UNIVERSITÀ DI PADOVA  FACOLTÀ DI INGEGNERIA

Relazione tirocinio

ISIC:

Scrittura del firmware per un controllore destinato all'automation building

Laureando: Riccardo Cusatelli

Relatore: Daniele Vogrig

Corso di laurea in Ingegneria Elettronica

Anno accademico 2009 / 2010

Data laurea: Marzo 2010

Indice

Capitolo

1 Introduzione

1.1 – ISIC, una prima panoramica.

Il lavoro che ho compiuto durante il tirocinio presso la Intellisys s.r.l. è stato la scrittura del firmware per una scheda elettronica, destinata alla *building automation*, basata su un processore ARM7 della NXP, modello LPC2388. Il nome del progetto è “ISIC”.

Il compito principale di ISIC sarà il controllo di attuatori meccanici o elettrici (elettrovalvole, relè, allarmi, ..) basandosi su una propria programmazione interna personalizzabile, potendo contare sui dati di numerosi sensori interfacciabili (temperatura, umidità, pressione..), bus dati e pianificazioni orarie. La destinazione per cui è stato concepito è la gestione di edifici industriali e commerciali, ma può venir utilizzato anche per utenze domestiche particolarmente esigenti.

Un sistema di questo tipo è solitamente rivolto ad utenti competenti, sufficientemente esperti nel funzionamento degli impianti, mentre uno degli obiettivi del progetto ISIC è quello di essere utilizzabile anche da utenti con scarsa esperienza.

Per raggiungere tale obiettivo il controllo dei parametri di funzionamento deve essere il più semplice possibile per i meno esperti, ma permettere al contempo agli utenti avanzati di gestire e sfruttare tutte le potenzialità di cui la scheda dispone. In poche parole il sistema deve essere *semplice* ed *adattabile* alle esigenze di ogni cliente. La richiesta di semplicità sarà affidata al software per PC che lo affiancherà, la cui descrizione esula però da questa relazione; compito del firmware sarà invece l'adattabilità.

Come per quasi tutti i prodotti della stessa categoria, prima dell'utilizzo di ISIC sarà necessario elaborare il progetto di utilizzo che sarà caricato nella memoria del dispositivo. Tale progetto sarà definito in seguito come “*strategia*” e rappresenterà tutto ciò che ISIC andrà a fare in quel particolare impianto.

La strategia viene realizzata a PC tramite un software proprietario, attualmente in sviluppo. Data la complessità della stesura, usualmente questa non verrà svolta dal cliente ma dall’azienda produttrice o da un *system integrator*.

Nella strategia vengono definiti anche tutti i parametri di controllo che il cliente può variare in modo molto semplice tramite interruttori, trimmer, LCD touchscreen o altro.

Per gli utenti più evoluti sarà possibile modificare queste impostazioni tramite un PC con un software cosiddetto di “supervisore” (anch’esso in sviluppo) o con un normale browser accedendo direttamente al webserver presente nella scheda.

1.2 – Stato della concorrenza

Numerose aziende sono presenti sul mercato con prodotti destinati al controllo nella automation building. La quasi totalità di questi però si colloca solo su due fasce: quella *high-end*, con grande ingegnerizzazione ma prezzi elevati, e quella *low-end*, relativamente semplice e con sistemi molto limitati.

Nella prima fascia troviamo prodotti molto complessi, dotati di numerose interfacce di comunicazione e ampio grado di personalizzazione. Questi prodotti usualmente dispongono di evoluti sistemi di gestione come la presenza di un webserver per permettere la loro amministrazione remota via internet. Molto spesso i prodotti di questo tipo sono la punta della piramide di tutta una serie di altri dispositivi e periferiche meno complesse commercializzate dalla medesima azienda. Per tale motivo alcune marche creano sistemi chiusi impedendo l’espansione con periferiche concorrenti, obbligando cioè l’acquisto di prodotti della loro stessa azienda. Tale problema a volte è un grosso limite perché spesso ci si scontra con la necessità di far coesistere periferiche preesistenti o con prodotti diversi da quelli in catalogo.

I prodotti low-end hanno principalmente come punto a loro favore il prezzo. Normalmente questi non permettono una programmazione flessibile e si limitano a fornire un’unica funzione personalizzabile solo sotto qualche aspetto.

Il posizionamento strategico di ISIC è quello di coprire una fascia intermedia, che necessita di buone potenzialità ma ad un prezzo contenuto. Si è scelto di non creare un sistema chiuso, permettendo agevolmente l’espansione anche con altri prodotti di altre marche.

ISIC, anche se concepito per una fascia intermedia, presenta al proprio interno notevoli innovazioni, alcune tuttora non presenti nel mercato, che lo rendono sotto molti aspetti vicino allo *stato dell’arte*.

1.3 – Il firmware e una panoramica sulle caratteristiche di ISIC

Il firmware all’interno di ISIC si presenta come un unico blocco sviluppato in linguaggio C. Compito fondamentale di ISIC è l’esecuzione della *strategia* caricata. Questa è presente all’interno del processore non sotto forma di codice eseguibile, ma di strutture dati che vengono interpretate. In tal modo si crea un layer intermedio tra strategia e hardware sottostante, svincolando il progettista dai dettagli tecnici propri dell’ISIC. L’interpretazione della strategia permetterà in futuro di poter

facilmente evolvere il firmware o l'hardware senza la necessità di riscrivere nuovamente tutte le strategie, o ricompilarle per un nuovo processore.
Numerosi sono i modi di interagire e scambiare dati tra il processore di ISIC, vero e proprio cuore del sistema, e le altre zone interne o esterne alla stessa scheda.
La seguente lista mostra le connessioni e le periferiche che ISIC disporrà e che il firmware deve poter gestire:

- Ingressi digitali e analogici (3 modi: 0-10V, 4-20mA, termistore)
- Uscite digitali, relè ed analogiche
- Real-Time Clock
- Porte seriali (RS232 e RS485, protocollo ModBus e protocollo proprietario)
- Bus I^2C
- Bus SPI
- Bus CAN (con protocollo CANopen)
- Porte USB (slave)
- Ethernet (con server HTTP integrato)
- Lettura/Scrittura di su Flash Card SD/MMC
- Possibilità di espansione con moduli ausiliari: ZigBee, LCD, GSM/GPRS, ecc..

Oltre alle precedenti periferiche esterne, il firmware implementerà internamente altre feature come:

- Gestione temporale (settimana standard, giorno o settimana particolare, calendario eventi futuri)
- Backup di strategia su Eeprom
- Bootloader per l'aggiornamento del firwmare
- Ripresa "intelligente" dei processi da un'eventuale mancanza di alimentazione

Lo sviluppo dei moduli di espansione è previsto solo in futuro. Data quindi l'impossibilità di conoscere già adesso i loro dettagli tecnici, ed essendo questi indispensabili per lo sviluppo della loro parte di gestione nel firmware, si è scelto di non considerarli attualmente nella scrittura del firmware. La loro trattazione, pertanto, non sarà presente in questa relazione.

Capitolo

2 Il progetto

2.1 – Processore e ambiente di sviluppo

Come già detto, tutto il progetto verte sul microprocessore LPC2388 della NXP, architettura ARM7, dotato di ben 144 pin. L'integrato è capace di funzionare fino a 72Mhz, avvicinandosi a una media di quasi 70 MIPS, sfruttando una veloce flash interna. Il processore lavora a 32bit, dispone di 512kb di flash, 64kb di SRAM, numerose periferiche integrate di cui alcune addirittura con ulteriore RAM dedicata.

L'architettura ARM è un'architettura molto diffusa e presente in moltissimi prodotti elettronici, dai cellulari fino addirittura ai computer.

Date queste premesse si può capire come la potenza disponibile nel progetto sia notevole se si considera l'ambito embedded in cui stiamo lavorando. Questa considerazione ci ha portato a scartare la programmazione in *assembly* privilegiando il *C* per facilità e velocità di scrittura. Questa decisione è stata avvalorata anche dal non avere tempi di esecuzione molto stretti: la precisione temporale che le specifiche impongono non scende sotto la decina di millisecondi e quindi rende utilizzabile tranquillamente un linguaggio di più alto livello come il C al posto dell'assembly, che ha ragione di essere utilizzato quando i tempi devono essere molto brevi e precisi.

L'ambiente di sviluppo scelto è stato il "Keil uVision 4", basato sul compilatore della stessa ARM, preferito agli altri ambienti disponibili sul mercato in virtù delle sue corpose librerie in dotazione.

La programmazione del processore avviene tramite un'interfaccia JTAG fornita dalla stessa Keil e perfettamente integrata con l'ambiente di sviluppo. Il sistema consiste in un hardware esterno che funge da programmatore e da debugger on-circuit, connesso al PC via USB e alla scheda mediante un flatcable. E' possibile in questo modo programmare e debuggare il processore quante volte si voglia direttamente sulla scheda.

2.2 – La scheda e la demoboard.

Il prototipo hardware della scheda non è stato disponibile per tutta la durata del tirocinio. Per ovviare a questo problema si è lavorato su una demoboard della Keil montante lo stesso tipo di processore. La scheda in oggetto è la MCB2300. Questa possiede fortunatamente una struttura molto simile a quella finale dell'ISIC: ciò rende facilmente portabile il codice già scritto, rendendo necessario adattare soltanto piccole cose come, ad esempio, la variazione di alcuni pin di uscita.

2.3 – RTOS e librerie di terze parti

Nell'analizzare tutte le problematiche del firmware ci si è scontrati con la difficoltà di gestione di alcune periferiche molto sofisticate come USB, Ethernet e CanBus. Il codice necessario all'utilizzo di queste periferiche è molto complesso, richiede che sia robusto e il più possibile esente da bug. Discutendo della questione con l'ingegnere responsabile del progetto si è giunti alla conclusione che è preferibile utilizzare librerie di terze parti così da avere subito codice completo e sicuramente funzionante. In questa discussione si è trattato anche dell'eventualità di installare un *RTOS*, ovvero un *Sistema Operativo Real-Time*.

Un sistema operativo per un microprocessore di queste piccole dimensioni è radicalmente diverso da un sistema operativo più complesso come quelli da PC. Innanzitutto dev'essere *Real-Time*, ovvero capace di poter reagire in brevissimo tempo agli eventi che si generano, deve occupare meno risorse possibili (poche decine di KB), non è necessario che si occupi della gestione delle pagine di memoria. In sostanza lo si può vedere come una libreria da aggiungere al progetto, contenente il kernel ed eventuali altre sottolibrerie per la gestione delle periferiche.

Il mercato mette a disposizione numerose librerie e RTOS, sia commerciali che *open source*. La soluzione open source, sebbene sia inizialmente vantaggiosa dal punto di vista economico, ha alcuni punti deboli come la mancanza di assistenza, la non certificazione del software, le problematiche di licenza nel farlo coesistere con software proprietario.

Per il mio lavoro durante il tirocinio si è deciso di programmare solo le parti base, tralasciando le periferiche più complesse dove in futuro si integreranno le librerie. Anche l'uso di un RTOS è stato temporaneamente accantonato, pur ritenendo molto valida l'idea. Per non precludere in futuro questa strada si è scelto ugualmente di strutturare il codice in modo da facilitare, quando necessario, una migrazione verso un sistema operativo senza dover stravolgere tutto quanto già scritto.

2.4 – Prototipo iniziale preesistente

Prima del mio arrivo, il progetto ISIC era stato a lungo studiato e si era cercato di creare un primo prototipo. L'architettura su cui questo si basava era però completamente differente da quella definitiva. Esso era basato su *dsPic* con tutti i dati salvati in una RAM battery-backupped esterna, collegata mediante bus I^2C. La gestione della memoria era molto macchinosa in quanto ogni lettura dei dati sulla RAM comportava il calcolo dell'indirizzo di ogni singolo byte, la richiesta tramite il bus e la ricezione. Procedura simile anche per il salvataggio dei dati.

Nel mio lavoro di scrittura del codice per il processore definitivo, proprio in merito alla differente architettura che si è voluta dare alla scheda, non potendo riutilizzare quello già esistente, sono stato costretto a ripartire da zero. Il codice preesistente è stato comunque usato come linea guida.

2.5 – Strutturazione del codice

Il motore principale del firmware è il “main loop”, un ciclo infinito in cui, ad ogni iterazione, viene eseguita tutta la strategia ed effettuato il polling degli stati interni.

La strategia è contenuta come una serie di blocchi di dati, ciascuno contenente il tipo di funzione da eseguire e le informazioni specifiche per questa.

Al ciclo principale vengono affiancate le routine di *interrupt* generate dalle periferiche interne al microprocessore (ad es. l’arrivo di un carattere sulla seriale, l’evento di un timer..). Allo scatenarsi di un interrupt, ciò che il processore stava eseguendo, viene interrotto e si avvia il codice di gestione dell’evento. Una volta terminato questo codice, che deve essere il più veloce possibile in modo da non bloccare il microprocessore per lungo tempo, si riprende l’esecuzione da dove si era interrotta.

Una importante variazione che ho introdotto nel codice, favorita dall’architettura e dalla capiente RAM interna al processore, è stata l’utilizzo ove possibile delle “struct” del linguaggio C. Nel prototipo su dsPic la lettura del singolo dato nel relativo blocco di RAM veniva svolta calcolando a mano l’offset dell’indirizzo. Era un calcolo macchinoso, dal facile errore, che richiedeva durante la programmazione la continua consultazione di una tabella in cui fossero segnati, byte per byte, tutti i dati contenuti nel blocco. L’indirizzo era dato dalla somma dell’indirizzo iniziale del blocco e la somma di tutti i byte dei dati disposti prima di quello da leggere. Nel caso di una modifica alla struttura dati era necessario ricalcolare tutti gli indirizzi in ogni punto del codice dove avveniva una lettura della RAM. La difficoltà di manutenzione sarebbe così esplosa al crescere della complessità del firmware.

Con la sostituzione di queste strutture dati con le “struct” del linguaggio C, si è lasciato al compilatore l’onere di occuparsi dell’indirizzamento della memoria, non rendendo più necessario compiere letture e scritture tramite il bus I^2C. Ciò ha portato a un notevole aumento delle potenzialità e della velocità perché i dati sono subito disponibili al processore senza dover aspettare le latenze del bus, risparmiando inoltre molto codice.

L’aggiunta di un nuovo campo nella struttura ora si può fare facilmente in unico punto del codice, con la certezza di non commettere errori negli indirizzamenti.

Capitolo

3 La strategia

3.1 – Introduzione alla "strategia" di ISIC

La strategia eseguita dal "main-loop" è composta da una successione di funzioni di alto livello indipendenti tra loro, con degli input e output virtuali. Ogni funzione si può paragonare ad un blocco. La realizzazione della strategia è la disposizione e il collegamento dei blocchi tra di loro.
Il firmware contiene al proprio interno tutto il codice per far funzionare ciascun tipo di blocco, ha solo bisogno di dati che lo caratterizzino. Questi dati caratterizzanti sono salvati nella memoria e collegati in modo indissolubile e unico a uno specifico blocco. Se, ad esempio, avessimo tre blocchi ADD nella strategia avremmo anche tre descrizioni diverse tra di loro. Nella descrizione sono contenute le informazioni in merito all'indirizzo della Master Table per i collegamenti (vedi paragrafi successivi), a dati interni, a costanti, ecc...

Il seguente schema illustra meglio una semplice strategia:

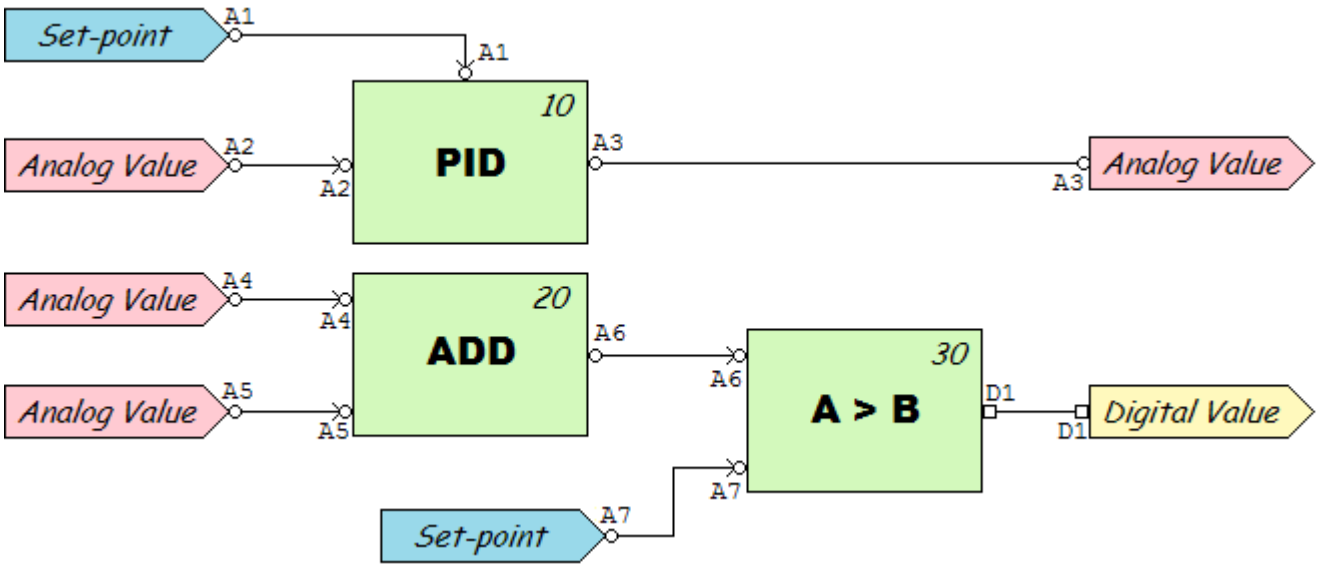

Figura 3.1 – Schema a blocchi di una strategia

Lo schema rappresenta in modo molto semplificato una strategia di tre blocchi funzionali: un controllore PID, un blocco sommatore e uno comparatore. Ai blocchi PID e ADD sono collegati 3 nodi analogici, ad esempio gli ingressi di tre sensori di temperatura. L'output del PID è di tipo analogico mentre quello del comparatore è di tipo digitale: 1 se A>B altrimenti 0.

Le tipologie di blocchi disponibili sono numerose: funzioni matematiche, PID, timer... Con una loro combinazione è possibile ottenere strategie molto complesse, svolgenti numerosi compiti.
Ingressi ed uscite sono tutti nodi analogici, senza distinguo, perché entrambi sono trattati allo stesso modo, ossia come numeri a virgola mobile.

3.2 – Q number e main-loop

Ogni blocco utilizzato nella strategia dispone di un proprio numero, chiamato "Q number" (ad esempio il numero 10 del PID nello schema). Tale valore è unico e non può essere presente in un altro blocco. Il numero Q ha molta importanza perché rappresenta la posizione di esecuzione della funzione nel main-loop.
Il ciclo principale ha come unico scopo il continuo scorrimento di tutti i blocchi in ordine crescente, secondo il loro valore Q. Tale ordine non è sempre necessario, come nel caso dell'esempio, ma in altri momenti può diventare indispensabile.
Una volta eseguito il codice per l'ultimo blocco il ciclo viene azzerato e si ricomincia dal Q più piccolo. La frequenza di esecuzione del main-loop non è critica e diminuisce conseguentemente al complicarsi della strategia. E' stato stimato che anche una strategia pur molto complessa, in ISIC non scenderà mai sotto i 50 Hz, corrispondente a una reattività di 20ms. Tale tempistica è abbondantemente più veloce dei tempi necessari ad applicazioni standard per la building automation, garantendo però in questo modo una migliore fluidità e sicurezza.

3.3 – Set-point

Dallo schema si nota anche la presenza di "set-point". I set-point possono essere fissati al momento della realizzazione della strategia oppure modificabili dall'utente tramite una qualche interfaccia.
Rappresentano un valore che non deriva da rilevazioni esterne ma un numero assegnato esplicitamente, ad esempio la temperatura voluta in una stanza.

3.4 – Strutture dati

Il salvataggio della strategia nella memoria è basato su un grande *array*. Ciascuna locazione contiene un identificatore del tipo di blocco funzionale contenuto e un puntatore all'area di memoria dove sono salvati i dati specifici. L'array, attualmente, è di dimensione fissa, 256 elementi, e il Q corrisponde all'indice della cella. Non è possibile quindi inserire un valore di Q superiore a 256 ed è questo l'attuale limite massimo del progetto.
Riferendoci allo schema della pagina precedente avremo, per esempio, un vettore in cui alla cella 10 l'identificatore specificherà che si tratta di un PID e il puntatore conterrà l'indirizzo della zona di

RAM contenente le informazioni necessarie per caratterizzarlo. Nelle celle del vettore che non contengono dati l'identificatore sarà posto a zero, per indicare così una cella vuota.
La dimensione fissa dell'array crea alcuni svantaggi, ma ha il notevole beneficio di robustezza e semplicità di implementazione. Per ora si è preferito utilizzare questo sistema per velocizzare la scrittura del firmware. In futuro si pensa di sostituire l'array con una LinkedList, ovvero una lista di lunghezza variabile. Oltre a poter essere ridimensionata a run-time, una LinkedList ha il pregio di occupare memoria solo per i dati effettivamente inseriti e quindi non avere più la necessità di limitare il numero Q.

3.5 – Master Table

La comunicazione tra le porte di blocchi diversi avviene tramite la scrittura e lettura in una tabella accessibile a tutte le funzioni. Tale tabella prende il nome di "Master Table". Quando si collegano due blocchi, ovvero si collega l'output del primo all'input del secondo, si indirizzano entrambe le porte alla rispettiva scrittura e lettura della stessa area nella tabella. Sarebbe un errore assegnare a due collegamenti distinti la stessa area poiché essi andrebbero in conflitto nella scrittura; per ogni connessione, pertanto, è necessaria una distinta zona nella tabella. Al momento la Master Table implementata è di dimensione fissa e dispone di 255 float, per lo scambio di dati numerici, e di 288 bit (36 byte) per lo scambio di valori digitali On/Off.
Questa tabella è stata fatta volutamente risiedere in una parte di RAM Non-Volatile dell'LPC2388. Questa memoria, grazie ad una batteria tampone, non perde i dati nel caso di mancanza di alimentazione e permette, per esempio dopo un blackout, di riprendere tutto dalle condizioni dell'istante prima che si verificasse l'interruzione di energia. Come per il vettore della strategia, anche per la Master Table si pensa in futuro di rendere la dimensione espandibile, utilizzando zone di RAM volatile per il salvataggio di valori che non necessitano di essere ripristinati.

3.6 – Allocazione dinamica della memoria

La notevole differenza di dimensioni tra i dati di configurazione di tipi di blocchi diversi ha fatto presto scartare la strada di allocare sempre la stessa quantità di memoria per ogni blocco, partizionando a priori la memoria. Il sistema si è evoluto così verso la strada di una allocazione dinamica della memoria. Tale strada si è rivelata poi la vincente perché permette un'estrema flessibilità, evitando spreco di memoria.
Le librerie del C forniscono tutti gli strumenti necessari per una tale gestione della memoria. Le librerie interne all'ambiente di sviluppo usato, il Keil, inoltre contengono una variante specifica pensata per l'ambito embedded.

Le funzioni principali sono *malloc()*, *realloc()*, *free()*.

malloc() richiede in ingresso la dimensione di RAM da allocare ritornando un puntatore. *realloc()* si occupa di ridimensionare una zona di memoria già allocata. *free()* annulla l'allocazione e rende disponibile la memoria per una futura *malloc()*.

Nel caso finisse la memoria il *malloc()* restituisce *null*, spetterà al firmware gestire questa eccezione.

Prima di utilizzarle nel progetto, queste funzioni sono state testate in condizioni limite per valutarne l'affidabilità e l'utilizzabilità. Uno dei problemi principali rilevati è stato la frammentazione della memoria. Mano a mano che si creano e si cancellano numerose zone di memoria, un po' alla volta, nascono delle piccole parti vuote tra un blocco e l'altro, praticamente inutilizzabili proprio per la loro dimensione. Può capitare così che all'esecuzione del *malloc()*, questo fallisca anche se la quantità di memoria richiesta è inferiore a quella disponibile. La causa è dovuta al fatto che la memoria allocata dev'essere contigua, mentre la memoria disponibile è frammentata e non è quindi disponibile una zona libera abbastanza grande.

Questo problema in ISIC tuttavia può essere considerato marginale in quanto le modifiche avvengono principalmente solo in fase di stesura della strategia. Nel caso si verificasse una frammentazione eccessiva basterebbe salvare tutti i dati, resettare ISIC e ricaricare nuovamente i dati in memoria. Ricaricando, la memoria viene scritta in modo sequenziale e ordinato, senza frammentazione.

Un altro problema che si può verificare con l'uso di *malloc()*, se non si presta molta attenzione, è il verificarsi di "memory-leak". Ogni volta che si alloca una zona di memoria, questa viene riservata fino a quando non si dice esplicitamente di rilasciarla. E' quindi fondamentale, al fine di non saturare la memoria, utilizzare *free()* ogni qualvolta la memoria precedentemente allocata non sia più utilizzata. Durante il debug sono stati trovati un paio di memory-leak, anche se il loro riconoscimento non è stato proprio immediato. Nello specifico ha creato qualche grattacapo un memory-leak di un piccolo buffer, che causava il blocco del firmware dopo un numero casuale di messaggi ricevuti dalla porta seriale.

Capitolo

4 Periferiche, bus e caratteristiche di ISIC

Procediamo ora ad un'analisi più nei dettagli di ciò che il firmware svolge. In questo capitolo si introdurranno le principali periferiche presenti nel microprocessore oltre ad alcune importanti parti di codice sviluppate.

Quando si parla di *periferiche hardware* interne al processore si intendono delle strutture fisiche, integrate nel silicio del core, che assolvono ad alcuni compiti comuni e ripetitivi, sgravando così la CPU principale dalla loro gestione. Questo garantisce migliori prestazioni, minore scrittura di codice e la possibilità di gestire più cose contemporaneamente. Periferiche comuni, presenti ormai in molti microprocessori odierni, sono per la gestione di bus (come I^2C, SPI ed Ethernet) oppure per il PWM e molto altro ancora.

L' LPC2388 è dotato di numerose periferiche anche molto complesse. Per il loro utilizzo è necessario prima inizializzarle. Il processo di inizializzazione avviene tramite la scrittura di opportuni valori di configurazione negli appositi registri. In questo momento si settano velocità, routine di interrupt e quant'altro sia necessario. E' perciò molto importante per il firmwarista conoscere bene il funzionamento di ciò che sta andando a configurare sia leggendo attentamente l'*User Manual* degli LPC della NXP sia informandosi sul funzionamento di ciò che la periferica dovrà svolgere.

Una volta attivate le periferiche, la comunicazione con il programma principale si svolgerà tramite la scrittura di registri.

4.1 – Porta Seriale

La prima periferica inizializzata è stata la porta seriale, per la sua indubbia utilità nel debug.

Il processore utilizzato presenta al proprio interno quattro porte seriali con relativo hardware di gestione. L'utilizzo è molto semplice: una volta impostata la velocità, il numero di bit per parola e la parità, si può inviare o ricevere byte semplicemente scrivendo o leggendo un registro. In ISIC viene sfruttata la possibilità offerta dal processore di attivare un interrupt alla ricezione di un byte. Allo scatenarsi di questo evento, la routine di gestione si occupa di aggiungere ad un buffer il byte ricevuto. Le porte seriali utilizzate nel progetto per ora sono due: una per la programmazione e una per il debug. Quest'ultima resterà abilitata solo finché il programma sarà sotto test. La velocità

raggiunta è stata 57600 baud. Si sarebbe potuto salire ancora di più ma non è stato ritenuto necessario.

Per facilitare il debug è stata utilizzata ampiamente la funzione *printf()* del C, reindirizzando l'output sulla porta seriale. Questa funzione permette di inviare stringhe di caratteri, trasformando in testo anche i valori numerici delle variabili. Collegando la scheda a un PC dotato di porta seriale e di un programma apposito è stato possibile visualizzare tutti i valori voluti. Il programma utilizzato è stato sviluppato per l'occasione in Visual Basic 6 e si limita a mostrare a video i caratteri ricevuti sulla porta seriale.

Per visualizzare sul PC il valore di una ipotetica variabile float *x* basta inserire nel codice la seguente riga:

```
printf("valore variabile = %f\n",x);
```

Tramite l'ausilio di questo sistema, il debug è stato notevolmente semplificato. Per capire e studiare il flusso del codice, quando si sono verificati comportamenti non desiderati, si è attuato il trucco di inserire all'inizio di ogni procedura interessata il comando *printf()* stampando il nome della procedura stessa. In tal modo a PC si è potuto controllare la successione delle chiamate a procedura, riconoscendo facilmente il punto in cui concentrare il debug.

La porta seriale di ISIC non segue necessariamente lo standard rs-232, quello utilizzato per i collegamenti con i personal computer. Potrebbe utilizzare anche gli standard *rs-485* o *r-s422*, anch'essi seriali ma con un trasferimento elettrico differenziale su doppino. I vantaggi stanno nella migliore immunità ai disturbi e quindi nelle maggiori distanze raggiungibili. Per tali motivi questi protocolli sono molto utilizzati nel campo industriale. Nell'hardware di ISIC saranno presenti driver di linea per tutti e tre i tipi.

4.2 – Bus I^2C

Il bus I^2C è un protocollo molto utilizzato in ambito industriale per il colloquio di componenti usualmente montati sulla stessa scheda. Esso è basato su due fili, *data* (SDA) e *clock* (SCL), oltre a un riferimento a massa comune ai dispositivi che vogliono dialogare. La linea SCL fornisce la temporizzazione per i dati presenti sulla linea SDA, la quale è bidirezionale. Il processore utilizzato, prodotto dalla divisione semiconduttori della Philips, sviluppatrice del bus, dispone di un completo ed estremamente potente hardware per la gestione di ben 3 porte I^2C. Il firmware per il controllo del bus è basato su una macchina a stati, con un interrupt ad ogni cambiamento di stato. ISIC è stato pensato per funzionare come *Master*, con la capacità quindi di gestire il bus e le comunicazioni con le periferiche.

Per l'invio e la ricezione di dati ho implementato delle interfacce software facilmente utilizzabili nel firmware.

```
int MasterSendData(unsigned char address, void * data, unsigned int len);
```

Funzione che invia un pacchetto di dati sul bus. Accetta in ingresso l'indirizzo della periferica di destinazione, un buffer contenente i dati che si vogliono inviare e il numero di byte del messaggio.
Ci penserà la funzione a gestire l'hardware sottostante. Il valore di ritorno specifica l'esito dell'operazione: se tutto è andato bene ritornerà I2C_ALL_OK (= 0) altrimenti un numero per indicare il tipo di errore.

```
int MasterReadData(unsigned char address, void * data, unsigned int len);
```

Funzione che legge *len* byte dalla periferica con l'indirizzo specificato e li mette nella zona di memoria puntata da *data*. I valori di ritorno sono gli stessi di *`MasterSendData()`*.

```
int MasterSendAndReceive(unsigned char address, void * senddata,
        unsigned int lensend, void * recbuffer, unsigned int lenbuffer);
```

Funzione che una serie di dati e appena finito l'invio passa in ricezione. Questa funzione è stata utilizzata molto per la gestione dell'eeprom, che per la lettura dei dati richiedeva prima l'invio dell'indirizzo della cella da leggere. I valori di ritorno sono gli stessi di *`MasterSendData()`*.

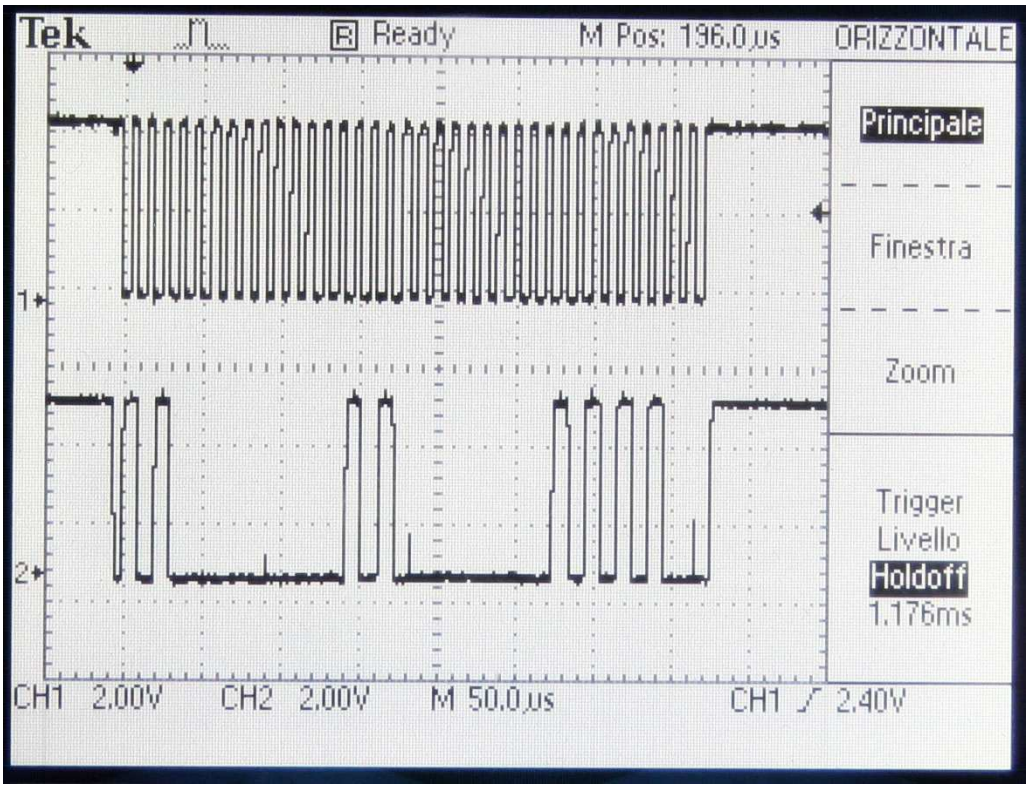

Figura 4.1 – Comunicazione I2C vista con un oscilloscopio

In Fig.4.1 è mostrata la schermata dell'oscilloscopio mentre riprende l'invio di 4 byte sul bus. In alto vediamo la linea di clock, in basso la linea dati. Il primo byte inviato è l'indirizzo del nodo slave, destinatario del messaggio. I restanti byte rappresentano il messaggio.

4.3 – Bus SPI

Il bus SPI (Serial Peripheral Interface) è una *seriale sincrona full-duplex* e può essere considerato un incrocio tra una seriale UART e un bus I^2C. I collegamenti necessari sono tre: un segnale di *clock* (SCLK), una linea di invio dati dal master allo slave (MOSI), una linea di invio dati dallo slave al master (MISO). Oltre a questi è necessario un riferimento a massa, comune a i dispositivi, e un segnale di *enable*, che determina quale sia il nodo con cui si vuole dialogare. La presenza di questo segnale toglie dal protocollo la necessità di prevedere dei comandi di indirizzamento.
Il bus SPI ha il vantaggio di una maggiore velocità e semplicità di gestione. L'invio dei bit è sincronizzato dalla linea di clock, con linee separate per l'invio e la ricezione. La comunicazione per aver luogo necessita di un unico dispositivo master, che gestisce il clock e l'enable, e almeno di un dispositivo slave. Invio e ricezione avvengono contemporaneamente. La linea MOSI viene guidata dal master e contiene i dati da inviare allo slave. Reciprocamente la linea MISO è guidata dallo slave con i dati che questo vuole inviare al master.
Anche lo SPI è presente nel microprocessore come periferica hardware e, come per l'I^2C, è stata sviluppata un'interfaccia software per astrarre nel firmware la gestione sottostante a più basso livello.

```
int SPISendAndReceive(void * data, unsigned int len, void * recbuffer,
                      unsigned int lenbuffer, int offset);
```

Questa funzione riceve in ingresso un buffer di dati da inviare, un buffer per i dati da ricevere con la relativa dimensione in byte, più un offset che specifica dopo quanti byte inviati iniziare la ricezione.
La necessità di un offset è dovuta al fatto che la SPI è full-duplex: invio e ricezione avvengono sempre contemporaneamente. Nel caso si dovesse gestire una comunicazione con un dispositivo slave che, per ottenere una ricezione, richiede prima l'invio di un pacchetto di richiesta, avremmo dovuto creare due buffer ognuno di dimensione *byte_da_inviare* + *byte_da_ricevere* e riempire con '0' le parti non relative. Con l'utilizzo di un offset e del passaggio delle dimensioni di ciascun buffer, possiamo evitare sprechi di memoria. La funzione si occupa di iniziare con l'invio dei byte da spedire; solo dopo la trasmissione del numero di byte specificato dall'offset si inizia anche a ricevere. I byte ricevuti prima dell'offset vengono scartati. Nel caso di superamento dell'offset, se ci fossero ancora byte da inviare, l'invio continua senza interruzioni.

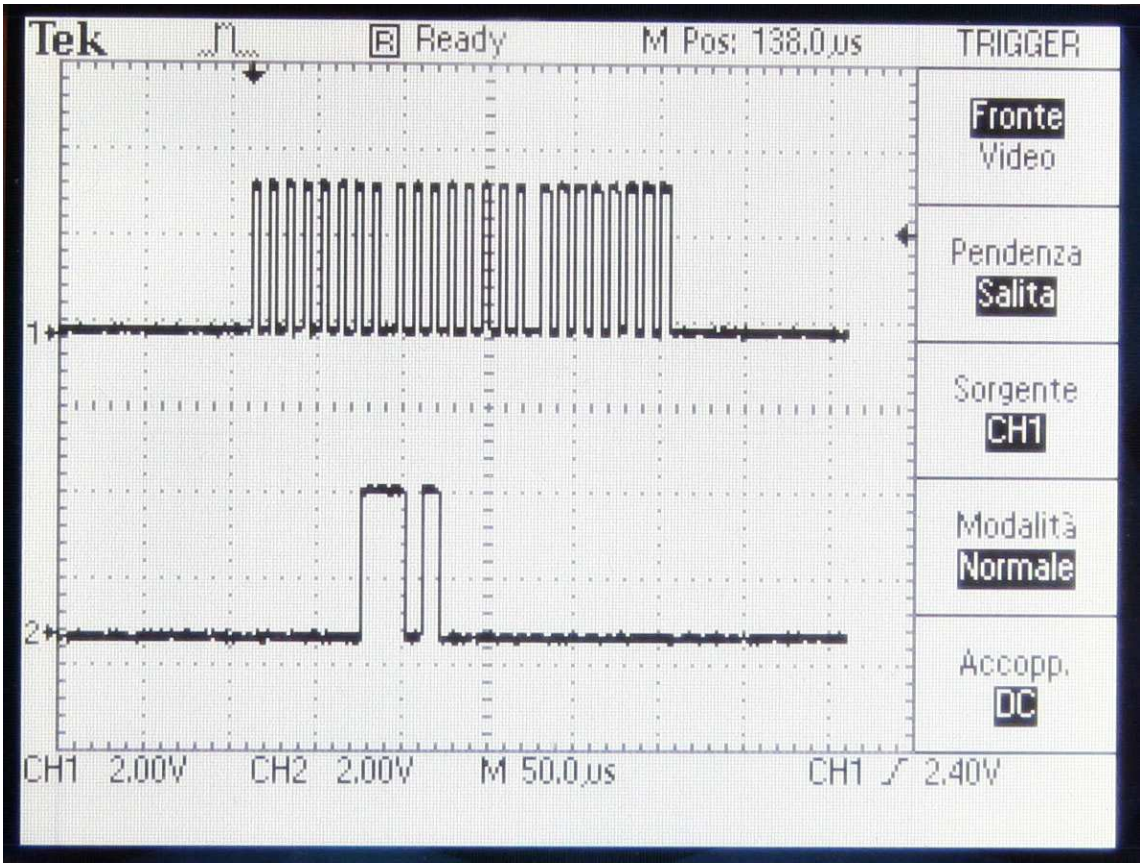

Figura 4.2 – Comunicazione SPI vista con un oscilloscopio

La fig.4.2 mostra l'invio di 3 byte su bus SPI. E' visualizzato in alto il clock e in basso la linea MOSI.

4.4 – ADC e DAC

I convertitori *analogico-digitali* (ADC) e *digitale-analogici* (DAC) sono di estrema importanza in questo progetto dato che la maggior parte dei sensori e degli attuatori nel campo dell'automation building lavora con valori analogici. Spetterà ad ISIC, che al proprio interno lavora esclusivamente in digitale, compiere le trasformazioni necessarie.

Il processore utilizzato, LPC2388, dispone internamente di 8 ADC a 10-bit e di un DAC a 10-bit.

La gestione dell'ADC è relativamente semplice e si può basare su un sistema di lettura sincrono o tramite un interrupt scatenato al completamento della lettura. Il convertitore è di tipo ad approssimazioni successive e impiega un tempo intorno a 2.5µs per la conversione. Nel caso di lettura asincrona durante questo tempo il processore è bloccato in attesa del termine della lettura. In una lettura tramite interrupt invece il programma, dopo aver avviato la conversione, può procedere con altre istruzioni; quando la conversione sarà ultimata verrà scatenato un interrupt. In entrambi i casi alla fine della conversione la lettura del valore sarà la semplice lettura di un registro.

Otto ingressi analogici possono non sembrare pochi, ma le potenzialità del processore di elaborare dati rendono questo numero esiguo. E' stata prevista infatti la possibilità di espandere, tramite moduli ausiliari, il numero di convertitori e quindi di relativi ingressi. In futuro saranno realizzate anche altre differenti tipologie di moduli. Molto probabilmente queste espansioni dialogheranno con la scheda principale tramite i bus CAN o RS485/422, scelti perché rappresentano standard industriali molto conosciuti e godono inoltre di una spiccata robustezza e insensibilità ai disturbi.
Il DAC interno, purtroppo unico, è gestibile molto semplicemente scrivendo in un particolare registro.
I valori che si vanno a leggere o a scrivere nei registri di DAC e ADC sono tutti a 10 bit, compresi quindi tra 0 e 1023. Per ottenere la corrispondenza tra un numero in questo intervallo e valore reale di tensione bisogna considerare che esso corrisponde alla normalizzazione dell'intervallo 0-3 Volt.
Per ricavare la tensione basta usare quindi la seguente formula:

$$V_{pin} = \frac{3}{1023} * x \quad [\mathrm{V}]$$

dove con *x* si intende il numero a 10-bit.

La precisione di conversione è quindi $\frac{3}{1023}\,\mathrm{V} \approx 3\mathrm{mV}$.

4.5 – Salvataggio su eeprom

Tutta i dati contenuti nella memoria di ISIC sono letti e scritti continuamente. Per tale motivo è stato necessario tenere tutto caricato nella RAM, anche se ciò obbliga a doversi limitare alla quantità di spazio presente nel micro (64KB). Qualunque operazione di inserimento o di modifica della strategia avviene direttamente qui: per l'esecuzione non sono stati usati altri tipi di memoria di appoggio, come ram esterne, per motivi di velocità, costo e per il limitato numero di scritture possibili su alcuni supporti (ad esempio memorie flash o eeprom).
La RAM purtroppo è volatile e nel caso di mancanza di alimentazione i dati in essa presenti vengono definitivamente persi. Per ovviare a questo problema si è prevista la possibilità di salvare tutti i dati contenuti su una memoria non volatile. Il salvataggio avviene all'invio di un comando dal PC. Essendo quindi la scrittura eseguita solo raramente si risolvono i problemi di rottura del supporto dovuti ad un numero eccessivo di scritture. Per una normale memoria eeprom vengono usualmente assicurati 100.000 cicli di scrittura per ogni byte. Questi possono sembrare un'infinità, ma se si pensasse di compiere anche una sola scrittura al minuto della stessa cella, si superebbe questo numero in poco più di due mesi.

La lettura avviene ad ogni avvio della scheda, ma questa non comporta nessun rischio di danneggiamento. In questa fase tutti i dati vengono presi dalla memoria non volatile e ricaricati in RAM, ripristinando le strutture dati.

Si è scelto di usare le eeprom della famiglia *24Cxxx*, funzionanti su bus I^2C e di dimensioni fino a 128KB. In attesa della scheda definitiva di ISIC si è collegato, su un supporto esterno di prova, il modello 24C256, contenente 32KB.

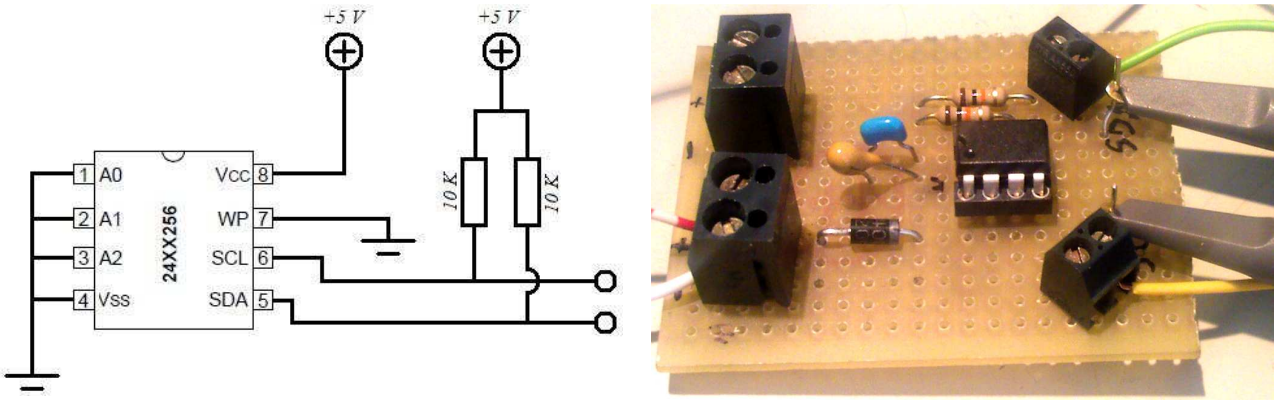

Figura 4.3 – a)Schema di collegamento eeprom. *b)Foto del circuito*

Leggere o scrivere in una eeprom è molto semplice perche, prima dei dati, è sufficiente inviare prima l'indirizzo (2 byte). Per velocizzare le operazioni sono state create le funzioni di seguito riportate, che a loro volta incapsulano le interfacce per il bus I^2C. Le prime due funzioni lavorano su più dati contenuti in un buffer, le seconde invece sul singolo byte.

```
int Eeprom_ReadMem(ADDRESS address, void * dest, unsigned int size);
int Eeprom_WriteMem(ADDRESS address, void * source, unsigned int size);
int Eeprom_ReadByte(ADDRESS address);
int Eeprom_WriteByte(ADDRESS address, char byte);
```

Il salvataggio di dati nella memoria è stata un'operazione complessa. In questo procedimento non è possibile limitarsi alla banale copia grezza di tutti i dati, ma è necessario progettare una struttura di salvataggio che ne consenta poi la corretta lettura e interpretazione.

Nella RAM, come già detto, si è fatto ampio uso di allocazioni dinamiche e sono conseguentemente molte le strutture che contengono puntatori ad altre zone di memoria. Salvare in una memoria non volatile il valore di un puntatore a una memoria volatile non avrebbe senso in quanto alla cancellazione della memoria tale valore perderebbe di significato. Basandomi sulla mia personale conoscenza del formato file *3ds*, ho sviluppato un sistema simile che mi consentisse, durante il salvataggio nell'eeprom, di identificare a cosa si riferisse ogni porzione di dati inserita.

L'elemento fondamentale del formato creato è il "*chunk*", ossia un blocco di dati di lunghezza variabile in cui ogni dato viene inglobato. Ogni chunk contiene all'inizio 2 byte per identificare il tipo di dati contenuto e altri 2 byte per specificarne la lunghezza.
Il formato si presenta come una successione di chunk. La lettura è di tipo sequenziale; per raggiungere un qualsiasi blocco è quindi necessario visitare tutti quelli prima. Per identificare la fine del file viene apposto un chunk di dimensione zero, con l'identificatore di fine file.

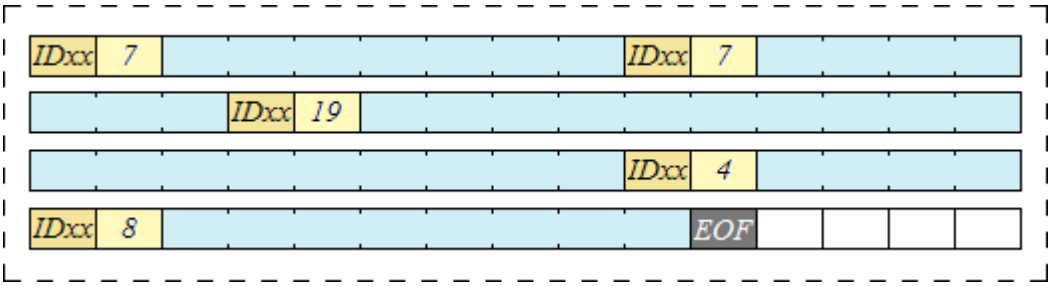

Figura 4.4 – Schema di una successione 5 chunk all'interno di una memoria

Nel caso si dovesse modificare o rimuovere un dato già inserito ci sono tre possibilità. Se i dati da inserire hanno la stessa dimensione di quelli da sostituire si può semplicemente sovrascrivere . Nel caso questo non fosse possibile si può pensare di azzerare quel blocco cambiando l'ID del vecchio chunk e scrivendo un nuovo chunk alla fine. Ultima alternativa è riscrivere tutta la struttura da capo ma è preferibile tenere questa possibilità solo se estremamente necessario in quanto l'elaborazione può impiegare tempi lunghi e stressare le celle della memoria.
Un formato del genere ha sia il vantaggio di essere applicabile facilmente ad una eeprom, sia di poter essere utilizzato anche per strutturare un file, all'interno di un *file system* (come il FAT).
In futuro è prevista la possibilità di caricare o salvare la strategia su *Flash Card*, si potrà così riutilizzare buona parte del codice.

4.6 – Zone orarie

Il firmware di ISIC implementa al proprio interno un evoluto sistema di gestione oraria estremamente flessibile e prevede la possibilità di gestire un numero virtualmente illimitato di temporizzazioni. La presenza di questa funzione è indispensabile nel campo dell'automation building, a cui il prodotto è destinato; si pensi per esempio alla necessità di portare in temperatura un edificio alla mattina, o allo spegnimento automatizzato di alcuni impianti a fine settimana per evitare sprechi di energia.
Per come è stato concepito il sistema, il collegamento tra i blocchi della strategia e la gestione oraria avviene tramite il passaggio di un valore binario on/off, con un bit nella Master Table. Tale bit

assumerà il valore 1 quando l'ora attuale coincide con quella impostata; altrimenti assumerà il valore 0. Il compito di gestire l'evento spetterà alla strategia.
I bit temporizzabili non hanno limite. Ogni uno di questi dispone di una propria zona oraria e di un proprio calendario, indipendente dagli altri.
La zona oraria è personalizzabile per ogni giorno della settimana, inserendo per ciascuno giorno fino a 255 fasce orarie in cui portare a '1' il bit relativo. A una gestione settimanale definita "standard", valida per tutto l'anno, si possono sovrapporre delle maschere giornaliere o addirittura settimanali, chiamate "particolari". Ad esempio, nel caso il martedì della settimana corrente fosse festivo, sarà possibile impostare tale giorno semplicemente come "giorno di ferie". Stessa possibilità è offerta per l'intera settimana, impostando per esempio "settimana bianca". I vantaggi di questa caratteristica sono quelli di poter configurare giorni particolari senza dover cambiare tutti i periodi dell'intera giornata e reimpostarli nuovamente quando il giorno o la settimana particolare sono terminati. Giorni e settimane particolari sono inseribili e personalizzabili a piacimento, senza limiti al loro numero.
Accanto a questo tipo di gestione si affianca il calendario. Questo permette di poter inserire eventi molto lontani nel tempo e scegliere quale maschera applicare in quei giorni. Risulta possibile ora impostare già a inizio anno le festività o gli eventi programmati in anticipo, senza la necessità di dover gestire le tempistiche settimana per settimana.
Per ottenere data e ora il firmware utilizza l'RTC (*Real-Time Clock*) hardware interno al processore. Questo garantisce un preciso orologio digitale autonomo, funzionante anche senza l'alimentazione principale grazie ad una batteria tampone. Tiene il conto, oltre a ore, minuti e secondi, anche di giorno, mese, anno, giorno della settimana, giorni da inizio anno, e gestisce automaticamente gli anni bisestili.

Capitolo

5 La comunicazione con ISIC

Mano a mano che la scrittura del firmware è proceduta, parallelamente si è sviluppato in Visual Basic 6 anche un software al PC per poter testare se il codice scritto si comportava come previsto. Questo rappresenta una versione molto semplificata del programma di creazione della strategia che userà poi l'utente finale o il system integrator.
Il dialogo tra PC e ISIC avviene continuamente: ad ogni modifica nella strategia viene direttamente inviato il cambiamento tramite l'invio di comandi.
Ogni secondo il programma richiede ad ISIC la lettura dei dati di tutti blocchi configurati, con i relativi valori dei nodi all'ingresso e all'uscita. In questo modo è possibile visualizzare chiaramente e in tempo reale il flusso di esecuzione, identificando errori sia nel firmware sia nella strategia.
ISIC e computer si scambiano dati tramite due porte seriali. Una è usata esclusivamente per il debug, l'altra invece è quella principale ed è destinata all'invio della strategia e di ogni sorta di comando. A progetto ultimato sparirà la seriale destinata al debug e la seriale per la strategia sarà rimpiazzata da un più veloce e moderno collegamento USB.

5.1 – Protocollo proprietario

Per l'invio di comandi e strategia è stato progettato un protocollo di comunicazione proprietario.
Tale protocollo è stato concepito per essere utilizzato su un sistema di comunicazione non a pacchetti, come ad esempio in una seriale rs232, rs485 o rs422. Ogni frame, pacchetto di dati, è formato da: *header*, *dati*, *footer*. Il campo dati, di dimensione variabile, virtualmente può essere lungo fino a 16 Kb, ma per scelta progettistica è stato limitato a 1024 byte.
Ogni periferica connessa al bus è identificata da un numero, da 0 a 255, assegnato manualmente. Non è prevista nessuna differenza tra master o slave. Quando una periferica ha bisogno di inviare un dato aspetta che il bus sia libero e invia il proprio pacchetto. La periferica di destinazione resterà sempre in ascolto di ogni pacchetto, se riconosce che il messaggio è destinato a lei allora lo processa, altrimenti lo ignora. Il protocollo non lo prevede obbligatoriamente, ma dopo ogni messaggio è consigliato alla periferica di destinazione l'invio di un messaggio di risposta contenente i dati richiesti o semplicemente un *acknowledge*.
Il protocollo non prevede nessun controllo di collisione, in quanto si lascia al driver hardware sottostante la gestione di questi problemi elettrici.

Queste le strutture in C di header e footer e l'esempio di un frame:

```
__packed typedef struct {
      unsigned char stx;
      unsigned char destination;
      unsigned char source;
      unsigned short length;
} FRAME_HEADER;
```

```
__packed typedef struct {
      unsigned char etx;
      unsigned short CRC16;
} FRAME_FOOTER;
```

Figura 5.1 – Struttura di un frame

STX è un valore costante che identifica l'inizio del frame, vale sempre 0x02.

Destination e source rappresentano l'indirizzo di destinazione e di partenza del messaggio.

Lenght indica il numero di byte della parte dati; è salvato come *Little-Endian*.

ETX indica la fine dell'invio dei dati, è sempre 0x03.

CRC16 serve per controllare la correttezza della trasmissione. E' calcolato su tutto il messaggio da stx a etx compresi. Il CRC (*cyclic redundancy check*) è un numero ottenuto tramite calcoli matematici basandosi su un'intera successione di byte ed è univoco per quella sequenza di dati. Nel caso cambiasse il valore di un byte, cambierebbe di conseguenza anche il valore del CRC. Proprio per questa caratteristica viene utilizzato frequentemente per il controllo di correttezza nelle trasmissioni. Il calcolo viene eseguito sia dal mittente che dal destinatario, se i due valori corrispondono allora, con buona probabilità, i dati sono stati trasferiti correttamente.

La funzione che si occupa di controllare se il messaggio è valido è:

```
int CheckValidFrame(void * buffer, int len)
```

I controlli che questa funzione compie sono quattro e si è cercato di metterli in ordine di complessità di controllo, in modo che la funzione impieghi meno tempo possibile a rilevare un frame non valido.

Innanzitutto si accerta che il frame sia almeno lungo 8 byte, somma dei byte di header e footer. Seconda verifica è se STX ed ETX hanno i valori corretti rispettivamente di 0x02 e 0x03. Dopo questo si passa a controllare se la lunghezza del frame è corretta rispetto alla lunghezza dei dati specificata nell'header. La lunghezza totale dovrebbe essere quella specificata più 8. Alla fine rimane la verifica della correttezza del CRC. Si calcola quindi il CRC e si controlla se corrisponde con quello inviato.

La funzione restituisce la correttezza del messaggio con un'ottima probabilità, anche se non si può dire di averne la certezza.

Nell'implementazione del protocollo ci si è chiesti quando eseguire questo controllo. La domanda può essere posta in modo alternativo: "quando si può dire di aver ricevuto un frame completo?". Se non si tenesse conto del tempo, i dati sarebbero accumulati continuamente in un buffer senza che questo si svuoti mai. Si è giunti così all'introduzione di una ulteriore restrizione al protocollo. Si è scelto di ritenere conclusa la ricezione di un frame se non si ricevono altri dati per più di 3 millisecondi, similmente a quanto prevede il protocollo Modbus. La periferica sorgente deve quindi preoccuparsi di trasmettere un flusso dati continuo, senza lasciar correre tra un byte e il successivo più di 3ms. Nel caso succedesse, la destinazione riterrebbe il messaggio diviso in due o più frame, probabilmente riconoscendoli poi errati.
Se per differenziare un frame dal successivo si ricorresse solo a questo timeout, si introdurrebbe tuttavia un notevole svantaggio per la reattività e la velocità, limitando il numero di frame a non più di 300 al secondo. Per risolvere questo problema si è scelto di controllare la validazione del buffer di ricezione all'arrivo di ogni carattere. Se viene rilevato un frame valido, il firmware può occuparsi di processare subito quel frame e svuotare il buffer per essere pronto alla ricezione un altro frame contiguo. Finché nel buffer non viene rilevato un messaggio valido, ogni byte ricevuto verrà semplicemente posto alla fine.
Allo scatenarsi del timeout, con questo procedimento, il frame contenuto è sicuramente non valido (potrebbe comunque essere un frame Modbus, di cui si parlerà nel prossimo paragrafo). Si può sfruttare però questo evento per svuotare il buffer ed essere pronti a ricevere correttamente il prossimo frame.
Utilizzando queste tecniche è possibile rilevare e processare subito un frame all'istante del suo arrivo. Grazie a ciò è possibile aumentare drasticamente la velocità di risposta.

5.2 – Protocollo Modbus

Il Modbus è un protocollo di comunicazione industriale molto diffuso, basato su porta seriale (rs232 o rs485) o ethernet. Data l'importanza di questo bus si è scelto di implementarlo in ISIC. Si è deciso di utilizzare la stessa porta seriale usata per la configurazione. Tale scelta è stata motivata dal poter ridurre il numero di porte seriali esterne che l'hardware avrebbe dovuto mettere a disposizione, riducendo costi e complessità di gestione.
Un frame Modbus deve obbligatoriamente terminare con una pausa di almeno 3ms senza l'invio di altri byte. Per tale motivo non è possibile l'invio di messaggi in rapidissima successione.
Utilizzando la stessa seriale si è dovuto risolvere il problema dell'identificazione del tipo di messaggio: Modbus o protocollo proprietario. Si è creato così la funzione:

```
int CheckModBusFrame (void * buffer, int len);
```

Essa lavora in modo molto simile alla funzione `CheckValidFrame()`. Viene chiamata solo nel caso di timeout e fa un controllo delle caratteristiche che identificano un messaggio modbus: lunghezza e CRC. Per il calcolo di CRC si è scelto di usare nel protocollo proprietario lo stesso algoritmo usato anche qui per il Modbus, riutilizzando il codice senza doverne scrivere di altro.

5.3 – Istruzioni e comandi

Le modifiche alla strategia e alla configurazione di ISIC avvengono tutte tramite l'invio di comandi contenuti nel campo dati del protocollo proprietario. Per strategia aziendale si è scelto di consentire la possibilità di impartire istruzioni solo tramite questo protocollo.

Ogni comando viene identificato dai primi due byte del messaggio.

Il primo byte, rinominato *Type*, identifica il gruppo a cui appartiene il comando; il secondo byte, a cui è stato dato il nome di *ActionType*, identifica l'istruzione all'interno del gruppo.

I dati successivi sono dipendenti dal tipo di comando inviato.

Alla ricezione di un comando, ISIC controlla che questo sia valido. Se questo viene riconosciuto, alla fine dell'elaborazione viene generata una risposta in cui i primi due byte sono il valore di Type, a cui però è stato portato a 1 il bit più significativo, e l'ActionType. Oltre a questi, nella risposta, ci può essere semplicemente un byte che indica l'esito del comando oppure più di uno, come nel caso di una risposta a una lettura.

La quasi totalità dei comandi che ISIC riconosce sono per la programmazione della strategia. La strategia viene passata dal PC alla memoria di ISIC come successione di istruzioni: ad ogni inserimento di un blocco corrisponde un relativo comando.

Per qualunque tipo di dato che risiede nella memoria esiste anche una funzione per leggerlo. Le funzioni di lettura, in ISIC, assumono una importante rilevanza. Esse, oltre ai dati della struttura, riportano anche tutti i valori in tempo reale che il programma sta elaborando.

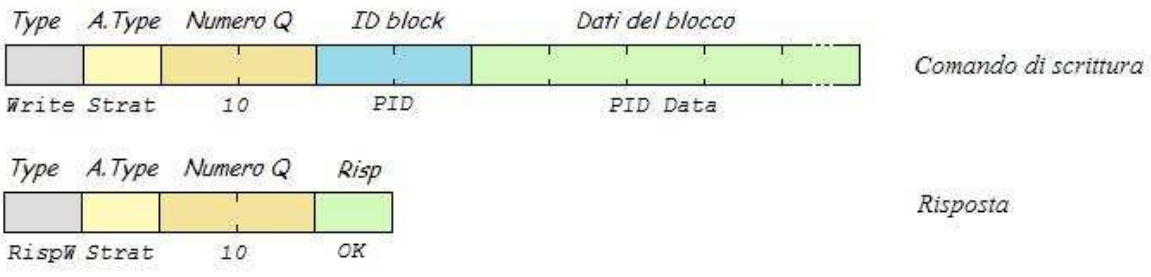

Figura 5.2 – Schema di una comunicazione PC ↔ ISIC

La figura 5.2 illustra come avviene una comunicazione tra PC e ISIC. E' rappresentata solo la parte dati del protocollo proprietario, tralasciando header e footer.

La prima riga è il messaggio che il PC invia per configurare un PID.
Type ed ActionType dicono al controllore che stiamo per scrivere un blocco di strategia. Successivamente viene passato il numero Q, in 2 byte, che identifica unicamente quel blocco nel main-loop. ID_block identifica quale tipo di blocco si vuole inserire, nell'esempio un PID. Per ultimi vengono passati tutti i dati che caratterizzano quel blocco.
ISIC risponde ponendo a 1 il MSB di Type, con lo stesso ActionType e numero Q. Alla fine del messaggio invia un byte che informa se tutto è andato correttamente o ci sono stati degli errori.
I due byte contenenti Q non sono sempre presenti, come lo sono invece Type e ActionType, perché tale numero ha motivo di essere presente solo quando si tratta dell'inserimento o della lettura di un blocco di strategia.

5.4 – Bootloader

La complicazione del firmware statisticamente gioca a sfavore riguardo la presenza di bug nel codice. Più il codice è complesso più i bug sono di difficile identificazione e gli effetti si possono notare anche solo in particolari condizioni o dopo lungo tempo. Per permettere quindi di rilasciare nel futuro nuove versioni del firmware con migliorie e correzioni di errori, senza dover ritirare tutti i dispositivi e riportarli alla casa madre per essere aggiornati, si è previsto di dare la possibilità all'utente di poter aggiornare la scheda facilmente e in modo autonomo.
Questa funzione viene svolta da un bootloader caricato nella flash del processore che, all'avvio di ISIC, controlla se ci sono le condizioni previste per eseguire l'aggiornamento del firmware. Nel caso tale condizione si verifichi, ad esempio la pressione contemporanea di un pulsante al riavvio della macchina, ISIC se viene collegato al PC via USB si presenta come una periferica di archiviazione di massa. Il sistema operativo lo vedrà come un'unità esterna, al pari di una chiavetta di memoria USB, in cui all'interno ci sarà un unico file rinominato "*firmware.bin*".
Questo file rappresenta il firmware. La cancellazione, la copia o la sovrascrittura di questo file si ripercuoterà sul firmware. L'aggiornamento potrà essere semplicemente scaricato da Internet sotto forma di nuovo file *firmware.bin* sovrascrivendo la versione precedente. Questa operazione molto semplice ed immediata rappresenta uno dei punti di forza del progetto ISIC.
Il bootloader risulta invece invisibile se non si verificano le condizioni per l'aggiornamento perché l'esecuzione passa automaticamente alla prima istruzione del firmware.

Capitolo

6 Conclusioni

6.1 – Risultati raggiunti

Nella durata del tirocinio sono riuscito a coprire una buona fetta degli I/O e dei bus: ingressi e uscite digitali e analogiche, RTC, porte seriali (con Modbus e protocollo proprietario), I^2C, SPI, eeprom. Le altre periferiche più complesse sono state accantonate perché la scrittura del codice per il loro utilizzo avrebbe occupato molto tempo e si è preferito privilegiare prima le funzionalità di base necessarie.

Il sistema di interpretazione della strategia, a parte piccoli dettagli e ottimizzazioni, è stato quasi del tutto ultimato.

6.2 – Lavoro da fare e futuri sviluppi

Il lavoro da svolgere su ISIC, prima di portarlo alla sua commercializzazione, è ancora lungo. Il prossimo passo sarà l'implementazione dell'USB che permetterà di poter collegare ISIC anche ai computer portatili, data la quasi impossibilità di trovarne dotati di porta seriale. Questo darà il via al testing di ISIC direttamente nei luoghi per cui è stato progettato, dove non è possibile disporre un PC fisso. Data la criticità del lavoro che ISIC dovrà compiere, il fatto che sarà in funzione 24h 365 giorni all'anno per molti anni, e l'ambiente industriale in cui sarà posizionato è necessario assicurare che il suo funzionamento sia continuo, scongiurando blocchi hardware e firmware. Questa fase sarà molto importante e si prospetta come lunga e complessa.

Si implementerà in futuro la possibilità di leggere e scrivere in una memoria flash esterna, creando anche un sistema di logging.

Verrà inserito nel firmware anche lo stack TCP/IP per il collegamento ad una rete LAN con il conseguente sviluppo di un webserver con cui sarà possibile modificare alcune opzioni. Le pagine presumibilmente saranno salvate all'interno della flash card. Si sfrutterà il TCP/IP anche nel collegamento per lo scambio di dati tra più moduli ISIC uniti insieme in rete.

Si prevede di modificare il codice del bootloader per mappare in un file virtuale anche la memoria eeprom e poter sostituire velocemente l'intera strategia.

L'introduzione di un sistema operativo, già affrontata a inizio relazione, non è stata ufficialmente prevista tra gli sviluppi futuri. La questione si presenterà probabilmente se si verificheranno problematiche che richiederanno il lavoro in multitasking di più parti di codice.

Grande lavoro sarà da dedicare anche per lo sviluppo del software di creazione della strategia e di supervisione. Questo sarà fondamentale per garantire la facilità d’uso di ISIC che dovrebbe preludere al suo successo commerciale.

Bibliografia e riferimenti

1. *"UM10221 – LPC23xx User Manual"* – 25 agosto 2009
   NXP founded by Philips©
2. *"UM10204 – I^2C-bus specification and user manual"* – 19 giugno 2007
   NXP founded by Philips©
3. www.wikipedia.com
   Wikimedia Foundation©
4. *"The Insider's Guide To The NXP LPC2300/2400 Based Microcontrollers"* – febbraio 2007
   Hitex Development Tools. All Rights Reserved
5. *"USB secondary ISP bootloader for LPC23xx"* – 16 ottobre 2008
   NXP founded by Philips©