'utente in questo modo:

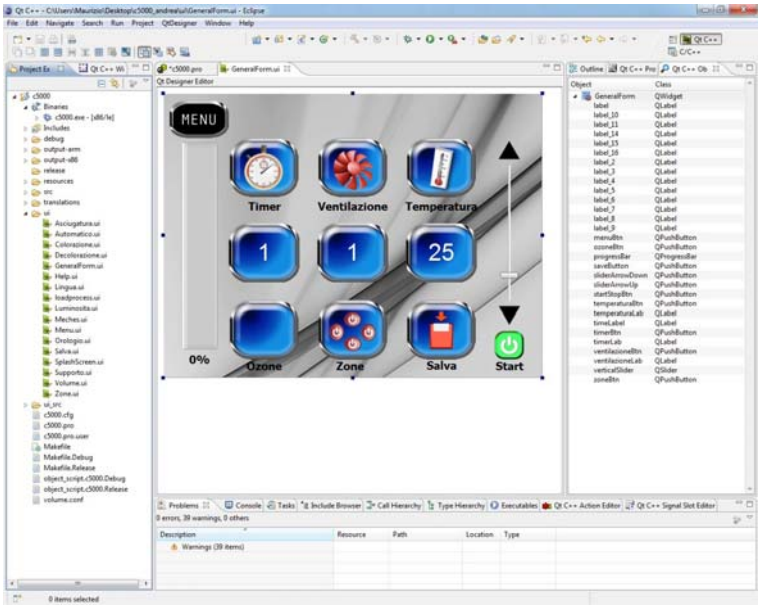

Figura 21 – Finestra di lavoro Eclipse.

Si noti:

- l'organizzazione del progetto alla sinistra della finestra di lavoro
- la UI (User Interface) al centro, per modificarla basta trascinare gli oggetti disponibili nel menu Qt C++ Widgets, adiacente ai file di progetto
- la descrizione degli oggetti della finestra, il tipo e le loro proprietà
- in basso è presente una consolle, molto utile in fase di debug, in quanto riporta avvertenze (Warning) ed errori nella costruzione delle classi

5.15 PROGETTAZIONE INTERFACCIA GRAFICA

Per iniziare lo sviluppo dell'interfaccia grafica, sono state seguite tutte le specifiche di funzionamento volute dal cliente. Il cliente ha inoltre fornito i clipart da utilizzare per i vari pulsanti e le videate da visualizzare sul touch screen nonché le specifiche di trattamento standard da inserire in memoria.

Il primo passo compiuto è stato quello di realizzare l'impostazione grafica, quindi la creazione delle finestre con i relativi pulsanti di funzionamento. La spiegazione sarà affrontata fornendo un esempio pratico. Si consideri la schermata iniziale dei Menu.

Figura 22 – Interfaccia grafica C5000..

La realizzazione di questa consiste nella creazione degli elementi da posizionare nella finestra, in questo particolare caso sono stati creati otto QPushButton e dieci QLabel . Per dichiarare i pulsanti inseriti come QPushButton, si dovrà inserire un pulsante di tipo Push Button e successivamente cambiare le sue proprietà nel menu Qt C++ Object Inspector (Qt C++ OI). Analogamente di dovrà fare per le etichette: sarà sufficiente creare una Text Edit e cambiare la sua definizione nel Qt C++ OI in QLabel. Alla pressione dei pulsanti ci si sposterà nei relativi sottomenu. La creazione dell'oggetto è realizzata per il momento solo mediante trascinamento dei pulsanti sulla finestra. Tale procedura include anche nella realizzazione del codice automatica da parte di Eclipse del file cpp, che è l'implementazione in codice della grafica appena realizzata. L'output cpp è il seguente:

```
#include "Include.h"

/* Personalizzazione finestra (Type & Hints) */

MenuDef::MenuDef()
{
	setupUi(this);
	setWindowFlags(Qt::CustomizeWindowHint | Qt::FramelessWindowHint |
	Qt::Window);
	move(0,0);
}
```

```
/* Gestione pulsanti menu */

void MenuDef::on_luminositaBtn_pressed()
{}

void MenuDef::on_zoneBtn_pressed()
{}

void MenuDef::on_volumeBtn_pressed()
{}

void MenuDef::on_orologioBtn_pressed()
{}

void MenuDef::on_linguaBtn_pressed()
{}

void MenuDef::on_manualeBtn_pressed()
{}

void MenuDef::on_automaticoBtn_pressed()
{}

void MenuDef::on_helpBtn_pressed()
{}
```

5.15.1 PERSONALIZZAZIONE FINESTRA – TYPE & HINTS

In Qt è possibile personalizzare in tutti i loro dettagli le finestre. Per quanto concerne la voce TYPE è possibile stabilire il tipo di finestra. Sono disponibili quattro tipi di finestra: Window, Dialog, Sheet, Drawer, Widget, Popup, Tool, ToolTip, SplashScreen, Desktop e SubWindow. Le finestre di tipo Window sono le classiche finestre senza pulsanti di riduzione/ingrandimento e chiusura, in pratica è presente solo la cornice. Le Sheets sono delle Dialog speciali (usate in Mac), di solito usate per la notifica di chiusura delle applicazioni; sono un esempio le tipiche finestre che chiedono se si desidera salvare un programma prima dell'uscita. Negli altri casi i TYPE di finestra si esplicano da soli, come ad esempio le finestre Popup (vedi popup quando si naviga in internet – finestre che si aprono automaticamente, spesso indesiderate) o le SubWindow che vengono spesso considerate finestre figlia, in pratica sono finestre che hanno una finestra madre a cui fanno capo. È possibile inoltre modificare gli HINT della finestra, sono delle opzioni aggiuntive che permettono di aggiungere i pulsanti di riduzione/ingrandimento e chiusura, menu di sistema, titolo alla finestra, togliere il titolo alla finestra, nascondere i bordi e la priorità della finestra. Nel caso in analisi l'opzione TYPE utilizzata è la Qt::Window (finestra semplice) e le HINT utilizzate sono Qt::FramelessWindowHint (finestra senza bordi) e Qt::CustomizeWindowHint (copertura titolo della finestra).

Nella seconda parte del file cpp c'è la dichiarazione dei componenti della finestra. Come precedentemente detto i pulsanti sono tutti di tipo QPushButton impostati in modo da essere sensibili al "fronte di discesa", quindi il pulsante esegue l'operazione richiesta solo in caso venga rilasciato dopo la sua pressione. Finché il pulsante è premuto nessuna operazione viene svolta.
Il costrutto è on_PULSANTE_pressed, letteralmente tradotto come "alla pressione del PULSANTE". In questo modo vengono creati dei metodi, entro i quali si devono inserire le operazioni da svolgere alla pressione dei pulsanti.

Come in tutti i programmi C++, i file sorgente sono accompagnati da un file header.
Qui sotto il rispettivo file di header:

```
#ifndef __MENUDEF_H__
#define __MENUDEF_H__

#include <QtGui/QWidget>
#include <QDialog>
#include "ui_Menu.h"

class MenuDef: public QDialog, public Ui::Menu
{
	Q_OBJECT

	public:
		MenuDef();

	private slots:
		void on_zoneBtn_pressed();
		void on_volumeBtn_pressed();
		void on_orologioBtn_pressed();
		void on_linguaBtn_pressed();
		void on_manualeBtn_pressed();
		void on_automaticoBtn_pressed();
		void on_luminositaBtn_pressed();
		void on_helpBtn_pressed();
};
#endif
```

È presente la tipica e obbligatoria dichiarazione dei metodi del sorgente principale e l'importazione dei file Include.

5.16 PROGRAMMAZIONE CONCORRENTE

Come precedentemente descritto i sistemi embedded vengono gestiti da un sistema operativo ad hoc. Quindi, come tutti i sistemi operativi, si trova a dover gestire nello stesso tempo diverse istruzioni provenienti da metodi appartenenti a classi diverse.
Si parla quindi di programmazione concorrente. Questo tipo di gestione nasce dall'utilizzo del dispositivo da parte dell'utente finale. Per rendere un senso pratico della cosa si pensi ad un moderno telefono cellulare, dotato delle ultimissime tecnologie. Sarebbe inammissibile rilevare dei blocchi di sistema dovuti al sovraccarico di operazioni nel task. Il task, spiegando meglio, è l'insieme delle operazioni che sta eseguendo simultaneamente il μP; quindi si può ascoltare un mp3 finché si stanno inviando dati via seriale o bluetooth.
Quindi bisogna garantire l'esecuzione di più processi durante lo stesso lasso di tempo. Comunemente, quando si va a comprare un μP per un PC domestico o industriale, è possibile acquistare sistemi multicore, appunto per ottimizzare l'esecuzione degli applicativi, ormai sempre più pesanti.
La programmazione concorrente consta quindi nella esecuzione in parallelo di più programmi o parti dello stesso programma. La parte più difficile nella realizzazione di questo, sta nella coordinazione di lettura e scrittura nelle risorse condivise del sistema quali RAM, EEPROM o periferiche esterne. Per realizzare ciò non esiste un metodo univoco, uno dei metodi più utilizzati è la realizzazione di thread.

Con questa realizzazione l'utente non si accorge mai di ciò che sta realmente elaborando la macchina, l'interfaccia grafica non viene bloccata per terminare le operazioni in esecuzione, quindi la stabilità è assolutamente garantita.
Per realizzare la sincronizzazione è stata utilizzata una macchina a stati finiti di Moore, dove lo stato futuro dipende esclusivamente dallo stato attuale della macchina.
Gli stati rappresentano la situazione istantanea della macchina.
Di seguito viene riportata la macchina a stati finiti che realizza la sincronizzazione tra i diversi processi.

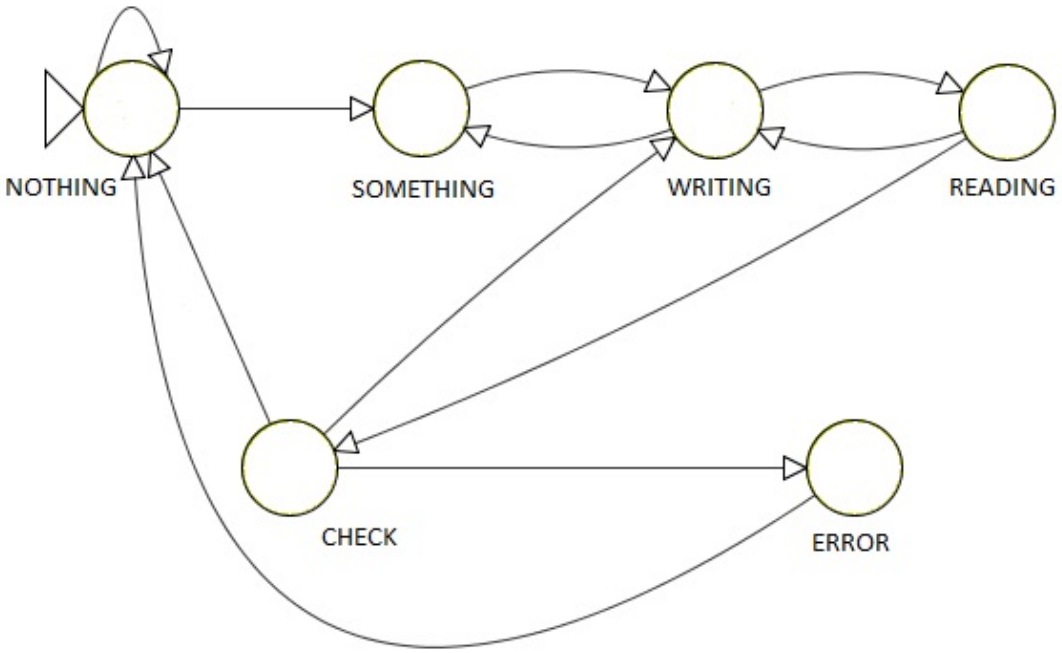

Figura 23 – MSF per la gestione di lettura e scrittura.

Gli stati raggiungibili della macchina sono i seguenti:

- ***NOTHING***
- ***SOMETHING***
- ***WRITING***
- ***READING***
- ***CHECK***
- ***ERROR***

La macchina funziona così (lo stato iniziale della MSF è ***NOTHING***):

- Dallo stato ***NOTHING*** è raggiungibile ***SOMETHING*** solo nel caso in cui venga richiesta un'operazione di scrittura. Il sistema quindi blocca altre eventuali operazioni di scrittura e mette in lista le nuove operazioni da processare, evitando così errori di scrittura nei registri. Se non venissero chieste delle operazioni di scrittura significa che nessuna parte di programma sta interagendo con le risorse condivise di sistema, i buffer di lettura/scrittura sono liberi. Sostanzialmente la macchina rimane in ascolto in attesa di istruzioni.
- Lo stato ***SOMETHING*** raggiunge ***WRITING*** ad ogni richiesta di scrittura.

- Dallo stato ***WRITING*** sono raggiungibili ***READING*** e ***SOMETHING***. Saranno impostati il buffer RX/TX ed un timer che servirà per garantire la corretta operazione di scrittura.
- Dallo stato ***READING*** sono raggiungibili ***CHECK*** e ***WRITING***. Lo stato ***CHECK*** si raggiunge se sono stati ricevuti tutti i byte. Si passerà in ***WRITING*** se il timer inizializzato da ***WRITING*** stesso non fosse ancora finito.
- Dallo stato ***CHECK*** sono raggiungibili ***ERROR***, ***WRITING*** e ***NOTHING***. Per sapere in quale stato portarsi viene eseguito un controllo sui dati inviati di tipo "*CHECKSUM*" (verifica di integrità del dato). Se il dato processato è corretto allora si può portare la macchina in stato di idle, quindi ***NOTHING***. Se ciò non dovesse accadere allora per al massimo 50 tentativi si ritorna allo stato ***WRITING***, in modo che i dati possano essere inviati e processati nuovamente. Se la parola ricevuta non dovesse essere corretta dopo i 50 tentativi prestabiliti, allora la macchina viene impostata in ***ERROR***, impostando a '1' il flag di errore.
- Dallo stato ***ERROR*** la macchina viene resettata, quindi i tentativi di invio dati, precedentemente usati da ***CHECK***, vengono ripristinati a 50 e viene spento il flag di errore. La macchina torna a ***NOTHING***.

Il passaggio da uno stato all'altro della MSF è stabilito essenzialmente dalle operazioni di lettura e scrittura in RAM e in EEPROM. Ad ogni richiesta di queste operazioni lo stato passa prima in ***SOMETHING*** e poi in ***WRITING*** per i motivi sopracitati. Le implementazioni di queste funzionalità risiedono nel basso livello e si ottengo smontando e rimontando la partizione all'interno della NAND FLASH.

In appendice è stato inserito il sorgente in C, che realizza la MSF.

5.17 GESTIONE OPERAZIONI DI LETTURA E SCRITTURA

La NAND FLASH (Samsung K9F2G08U0M-PCB0 da 256 Mbyte) è la memoria di archiviazione scelta per la memorizzazione del firmware, essa include la gestione sia in alto che in basso livello. Questa memoria è indicata per i sistemi di archiviazione dati, per motivi puramente costruttivi. Infatti l'architettura NAND permette l'accesso sequenziale ai dati, a differenza di una memoria NOR che è ad accesso casuale (RAM) ed è indicata per esecuzioni veloci come le istruzioni di un μP. Per questo motivo le istruzioni contenuto nella memoria NAND non possono essere eseguite all'interno di essa, è necessario utilizzare una memoria RAM; in particolare sono state usate due memorie da 32 Mbyte Samsung K4S561632H-UC75.

La gestione della NAND è gestita con le procedure di formattazione standard di UNIX (file system), quella utilizzata in questo caso è la JFFS2 con la creazione di una opportuna area di swap (pari alla dimensione del quantitativo di RAM: 64 Mbyte). Per fare un parallelo, l'architettura x86 solitamente utilizza il file sytem EXT. JFFS2 è un file system open source che insieme allo YAFFS permette di gestire i supporti NAND.
Le operazioni di scrittura e lettura nella NAND sono molto delicate. Non è sufficiente montare la partizione una sola volta in modalità di scrittura/lettura, dopo aver fatto vari tentativi sono emersi problemi di blocco dell'applicazione; alto e basso livello non comunicavano più e la gestione dei carichi si bloccava, in quanto si interrompeva anche la comunicazione seriale.

Ad ogni operazione di lettura, se la partizione non è montata già come "read only", è necessario smontare la partizione attuale e rimontarla con i privilegi di sola lettura.

Analoga procedura è utilizzata nei processi di scrittura, la partizione è montata con il flag di "write". Queste operazioni vengono eseguite in automatico ad ogni richiesta di lettura e scrittura in RAM o EEPROM, queste operazioni fondamentali seguono l'organizzazione della MSF spiegata nel paragrafo precedente.

5.18 INTERFACCIA GRAFICA COLLAUDO SCHEDA DI POTENZA J017

Per il test della scheda di potenza del C5000 è stata progettata una UI semplice per la prova delle uscite.

Figura 24 – Interfaccia grafica di collaudo scheda J017 (gestione carichi di C5000).

I pulsanti Zona 1, Zona 2, Zona 3 e Zona 4 accendono le quattro lampade riscaldatrici. Il pulsante Ventilazione abilita la barra di scorrimento laterale. Essa prevede cinque passi (scatti) per default, che corrispondono alle cinque velocità della ventola. Il pulsante Aux serve per accendere un carico ausiliario (per semplicità di collaudo basta collegarci una lampada standard da 40 Watt) e Ozono accende una lampada ad ozono per la sterilizzazione.
Per verificare il corretto funzionamento della J017, sarà sufficiente osservare solo la risposta dei carichi al momento della sollecitazione via UI.
Per eseguire il test, la UI è gestita direttamente da PC e i comandi vengono inviati via seriale (SERAMI) alla J017. I carichi collegati per quanto riguarda ventola, lampade riscaldatrici e lampada ad ozono sono quelli montati sul C5000.

Capitolo 6
CONCLUSIONI

Queste due esperienze non hanno avuto l'obbiettivo di realizzare un progetto dall'inizio alla fine, l'intento era quello capire l'ottica del lavoro di squadra. La realizzazione di un progetto presenta vari aspetti da sviluppare, e per ogni aspetto c'è uno specialista che se ne occupa; dal progetto hardware allo sviluppo del firmware. Le chiavi di un buon lavoro di quadra sono racchiuse in due parole fondamentali: organizzazione e sinergia. Per fornire un esempio pratico si pensi anche al solo commentare il codice finché si programma; potrebbe non essere di grande aiuto al programmatore che lo sta scrivendo ma potrà essere in futuro di fondamentale importanza per una modifica da parte di altri tecnici.
Altro punto molto importante è sapere quello che si deve fare, tutto deve essere progettato e collaudato a compartimenti stagni, quindi quando tutto funziona si può procedere alla messa a punto globale del progetto.
È stata compresa la differenza tra la programmazione di microcontrollori a 8 e 32 bit. In base a quello che bisogna realizzare si usa un apposito supporto di calcolo. Un microcontrollore del valore di qualche euro, permette di realizzare applicativi di prestazioni quasi paragonabili a quelle di un personal computer di qualche anno fa. Infatti i sistemi embedded generalmente non necessitano di particolari prestazioni.
Gli obbiettivi posti all'inizio del tirocinio sono stati pienamente conseguiti pur essendo la prima esperienza in una media azienda. Le conoscenze possedute sono risultate buone nel settore elettronico/componentistico, discrete nel reparto informatico. C'è stata la necessità di imparare nuovi linguaggi di programmazione per adeguarsi alle specifiche di Micronova. Resta comunque una grandissima soddisfazione sapere di aver partecipato a progetti che portano al loro interno parte del mio lavoro, e che vengono prodotti in grandissime quantità e distribuiti in tutta Italia e nel mondo. Il bagaglio culturale costruito sarà sicuramente utile nel mio prossimo futuro e sicuramente porterà un vantaggio non indifferente.

Capitolo 7
RINGRAZIAMENTI

I ringraziamenti sono rivolti innanzitutto alla mia famiglia, (mio padre Mauro, mia madre Lorenza, mia sorella Angela Maria e i nonni materni Enrico e Teresa), che mi ha sempre sostenuto moralmente in questi quattro anni di studio. Ringrazio i compagni di corso, nonché ormai miei amici, che hanno condiviso con me lunghe e dure battaglie per la preparazione degli esami (Daniele Caliolo, Marco Maffei, Marcomattia Mocellin e Gabriele Miorandi), gli amici di sempre per essere sempre stati presenti anche quando ormai chiunque mi avrebbe considerato senza speranze (Fabio Benetollo, Mattia Perin, Nicolas Milani, Marco Salvador). Un pensiero va rivolto alla famiglia Caliolo (Carlo, Maria Angela e Sara) per avermi ospitato a casa loro innumerevoli volte durante la preparazione degli esami con Daniele. Ringrazio la Micronova, quindi la Fam. Benetti tutta, per aver posto fiducia in me e avermi fatto il regalo di imparare qualcosa che mi servirà per costruire un futuro; un pensiero particolare va a Giulio, per avermi insegnato molto e avermi sostenuto in momenti davvero difficili. È doveroso ringraziare anche i Sigg. Ceriotti e Dell'Acqua per avermi dato la possibilità di divulgare parti dei contenuti tecnici del progetto C5000. Ringrazio Manuel Segrè, per avermi dato un lavoro e per la sua amicizia. Sentiti ringraziamenti spettano al mio relatore, il prof. Meneghesso; egli ha sempre dimostrato disponibilità, puntualità ed interessamento durante la preparazione di questo elaborato. Infine desidero ringraziare dal profondo del cuore la mia Valentina, la quale ha saputo sostenermi nell'ultima parte della mia avventura universitaria, aiutandomi a superare i momenti che sono stati, per ora, quelli più difficili della mia vita.

Grazie anche a tutte quelle persone che sono entrate nella mia vita per poi non tornare più, tra le quali ricordo i miei nonni paterni Giuseppe e Iva che hanno saputo insegnarmi tanto finché hanno corso nella vita insieme a me.

APPENDICE

A.1 DESCRIZIONE PIEDINATURA I023

- ***VCC***
  Tensione di alimentazione.

- ***GND***
  Massa.

- ***PORT A (PA7…PA0)***
  Può essere utilizzato come port di I/O ad 8 bit oppure come ingresso per l'ADC. Tutti i pin hanno un pull-up interno e sono tri-state.

- ***PORT B (PB7…PB0)***
  E' un port di I/O da 8 bit tri-state e ogni singolo pin ha un pull-up interno.
  Se viene utilizzato come ingresso, i pin del PORT B che hanno un pull-down esterno, forniranno corrente se il pull-up dei resistori sarà attivato.
  Il PORT B gode di ulteriori funzioni speciali di seguito riportate:

PB7	SCK (Clock Bus SPI Seriale)
PB6	MISO (Master Bus Input/Slave di uscita)
PB5	MOSI (Master Bus Output/Slave di ingresso)
PB4	SS (Selezione Slave ingesso, lavora in logica negata)
PB3	AIN1 (Comparatore Invertente Analogico di ingresso)
	OC0 (Comparatore di uscita verifica soglia Timer/Counter0)
PB2	AIN0 (Comparatore non invertente di ingresso)
	INT2 (Interrupt Esterno 2 di ingresso)
PB1	T1 (Contatore Esterno di ingresso Timer/Counter1)
PB0	T0 (Contatore Esterno di ingresso Timer/Counter0)
	XCK (Clock di I/O esterno USART)

- ***PORT C (PC7…PC0)***
  E' un port di I/O da 8 bit tri-state e ogni singolo pin ha un pull-up interno.
  Se viene utilizzato come ingresso, i pin del PORT B che hanno un pull-down esterno, forniranno corrente se il pull-up dei resistori sarà attivato.
  Se l'interfaccia JTAG è abilitata, le resistenze di pull-up dei pin 5, 3 e 2 del PORT C saranno attivate anche in concomitanza di un evento di reset.

Il PORT C gode di ulteriori funzioni speciali di seguito riportate:

PC7	TOSC2 (Pin 2 Oscillatore del Timer)
PC6	TOSC1 (Pin 1 Oscillatore del Timer)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Linee Bus della seriale a 2 fili)
PC0	SCL (Linea Clock della seriale a 2 fili)

- ***PORT D (PD7…PD0)***
  E' un port di I/O da 8 bit tri-state e ogni singolo pin ha un pull-up interno.
  Se viene utilizzato come ingresso, i pin del PORT D che hanno un pull-down esterno, forniranno corrente se il pull-up dei resistori sarà attivato. Il PORT D gode di ulteriori funzioni speciali di seguito riportate:

PD7	OC2 (Comparatore di uscita verifica soglia Timer/Counter 2)
PD6	ICP1 (Cattura ingresso Timer/Counter1)
PD5	OC1A (Comparatore di uscita verifica soglia A Timer/Counter 1)
PD4	OC1B (Comparatore di uscita verifica soglia B Timer/Counter 1)
PD3	INT1 (Ingresso Esterno Interrupt 1)
PD2	INT0 (Ingresso Esterno Interrupt 0)
PD1	TXD (Uscita USART)
PD0	RXD (Ingresso USART)

- ***RESET (funziona in logica negata)***
  Quando il segnale collegato al reset è "0 logico" per un tempo più grande della durata minima dell'impulso campionabile (1,5 us) correttamente dell'integrato, resetta l'integrato.

- ***XTAL1***
  Ingresso dell'amplificatore invertente dell'oscillatore e ingresso del clock interno del circuito.

- ***XTAL2***
  Uscita dell'amplificatore invertente dell'oscillatore.

- ***AVCC***
  È la tensione di alimentazione per il PORTA e l'ADC. Deve essere connessa esternamente a VCC anche se l'ADC non è utilizzato. Se si utilizza l'ADC, il pin AVCC deve essere connesso a VCC tramite un filtro passa-basso.

- ***AREF***
  È il riferimento analogico per l'ADC.

A.2 SPECIFICHE TECNICHE

- ***ALIMENTAZIONE***

Tensione di alimentazione	230Vca ± 15%, 50/60 Hz
Consumo max (esclusa console e utilizzatori)	50 mA
Consumo max (console collegata esclusi utilizzatori)	55 mA

- ***INGRESSI***

Termocoppia temperatura fumi	Termocoppia tipo J
Termostato esterno	Contatto n.a.
Sonda NTC temperatura ambiente NTC	10 kΩ
Sonda NTC temperatura acqua NTC	10 kΩ
Sonda NTC temperatura pellet NTC	10 kΩ
Console	-
Encoder velocità rotazione estrattore fumi	-
Scheda opzionale cronotermostato	-
Termostato di sicurezza generale	230Vca
Pressostato di sicurezza	230Vca
Connessione seriale	(da utilizzare con adattatore)

- ***USCITE***

Aspiratore fumi (con reg. a controllo di fase)	230 Vca (TRIAC)
Scambiatore aria n°3 (con reg. a controllo di fase)	230 Vca (TRIAC)
Motore coclea	230 Vca (TRIAC)
Scambiatore aria n°1 (con reg. a controllo di fase)	230 Vca (TRIAC)
Scambiatore aria n°2 (con reg. a controllo di fase)	230 Vca (TRIAC)
Candeletta	230 Vca (Contatto)

- ***SPECIFICHE AMBIENTALI***

Temperatura ambiente operativa	da 0°C a +60°C
Temperatura di immagazzinamento	da -10°C a +60°C
Umidità relativa massima (senza condensa)	95%

- ***SPECIFICHE MECCANICHE***

Dimensioni scheda (LxPxH)	(125 x 101 x 35) mm
Peso	250 g circa

A.3 SCHEMA CONNESSIONI I023

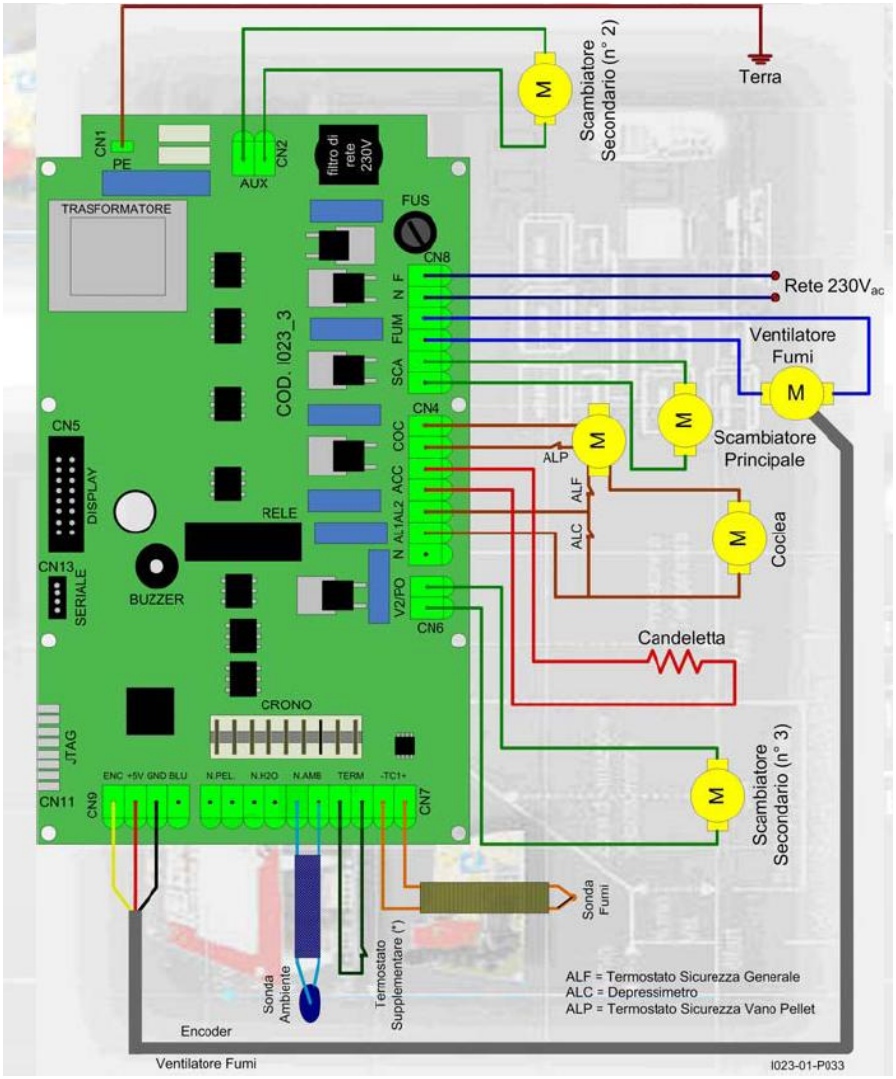

Figura 25 – Schema connessioni I023.

A.4 TABELLA I/O IO23

Connettore	Pin	Etichetta	Descrizione
CN1	-	-	Terminale a innesto rapido di terra
CN2	1 - 2	AUX	Uscita ventilatore aria n°2
CN3	-	OROLOG	Connettore cronotermostato opzionale
CN4	1	N	Neutro
	2	AL1	Ingresso allarme termometro di sicurezza (230Vca)
	3	AL2	Ingresso allarme pressostato di sicurezza (230Vca)
	4 - 5	ACC	Uscita candeletta (230Vca)
	6 - 7	COC	Uscita motore coclea (230Vca)
CN5	-	DISPLAY	Connettore per la console
CN6	1 - 2	V2/PO	Uscita ventilatore aria n°3 (circolatore)
CN7	1 - 2	N. PEL	Ingresso sonda temperatura aria/pellet (non utilizzato)
	3 - 4	N. H20	Ingresso sonda temperatura acqua (non utilizzato)
	5 - 6	N. AMB	Ingresso sonda temperatura ambiente
	7 - 8	TERM	Ingresso termostato esterno
	9 - 10	-TC+	Ingresso termocoppia fumi
CN8	1 - 2	SCAM	Uscita ventilatore scambiatore n° 1
	3 - 4	FUMI	Uscita ventilatore fumi
	5 - 6	N - F	Alimentazione scheda (230Vca)
CN9	1	ENC	Ingresso encoder ventilatore fumi
	3	+5V	Alimentazione encoder a + 5V
	4	GND	Comune ingresso encoder
	5	BLUE	Non utilizzato
CN12	-	JTAG	Connettore programmazione di fabbrica
CN13	-	SERIALE	Connessione seriale da usare con adattatore

A.5 PROGRAMMAZIONE CONCORRENTE – CODICE IN C

```
#include "Include.h"

/*definizioni*/
#define RSERAMIMASTER_BUFFER_SIZE           4    // dimensione buffer modulo
#define RSERAMIMASTER_ACCEPTED_COMMAND_NUM 4    // numero di comandi accettati
#define RSERAMIMASTER_CMD_MASK              0x1F// maschera comando Serami
#define RSERAMIMASTER_READ_RAM_BYTE         0x00// codice comando lettura byte da
ram
#define RSERAMIMASTER_READ_EEPROM_BYTE      0x20// codice comando lettura byte da
eeprom
#define RSERAMIMASTER_WRITE_RAM_BYTE        0x80// codice comando scrittura byte
su ram
#define RSERAMIMASTER_WRITE_EEPROM_BYTE     0xa0// codice comando scrittura byte
su eeprom
```

```
#define RSERAMIMASTER_RETRIES 50

static u8 iFrameBuffer[RSERAMIMASTER_BUFFER_SIZE]; // buffer di trasmissione
static u8 iFrameBufferRx[RSERAMIMASTER_BUFFER_SIZE]; // buffer di trasmissione
static BOOL iStatus = 0;
static u8 iByteNumToSend = 0;
static u8 iChkSum;
static u8 iRetries = RSERAMIMASTER_RETRIES;
static u8* iDataRxPtr;
static u8 iDataDummy;
static BOOL iErrorFlag;

static BOOL iIsRead; /* =1 if reading, =0 if writing */

int errCounter = 0;

static u8 RSeramiMasterCalculateCheck(u8 aLength)
{
	u8 i, fCheck;

	for (i = 0, fCheck = 0; i < aLength; i++)
		fCheck += iFrameBuffer[i];

	return fCheck;
}

void RSeramiMasterInit(void)
{
	iDataRxPtr = &iDataDummy;
}

void RSeramiMasterHandler(void)
{
	switch (iStatus)
	{
		case RSERAMIMASTER_NOTHING:
			iRetries = RSERAMIMASTER_RETRIES;
			iErrorFlag = FALSE;
			break;
		case RSERAMIMASTER_SMTHING:
			iRetries = RSERAMIMASTER_RETRIES;
			iErrorFlag = FALSE;
			break;
		case RSERAMIMASTER_WRITING:
			memset(iFrameBufferRx, 0, 4);
			RUsartSendString(iFrameBuffer, iByteNumToSend, 0);
			RUsartFlushRxBuffer(0);
			iStatus = RSERAMIMASTER_READING;
			TIMER(6) = 5;
			break;
		case RSERAMIMASTER_READING:
		{
			int fLen;
			if ((fLen = RUsartGetRxByteNumber(0)) > 1)
			{
				RUsartGetString(iFrameBufferRx, fLen, 0);
				iStatus = RSERAMIMASTER_CHECK;
			}
			else
			{
				if (!TIMER(6))
				{
					iStatus = RSERAMIMASTER_WRITING;
```

```
                        }
                  }
            }
                  break;
            case RSERAMIMASTER_CHECK:
            {
                  u8 chk_sum;

                  if(iIsRead)
                        chk_sum = iChkSum + iFrameBufferRx[1];
                  else
                        chk_sum = iChkSum;

                  if (iFrameBufferRx[0] == chk_sum)
                  {
                        *iDataRxPtr = iFrameBufferRx[1];

                        iStatus = RSERAMIMASTER_NOTHING;
                  }
                  else
                  {
                        if (iRetries)
                        {
                              iRetries--;
                              iStatus = RSERAMIMASTER_WRITING;
                        }
                        else
                        {
                              errCounter++;
                              iErrorFlag = TRUE;
                              iStatus = RSERAMIMASTER_ERROR;
                        }
                  }
            }
                  break;
            case RSERAMIMASTER_ERROR:
                  iErrorFlag = FALSE;
                  iRetries = RSERAMIMASTER_RETRIES;
                  iStatus = RSERAMIMASTER_NOTHING;
                  break;
            default:
                  iStatus = RSERAMIMASTER_NOTHING;
                  break;
      }
}

void RSeramiMasterRefresh(void)
{}

void RSeramiMasterWriteRam(u16 aAddress, u8 aValue)
{
      iStatus = RSERAMIMASTER_SMTHING;
      iDataRxPtr = &iDataDummy;
      iFrameBuffer[0] = RSERAMIMASTER_WRITE_RAM_BYTE | ((aAddress >> 8) &
RSERAMIMASTER_CMD_MASK);
      iFrameBuffer[1] = (u8) aAddress;
      iFrameBuffer[2] = aValue;
      iChkSum = RSeramiMasterCalculateCheck(3);
      iFrameBuffer[3] = iChkSum;
      iByteNumToSend = 4;
      iIsRead = 0;
      iStatus = RSERAMIMASTER_WRITING;
}
```

```
void RSeramiMasterWriteEeprom(u16 aAddress, u8 aValue)
{
	iStatus = RSERAMIMASTER_SMTHING;
	iDataRxPtr = &iDataDummy;
	iFrameBuffer[0] = RSERAMIMASTER_WRITE_EEPROM_BYTE | ((aAddress >> 8) &
RSERAMIMASTER_CMD_MASK);
	iFrameBuffer[1] = (u8) aAddress;
	iFrameBuffer[2] = aValue;
	iChkSum = RSeramiMasterCalculateCheck(3);
	iFrameBuffer[3] = iChkSum;
	iByteNumToSend = 4;
	iIsRead = 0;
	iStatus = RSERAMIMASTER_WRITING;
}

void RSeramiMasterReadRam(u16 aAddress, u8* aDataPtr)
{
	iStatus = RSERAMIMASTER_SMTHING;
	iFrameBuffer[0] = RSERAMIMASTER_READ_RAM_BYTE | ((aAddress >> 8) &
RSERAMIMASTER_CMD_MASK);
	iFrameBuffer[1] = (u8) aAddress;
	iChkSum = RSeramiMasterCalculateCheck(2);
	iByteNumToSend = 2;
	iDataRxPtr = aDataPtr;
	iIsRead = 1;
	iStatus = RSERAMIMASTER_WRITING;
}

void RSeramiMasterReadEeprom(u16 aAddress, u8* aDataPtr)
{
	iStatus = RSERAMIMASTER_SMTHING;
	iFrameBuffer[0] = RSERAMIMASTER_READ_EEPROM_BYTE | ((aAddress >> 8) &
RSERAMIMASTER_CMD_MASK);
	iFrameBuffer[1] = (u8) aAddress;
	iChkSum = RSeramiMasterCalculateCheck(2);
	iByteNumToSend = 2;
	iDataRxPtr = aDataPtr;
	iIsRead = 1;
	iStatus = RSERAMIMASTER_WRITING;
}

BOOL RSeramiMasterIsFree(void)
{ if (iStatus) return FALSE; else return TRUE; }

BOOL RSeramiMasterIsInError(void)
{ return iErrorFlag; }

void RSeramiMasterReset(void)
{ iStatus = RSERAMIMASTER_NOTHING; }
```

BIBLIOGRAFIA

- Patterson and Hennessy: **Computer Organization and Design. The Hardware/Software Interface**
- Dominic Sweetman: **See MIPS Run**
- Andrew Schulman: **Microprocessors From the Programmer's Perspective Review**
- Jan M. Rabaey: **Circuiti Integrati Digitali - l'ottica del progettista**
- Daniel Bovet, Marco Cesati: **Understanding the Linux Kernel**
- IBM: **Microprocessors History**
- **Using the GNU Compiler Collection**
- **Documentazione On Line QT**
- **Data Sheet – ATMega32**
- **Data Sheet – T91SAM9263**
- **Data Sheet Controller USB – SP2525A**
- **Data Sheet Controller Touch Screen – ADS7843E**
- **Data Sheet Controller Audio AC97 – AD1981B**
- **Data Sheet Amplificatore Audio – SSM2211**

INDICE DELLE IMMAGINI