ESCLUSIVA

L'articolo descrive la strategia seguita nella progettazione e realizzazione della funzionalità di Frequency Hopping per un apparato radio preesistente: la radio tattica CNR2000, di produzione Marconi Selenia Communications. L'attività è stata condotta in collaborazione da Marconi e PXL. Gli aspetti di interesse sono: l'adozione di un metodo di progetto Object Oriented, opportunamente calato nell'implementazione in linguaggio C; la creazione di un livello software di astrazione del dispositivo; la realizzazione di un simulatore su PC che ha permesso il test del codice in un ambiente di più facile utilizzo e controllo

Verso la Software Defined Radio: la realizzazione di un modulo di frequency hopping

Michele Badaracchi,
Umberto Mascia – PXL
Daniele Ferramola,
Fabio Mengoni
Marconi Selenia Communications

Lo Scenario

Software Defined Radio

Negli ultimi tempi l'idea della "Software Defined Radio" (SDR) sta assumendo maggior concretezza, con iniziative congiunte di cui un buon esempio, nonché fonte di informazioni al riguardo, è il Software Defined Radio Forum [1].

I moderni apparati radio realizzano una sempre crescente quota di funzionalità tramite elementi riprogrammabili e dunque in ultima analisi tramite software, i cui compiti spaziano dalle funzioni di controllo a quelle di elaborazione dei dati a quelle di modulazione e demodulazione. Questo contesto favorisce la spinta verso la standardizzazione, con l'obiettivo futuro di avere dispositivi radio le cui funzionalità possano essere facilmente variate al solo prezzo di caricare una diversa configurazione software; e l'obiettivo ancor più ambizioso di interoperabilità fra gli apparati in modo tale che la radio di un produttore possa funzionare correttamente con il software di un altro produttore.

Fig. 1 - L'apparato radio CNR2000

La possibilità per una SDR di operare in un'ampia gamma di situazioni è certamente legata al suo equipaggiamento hardware.

L'SDR Forum definisce [2] come Software Defined Radio propriamente detta, un sistema che possa lavorare sotto il controllo del software e senza variazioni dell'hardware in un range molto esteso (per esempio da 20 a 500 MHz, o da 1 a 2 GHz); in grado di conservare un gran numero di "forme d'onda" e di aggiungerne di nuove tramite supporti di memoria di massa oppure tramite caricamento in linea; con la possibilità di applicare aggiornamenti o correzioni

PXL

É una giovane società romana nata nel 2001, con l'obiettivo di offrire servizi specialistici nella progettazione e nello sviluppo di sistemi e dispositivi embedded.
Nel campo delle telecomunicazioni PXL collabora con Marconi Selenia Communications nella realizzazione di software e firmware per apparati radio. In quest'ambito PXL ha approfondito con dettaglio problematiche e strategie tipiche delle architetture "SDR – Software Defined Radio", nel contesto delle quali è in grado di proporre metodologie di sviluppo innovative ed efficaci, tese a ottenere un elevato grado di manutenibilità, portabilità e robustezza.
Grazie alla particolare esperienza nella fornitura di prodotti e servizi sulle tecnologie Microsoft, PXL ha ottenuto la qualifica di Microsoft Windows Embedded Partner. PXL è pertanto partner ufficiale di Microsoft e interlocutore dei suoi clienti nell'ambito delle tecnologie embedded. In virtù di tali competenze, PXL è partner privilegiato di RA.Fi. Elettronica e Kontron Italia per il supporto del loro hardware e l'assistenza ai loro clienti.

di errori di singoli moduli senza ricaricare l'intera dotazione del software.

La radio tattica CNR2000
Marconi Selenia Communications produce la radio tattica CNR2000 (Fig. 1), destinata all'uso da parte di forze militari di terra. La radio è in grado di funzionare in banda HF e VHF (da 1.6 a 60 MHz). Essa ha un sottosistema di controllo basato su processori Motorola 68030 68360 QUICC e un sottosistema di mo-demodulazione e di elaborazione digitale del segnale composto da tre DSP Motorola 56309.
Il CNR2000 è un dispositivo orientato alla prospettiva delle SDR: la sua architettura consente l'aggiunta di funzionalità caricando i moduli software che le realizzano. Nel corso della pur breve vita dell'apparato (è stato progettato e realizzato nel corso del 2001), questa scelta si è già rivelata vincente in più di un'occasione. Il presente articolo descrive una di queste esperienze.

Il Frequency Hopping
Il principio alla base del "Frequency Hopping" (FH) è di trasmettere (e ricevere) intorno a una frequenza portante variabile nel tempo. La totalità della banda disponibile è suddivisa secondo una canalizzazione opportuna, o più spesso secondo le canalizzazioni usualmente definite per detta banda. La radio trasmittente che lavora in FH, cambia con una certa periodicità la frequenza portante, scegliendola secondo un algoritmo di generazione pseudocasuale, comune a tutte le radio della rete. Le altre radio potranno ricevere il segnale solo se anch'esse generano in sincronismo la stessa sequenza di frequenze portanti.

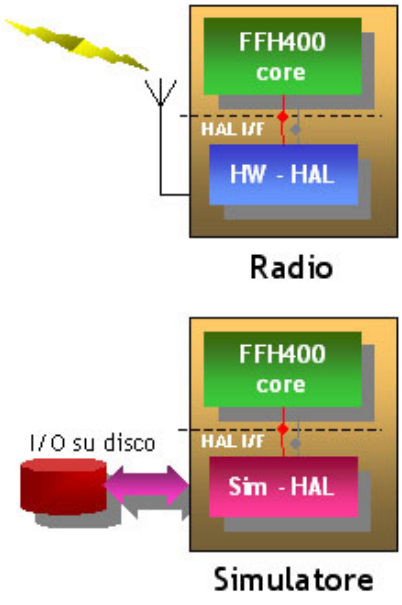

Fig. 2 - Componenti della radio e del simulatore

Nel contesto di radio tattiche militari, l'adozione del FH avviene soprattutto a fini di protezione (ECM, Electronic Counter Measure) da disturbi intenzionali e non, noti in letteratura come:

- Partial Band Noise
- Pulsed Noise
- Partial/Full Band Jammers
- Swept Tone Jammers
- Follower Jammers

Le prime due classi rappresentano i disturbi non intenzionali, che possono essere ricondotti a fenomeni fisici dovuti alla natura del canale Hertziano, e che usualmente si manifestano in uno scenario tattico, come il fading o il multipath.
Le restanti classi costituiscono i cosiddetti disturbi intenzionali (jamming) aventi lo scopo di impedire la comunicazione tra due o più radio.
Diverse sono le modalità di jamming, dalla semplice emissione di rumore a banda larga (Partial/Full Band Jammers) alle più sofisticate tecniche ad inseguimento (Follower Jammers).
Se l'algoritmo di generazione della sequenza di salto è ben ideato, ad un osservatore non sarà possibile ricostruire la sequenza stessa e dunque intercettare o effettuare jamming sulle comunicazioni.
Poiché la trasmissione percorre nel tempo tutti i canali disponibili, si troverà ad essere degradata solamente per brevi periodi. Gli effetti del degrado subito potranno essere ulteriormente attenuati mediante politiche di recupero d'errore, a carico dei livelli superiori del protocollo di trasmissione (Scrambling, Interleaving, Codifica per rivelazione e correzione di errore).
Una rete di radio operanti in FH permette le comunicazioni in simultanea tra differenti gruppi di radio, nonostante queste occupino nel tempo tutta la banda disponibile: è sufficiente che la sequenza di frequenze di ciascun gruppo non si trovi mai nello stesso momento intorno alla stessa portante di un altro gruppo. Ciò si ottiene solitamente parametrizzando l'algoritmo di generazione della sequenza.

L'effetto finale per l'utilizzatore è analogo al sintonizzarsi su un canale concordato nelle ordinarie trasmissioni a frequenza fissa.

Il Frequency Hopping per la radio CNR2000

Nel caso in esame, dato il contesto tipico di utilizzo del CNR2000, si è fissato il requisito fondamentale di 400 salti al secondo (hops per second, h/s).

La banda complessiva utilizzata è da 20 MHz a 60 MHz (banda VHF). Poiché si adotta una modulazione numerica CPFSK (Continuous-Phase-Frequency-Shift-Keyed) a 25 kbps, la banda è canalizzata a passo di 25 kHz, per un totale di 1600 canali disponibili.

Si può dimostrare come tale scelta sia efficace tanto contro i disturbi a larga banda quanto contro i disturbi ad inseguimento.

Per i primi offre un margine di jamming pari a:

$$10\log_{10}(40MHz) - 10\log_{10}(25kHz) - \frac{E_b}{N_0}(10^{-2}\,BER@25\,kbps) = 30dB$$

espressione il cui significato è che un trasmettitore disturbante su tutta la banda deve avere una potenza mille volte superiore rispetto a quella del trasmettitore che si vuole ricevere, per causare il raggiungimento del minimo BER accettabile.

Per il secondo tipo di disturbi, si può dimostrare che, per poter disturbare più del 20% del segnale utile, un jammer di tipo a inseguimento dovrebbe ispezionare l'intera banda in un tempo pari a 1.6 ms; mentre un solo ricevitore riesce a ispezionarla in non meno di 160 ms.

In modalità operativa FH, i servizi disponibili sono:

a) DATI
- 16000 bps
- 2400 bps
- 1200 bps
- 600 bps
- 300 bps

b) VOCE
- CVSDM (Continuously Variable Slope Delta Modulation)

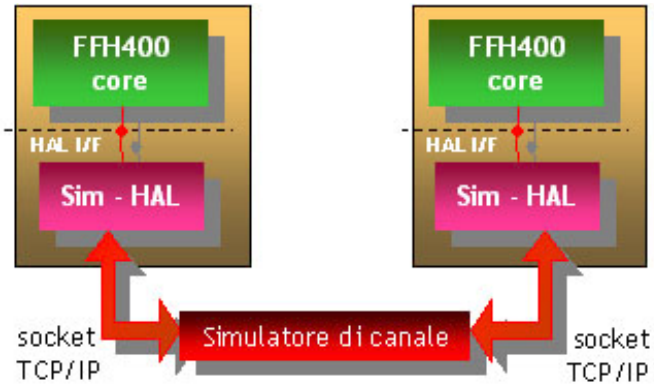

Fig. 3 - Struttura del simulatore completo

A causa delle restrizioni di sicurezza non è possibile presentare in maggiore dettaglio l'architettura Hardware – Software del sistema ECM in oggetto.

L'ambiente di sviluppo

Il compilatore Tasking

La funzionalità di FH rientra nella gestione del flusso digitale dei dati, la cui responsabilità spetta alla catena dei tre DSP di cui il CNR2000 è dotato.

In occasione di questa estensione, si è scelto di adottare come linguaggio di programmazione il C in luogo dell'assembler.

Il compilatore adottato è il Tasking di tipo commerciale: esso offre una versione sostanzialmente aderente allo standard del linguaggio C, con l'aggiunta di alcune parole chiave ed alcune funzioni di libreria che permettono la generazione di un codice più aderente alle caratteristiche dei DSP a disposizione. Citiamo solo a titolo di esempio:

- il tipo _fract che definisce una variabile in virgola fissa a 24 bit (tipo nativo del DSP Motorola 56309);
- il modificatore _circ che definisce un puntatore circolare (il DSP offre la gestione automatica della circolarità e della modularità dei puntatori in memoria);
- i modificatori _X, _Y, _P che permettono di impostare l'area di memoria in cui una variabile deve risiedere, rispecchiando le aree X, Y, P in cui un DSP suddivide lo spazio di indirizzamento;
- il modificatore _inline che obbliga l'espansione di una funzione nel punto in cui viene invocata, anziché generare il salto alla procedura.

Le prospettive offerte dal linguaggio C

La transizione dall'assembler al linguaggio C aveva come prima aspettativa quella di rendere più semplice la stesura e la verifica del software.

In realtà la scelta si è rivelata ben più feconda, perché è risultata subito evidente la possibilità di scrivere un codice portabile dalla piattaforma DSP a quella di un ordinario PC, almeno relativamente alla parte "algoritmica" (cioè non legata alle particolarità dell'hardware che compone il target finale). Il compilatore Tasking stesso suggerisce di percorrere questa strada, fornendo a corredo un include-file mediante il quale è possibile compilare il codice, scritto utilizzando le estensioni proprietarie dei DSP, anche con compilatori differenti.

Questa possibilità è stata utilizzata nel presente lavoro con la realizzazione di un simulatore, argomento del successivo paragrafo V.

Il progetto del software

Un approccio Object Oriented

Il progetto del software è stato affrontato secondo un metodo Object Oriented (OO).

Anzitutto è stato modellato il "dominio del problema", cioè i componenti individuati nell'architettura del sistema; essi sono stati descritti in termini di attributi e proprietà costitutive, e di insiemi di operazioni consentite.

Poiché il linguaggio C non offre nativamente i costrutti tipici della filosofia OO, abbiamo bilanciato i nostri sforzi e cercato di ottenere un insieme significativo di caratteristiche mediante la definizione di alcune regole di codifica:

- ogni classe di oggetti individuata corrisponde ad una coppia di file (.H e .C);
- gli attributi di una classe sono raccolti in una struct con il nome della classe, definita nel .H;
- le operazioni ammesse sulle istanze della classe sono solamente quelle dichiarate nel .H e definite nel .C; il nome dell'operazione ha come prefisso il nome della classe;
- il primo argomento di ciascuna operazione deve essere un puntatore ad una struttura rappresentativa della classe, che al momento della chiamata punterà all'istanza su cui si vuole compiere l'operazione (per somiglianza con il C++, questo puntatore è stato convenzionalmente chiamato pThis);
- operazioni non accessibili dall'utilizzatore della classe sono dichiarate e definite nel file .C con lo specificatore static;
- ogni classe espone una operazione denominata Init, che ha il compito di inizializzare l'istanza per portarla in uno stato congruente; se necessario, espone l'operazione complementare Deinit (analoghe al costruttore ed al distruttore del C++).

Abbiamo deciso di non simulare in alcun modo l'ereditarietà, avendo osservato che il riutilizzo del codice di funzionalità ricorrenti poteva essere ottenuto impiegando l'aggregazione.
Tra i componenti individuati e implementati, meritano una breve descrizione due di essi: la macchina a stati e lo scheduler.

La macchina a stati
Un modello ricorrente nella generalità dei sistemi embedded è la "macchina a stati finiti": rappresentare entità che evolvono secondo un protocollo, in termini di stati stabili e di transizioni causate da eventi istantanei, consente una descrizione del comportamento in modo chiaro e facilmente estendibile.
Nel nostro sistema vi sono più componenti descritti come macchine a stati: si è deciso di catturare questo concetto in una "classe" (StateMachine) che offre il motore di esecuzione delle transizioni all'arrivo di un evento; e di aggregare un'istanza di detta classe all'interno di ogni specifica macchina a stati.
Il motore della StateMachine è assolutamente generico; ciascuna componente che lo utilizza deve implementare le funzioni corrispondenti alle transizioni previste, e registrarle presso il motore nella fase di inizializzazione, definendo così la matrice di transizione che associa a ciascuna coppia (stato, evento) le azioni da svolgere e lo stato successivo.

Lo Scheduler
I DSP della catena di elaborazione del segnale mandano in esecuzione un programma stand-alone e non sono equipaggiati con un sistema operativo. Nel nostro progetto, sono stati individuati più moduli con responsabilità di gestione che devono essere contemporaneamente attivi: si pone il problema di alternarne l'esecuzione, in modo che tutti abbiano accesso alle risorse e possibilità di evolvere.
La soluzione è stata catturata nel concetto di Scheduler: una classe che permette agli oggetti attivi (solitamente macchine a stati) di registrare la loro funzione base, e che ha la responsabilità di alternare l'invocazione di tali funzioni. Il nome di Scheduler deriva chiaramente dalla terminologia dei sistemi operativi: si è ricreato un elemento base di essi, un esecutore di task. I task devono essere cooperativi; in altre parole è a loro carico una gestione equa delle risorse cui accedono e una durata finita (possibilmente breve) del tempo di impiego del DSP, in modo da permettere l'alternanza con gli altri task.
La scelta di non implementare uno scheduler per task preemptive è stata dettata dal semplice criterio di non introdurre complessità eccessive rispetto agli obiettivi da raggiungere.

L'Hardware Abstraction Layer

Il DSP Motorola 56309
Il DSP Motorola 56309 è un processore con capacità di elaborazione di 100 Mips, 20 kword di memoria di programma (P) e 14 kword di memoria dati (X e Y di 7 kword ciascuna).
Il DSP è dotato di 2 porte seriali, di una host port e di alcune linee generiche di I/O. Nella configurazione in esame, queste linee si attestano su una PGA che è utilizzata come elemento flessibile per impostare ulteriori canali di comunicazione.

La definizione delle interfacce delle risorse

La programmazione del dispositivo, precedentemente a questo progetto, era svolta in assembler, con ovvi problemi quali la difficile leggibilità, le occasionali replicazioni, la scarsa parametrizzazione. Secondo l'approccio OO che abbiamo adottato, il primo passo è stato di definire un Hardware Abstraction Layer (HAL): un livello di interfaccia software verso le risorse dei DSP e della logica programmabile, che fosse più naturale e più robusto.

Un esempio significativo è l'interfaccia per l'accesso alle porte seriali, offerta dalla classe ESSIManager (ESSI, Enhanced Synchronous Serial Interface, è il nome del componente all'interno del DSP), che espone le classiche quattro funzioni di accesso ad una interfaccia di streaming:

- Open
- Close
- Read
- Write

La classe ESSIManager ha la responsabilità della gestione del DMA (Direct Memory Access): l'utilizzatore consegna all'ESSIManager un'area di memoria riservata a tal fine nella chiamata alla Open; dopo di che utilizza la seriale secondo la logica usuale di lettura e scrittura senza doversi preoccupare dell'avanzamento dei puntatori di memoria che il meccanismo di DMA prevede, perché la logica è implementata all'interno delle funzioni di Read e Write dell'ESSIManager.

Il simulatore

L'utilità di un simulatore

Durante la fase di sviluppo, si è presentato subito all'attenzione il problema di verificare la correttezza del codice, mentre procedeva la stesura. Eseguire i test direttamente sull'apparato finale presentava alcuni ostacoli:

- **disponibilità dell'apparato ridotta**: nel laboratorio le radio CNR2000 sono condivise tra più linee di sviluppo, e sono una "risorsa preziosa", che non può essere detenuta continuamente da un solo gruppo;
- **difficoltà di controllo della catena dei DSP**: il debugger associato al compilatore compilatore Tasking (CrossView) non è ancora in grado di pilotare simultaneamente più DSP; di conseguenza l'esecuzione passo-passo del codice può essere svolta solo mediante altri tool decisamente meno amichevoli;
- **sovrapposizione delle potenziali cause di problemi**: l'apparato finale è ovviamente composto di parti hardware e parti software; quando si manifesta un problema, dover discriminare se la causa sia dovuta all'una o all'altra parte aggiunge un ulteriore grado di difficoltà.

La direzione verso cui ci siamo orientati è stata di realizzare un programma di simulazione su PC. In questo modo abbiamo dato risposta ai problemi sopra elencati:

- il PC, a differenza dell'apparato finale, è uno strumento di lavoro disponibile per tutti gli sviluppatori senza preoccupazioni di condivisione;
- l'ambiente di sviluppo disponibile su PC è fortemente user friendly; oltre alla possibilità di eseguire il codice passo-passo, è semplice anche generare file di log e far emettere ai programmi segnalazioni di interesse;
- la disponibilità di un simulatore col labora a realizzare il cosiddetto principio della "separation of concerns", o separazione dei problemi: il codice verificato al simulatore - che corrisponde alla parte "algoritmica" e prescinde dalle interazioni con l'hardware - non deve più essere testato quando si passa sull'apparato finale; d'altro canto, problemi che si presentino sull'apparato e non sul simulatore devono avere la loro origine solamente nelle parti che differiscono tra i due ambienti, vale a dire l'hardware e l'interfaccia software verso di esso (l'HAL). Da questa osservazione deriva una linea guida assai utile, pur nella sua ovvietà: quando si lavora sull'apparato finale, bisogna concentrarsi sulle differenze dal simulatore.

Il simulatore della radio

Il primo passo è stato la realizzazione del simulatore della radio (Fig. 2). Si è già accennato al paragrafo II.B di come il codice potesse essere scritto in modo portabile. La definizione di un HAL è stato l'altro aspetto decisivo per la realizzazione del simulatore. L'elemento chiave è la definizione di un'interfaccia ben precisa per l'accesso a ciascuna risorsa. Grazie a questa strategia, è stato possibile scrivere un'implementazione simulata dell'HAL, e generare un applicativo per PC senza dover modificare nulla del "cuore" del sistema software.

Il programma "radio simulata" sostituisce ai dati inviati e ricevuti dall'antenna dati scritti su file, quando si simula la trasmissione, o letti da file, quando si simula la ricezione.

Il simulatore del canale

Il passo successivo è stato di rendere possibile una simulazione con più istanze di "radio simulate" attive contemporaneamente (Fig. 3). In questo modo abbiamo potuto verificare la correttezza dei meccanismi di sincronizzazione e lo svolgimento di trasmissioni tra più radio sintonizzate su "canali" diversi (sempre da intendersi nel senso di sequenze differenti di salto in frequenza, come detto al paragrafo I.C).

Per raggiungere l'obiettivo, abbiamo realizzato un simulatore del canale, che esegue le seguenti operazioni:

- registra presso di sé le istanze attive di "radio simulate";
- a ogni ciclo elementare della simulazione interroga le radio registrate per conoscere lo stato in cui si trovano (trasmissione o ricezione);
- se la radio è in trasmissione, il canale le richiede la coppia (bit trasmesso, frequenza portante di trasmissione);
- se la radio è in ricezione, il canale le richiede la frequenza portante di ricezione valida per quell'istante;
- se esiste una radio in trasmissione intorno alla stessa frequenza, il canale invia alla radio in ricezione il bit trasmesso per quell'istante; altrimenti invia un valore binario casuale, che

rappresenta il campionamento di rumore da parte della radio ricevente.

Il colloquio tra canale e radio simulate avviene per mezzo di socket TCP/IP: questo permette anche di mandare in esecuzione le istanze delle radio su più PC differenti. Restano invariate le possibilità di generare file di log, e di far avanzare il codice passo-passo con un debugger.

Come si è esposto, il simulatore di canale rappresenta un punto in cui sono concentrati e poi smistati i dati trasmessi alle varie frequenze. È stato perciò naturale aggiungere la possibilità di simulare un disturbo, di entità configurabile e intorno a bande di frequenza specificate, in modo da verificare la reazione del sistema e la capacità di reiezione degli errori anche in presenza di jamming.

L'esperienza della realizzazione di un modulo software che ha esteso le funzionalità della radio CNR2000 di Marconi Selenia Communications, ha dimostrato che tale apparato è adatto all'evoluzione secondo i paradigmi delle Software Defined Radio.

Il sensibile salto innovativo di uno sviluppo ad alto livello ha permesso di svincolare ciò che sino a oggi veniva considerato firmware, strettamente dipendente dall'architettura del dispositivo finale, dal risiedere nelle memorie dei DSP, e di vincere la scommessa di un software estremamente flessibile tanto da poter essere caricato a tempo di esecuzione.

Per cogliere i massimi benefici da questa strategia, è fondamentale d'altro canto adottare un metodo di progetto adeguato che si basi su:

- progettazione Object Oriented
- adozione di un linguaggio di alto livello (in questo caso il C)
- definizione e realizzazione di un Hardware Abstraction Layer.

La realizzazione del simulatore è stata ampiamente ripagata dalla rapidità con cui è stato possibile far evolvere il software e portarlo in produzione. Inoltre l'ambiente di simulazione è del tutto riutilizzabile per futuri sviluppi.

Ringraziamo per la disponibilità e la collaborazione, il personale del laboratorio HF tattico di Marconi Selenia.
Un particolare ringraziamento va al responsabile del laboratorio HF Tattico, ingegner Mauro Parenti, il cui apporto di conoscenze è stato indispensabile per il buon esito del progetto.

Bibliografia

1)http://www.sdrforum.org
2)Documento "Definitions & Semantics" sul sito di SDR Forum:
http://www.sdrforum.org/tech_comm/definitions.html