160 PAGINE!

Mensile di elettronica applicata, attualità scientifica, novità tecnologiche.

Elettronica In

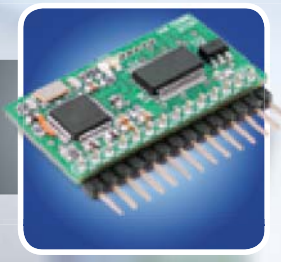
oltre l'elettronica

www.elettronicain.it

€ 5,00

Anno XVI - n. 146
Aprile 2010

Avionica & GPS



- Lettore MP3 indirizzabile

- Comandiamo 8 relé con la porta USB

Telecontrollo GSM modulare

- Data-logger su microSD
- Caricabatterie per localizzatore GSM
- Dimmerare le lampade a LED
- PIC Genius, un software di sviluppo per tutti
- Terzo stop lampeggiante
- Il film sottile: la nuova frontiera del fotovoltaico
- Power Management
- Corso di programmazione Wireless CPU

• novità • Corso LabVIEW

Poste italiane Spa - Spedizione in abbonamento Postale: D.L. 353/2003 (conv. in L. 27/02/2004 n°46) art. 1 - comma 1 - DCB Milano

Corso Wireless CPU

Introduzione alle Wireless CPU Wavecom

Moduli radio dotati di microcontrollori talmente potenti da poter assolvere tutte le funzioni che solitamente vengono affidate a CPU esterne. Iniziamo a conoscerli.

dell'Ing.
NICOLA FRISO

Volete sviluppare un progetto di telecontrollo GSM? Allora sulla vostra scheda avrete previsto almeno un microcontrollore e un modulo GSM industriale M2M "Machine to Machine".
Una UART del microcontrollore sarà occupata a gestire il collegamento con il modulo GSM. Il microcontrollore dovrà avere sufficienti capacità di memoria per contenere quel minimo di libreria che vi serve per la gestione dei comandi AT essenziali per il funzionamento del modulo GSM. Visto che avete a disposizione una connessione remota (dati GSM o GPRS) magari pensate anche di implementare una funzione che permetta l'upgrade da remoto della vostra applicazione. Una volta messa a punto la comunicazione con il GSM, se non lo avete già fatto, dovrete scrivere tutto il codice per gestire a basso livello le periferiche del microcontrollore (I/O, interrupt, timer, SPI, I²C ecc…) e poi finalmente dedicarvi allo

Fig. 1 *Augusto Wireless CPU*

sviluppo delle funzionalità specifiche della vostra applicazione. Potreste anche avere la necessità di disporre di un sistema operativo real-time. Questa scelta però comporta un ulteriore aumento delle risorse richieste al vostro microcontrollore oltre al doversi sorbire uno studio approfondito del funzionamento del sistema operativo scelto.
Lo sviluppo di un progetto di questo tipo in termini di tempo è molto oneroso. Se volete ridurre drasticamente il tempo di sviluppo, risparmiare sui costi dell'hardware, disporre di un parser di comandi AT completo, contare già su di una funzione per l'upgrade remoto della vostra applicazione, avere un microcontrollore potente con già incorporato un sistema operativo (S.O.) real-time, disporre di tutte le librerie necessarie per gestire la connessione GSM o GPRS e tutte le periferiche classiche normalmente disponibili in un microcontrollore, allora una Wireless CPU è quello che fa per voi.
Una Wireless CPU è un chip che integra un microcontrollore programmabile e tutto l'hardware (in genere un microcontrollore ed un DSP) e il firmware necessario per gestire i protocolli di rete e la parte in radiofrequenza che si occupa della connessione GSM/GPRS.

LA FAMIGLIA DI WIRELESS CPU DI WAVECOM

Da alcuni anni il produttore francese di moduli GSM "Wavecom", ora fusosi con l'azienda americana "Sierra Wireless", si è specializzato nel mercato M2M ed è all'avanguardia nello sviluppo di moduli GSM programmabili. Questi moduli consentono, tramite un sistema operativo e un insieme di librerie (API) scritte in C denominate "Open AT", di implementare le proprie applicazioni all'interno del modulo GSM stesso, sfruttando appieno la memoria e tutte le periferiche di cui dispone il microcontrollore presente all'interno del modulo GSM.
Le famiglie di Wireless CPU di Wavecom sono riportate nella **Tabella 1**: si va dai moduli della serie Q24xx, che montano un processore ARM7, passando per la serie Q26xx (che monta processori ARM9) fino ad arrivare alle più recenti Wireless CPU serie WMPxx (ARM9) le quali, anche nell'aspetto, ricordano quello di un microprocessore.
I moduli appartenenti alle famiglie "Quick" hanno tutti l'aspetto del classico modulo GSM standard con connettore SMD passo 0,5 mm da 40 poli in su, e sono tutti programmabili con l'ambiente di sviluppo Open AT, per cui non necessitano di microcontrollore esterno. Si differenziano per il tipo di CPU che montano, per le dimensioni della memoria Flash e RAM e per il numero di periferiche e GPIO che implementano. Alcuni di essi sono disponibili anche con porta-SIM integrato.
Esistono anche due moduli con caratteristiche speciali: il Q26 Extreme, che dispone dell'HSDPA e il Q52, che integra anche un modulo GPS.
Se ci seguite da molto, saprete che in passato abbiamo pubblicato progetti impieganti i moduli Q24 e Q25, anche se sfruttando questi ultimi essenzialmente come modem GSM/GPRS. Adesso riprendiamo il discorso, vedendo però il pieno utilizzo delle funzioni delle Wireless CPU.

LE WIRELESS CPU DELLA SERIE WMP

In questa serie di articoli, prenderemo in considerazione solo la nuova famiglia di Wireless CPU serie WMP, fermo restando il fatto che tutto quello che tratteremo, compreso il codice, è facilmente portabile anche sulla famiglia Quick, avendo Wavecom unificato il più possibile il set di API del sistema operativo.
La serie di Wireless CPU WMP propone diversi modelli (vedi Tabella 1) che si differenziano principalmente per la frequenza di clock del processore ARM9 e dal numero di GPIO che mettono a disposizione. Sono poi disponibili alcune Wireless CPU speciali con certificazioni automotive (WMP150) e certificazioni ATEX per ambienti potenzialmente

esplosivi (WMP120).
Le Wireless CPU serie WMP contengono al loro interno il microcontrollore ARM9 e il DSP per la gestione della radiofrequenza e dei protocolli GSM. La memoria Flash e RAM deve essere invece montata esternamente, connessa alle linee dedicate. Normalmente si utilizzano della memorie Combo, che nello stesso chip integrano la Flash e la RAM e che sono largamente utilizzate nell'industria del mercato consumer dei telefoni cellulari. I due modelli consigliati da Wavecom hanno 2 MB di capacità per quanto riguarda la RAM e 8 MB oppure 16 MB di Flash. Si possono comunque montare memorie Flash e RAM separate.
Il package della Wireless CPU è identico per tutti i modelli ed è un BGA (Ball Grid Array) con contatti a passo 1mm BGA576, 25x25x3 mm. Anche il package della memoria Combo è un BGA passo 0.8mm 8x10mm. Questo tipo di package è notoriamente impossibile da saldare a mano e anche per la produzione industriale, si deve far ricorso ad aziende specializzate per il montaggio di tali componenti. Proprio il particolare package, consente la riduzione dei costi delle Wireless CPU e dello spazio occupato sul circuito stampato dell'apparecchiatura dove le stesse CPU vengono utilizzate.
Le Wireless CPU di Wavecom nascono con il preciso scopo di essere utilizzate in progetti che prevedono produzioni in numerosi esemplari (da 3.000 pezzi/anno in su). In alternativa si possono montare i moduli della serie Q26xx, che si basano sullo stesso hardware ma hanno la classica pin-out di un connettore strip a 100 pin (passo 0,5mm) comunque difficile da montare a mano, ma sicuramente più facilmente gestibile anche nei processi automatici.

Tabella 1 - *Wireless CPU Wavecom*

Modello	Caratteristiche
WMP50	**Processore**: ARM9, 32 bit, 26 MHz, che supporta il sistema operativo Open AT® **Memoria:** esterna, combo Flash/PSRAM 32/16, 64/16 Mbit) and more **I/O digitali:** 11 GPIO, 2 INT, SPI, tastiera a matrice 5 x 5, memory-bus **Porte di comunicazione:** 2 UART, USB 2.0 **Interfacce analogiche:** 2 ADC **Interfacce audio:** 2 canali analogici **Cellulare:** Global operation (quad-band 800/900/1800/1900 MHz) **Dati Cellulare:** standard GSM, SMS, Fax, CSD (circuit), GPRS cl 10 (packet) **Voce Cellulare:** Quad codec (FR/HR/EFR/AMR) **Package:** BGA576, 25 x 25 x 4 mm, sfere da 0,5 Ø, passo 1 mm
WMP100	**Processore:** ARM9, 32 bit, 104 MHz, compatibile con il sistema operativo Open AT® **Memoria:** esterna, fino a 128 MB Flash, 128 MB PSRAM **I/O digitali:** Up to 44 GPIO, 4 INT, I²C, 2 SPI, tastiera a matrice 5 x 5, memory-bus **Porte di comunicazione:** 2 UART, USB 2.0, parallela **Interfacce analogiche:** 4 ADC, DAC **Interfacce audio:** 2 canali in PCM **Cellulare:** Global operation (quad-band 800/900/1800/1900 MHz) **Dati Cellulare:** GSM standard SMS, Fax, CSD (circuit), GPRS cl 10 (packet) **Voce Cellulare:** Quad codec (FR/HR/EFR/AMR) **Package:** BGA576, 25 x 25 x 4 mm, sfere da 0,5 Ø, passo 1 mm
WMP50	**Processore:** ARM7, 32 bit, 52 MHz, compatibile con il sistema operativo Open AT® **Memoria:** interna, 32 Mb Flash and up to 16 Mb SRAM or PSRAM **I/O digitali:** Internal, 32 Mb Flash and up to 16 Mb SRAM or PSRAM **Porte di comunicazione:** 2 UART **Interfacce analogiche:** 2 ADC **Interfacce audio:** 2 canali differenziali analogici **Cellulare:** Global operation (quad-band 800/900/1800/1900 MHz) **Dati cellulare:** GSM standard SMS, Fax, CSD (circuit), GPRS cl 10 (packet) **Voce cellulare:** codec supportati (FR/EFR/AMR) (HR opzionale), VDA2 C **Package:** 58x32x3,9 mm, 60 pin su connettore a passo 0,5 mm, porta-SIM integrato (opzionale)
Q2686	**Processore:** ARM9, 32 bit, 104 MHz, compatibile con il sistema operativo Open AT® **Memoria:** Internal, supports up to 128 Mb Flash, 128 MB PSRAM **I/O digitali:** Up to 44 GPIO, 2 INT, 2 SPI, 1 I²C, tastiera a matrice 5 x 5 **Porte di comunicazione:** 2 UART, USB 2.0 **Interfacce analogiche:** 2 ADC **Interfacce audio:** 2 analogue channels, 1 PCM **Cellulare:** Global operation (quad band 800/900/1800/1900 MHz) **Dati cellulare:** standard GSM, SMS, Fax, CSD (circuit), GPRS cl 10 (packet) **Voce cellulare:** Quad codec (FR/HR/EFR/AMR) **Package:** connettore 100 pin
Q2687	**Processore:** ARM946, 32 bit, 104 MHz compatibile con il sistema operativo Open AT® **Memoria:** Internal, 32 / 8 Mb Flash **I/O digitali:** fino a 44 I/O, 2 INT, 2 SPI, 1 I²C, tastiera a matrice 5 x 5 **Porte di comunicazione:** 2 UART, USB 2.0 , 1 parallel ports , 1 DAC **Interfacce analogiche:** 2 ADC **Interfacce audio:** 2 analogue differential audio, 1 PCM **Cellulare:** Global operation (quad band 800/900/1800/1900 MHz) **Dati cellulare:** GSM standard SMS, Fax, CSD (circuit), GPRS cl 10 (packet) **Voce cellulare:** codec supportati (FR/EFR/AMR) (HR opzionale) VDA2 C **Package:** 40x32x4 mm, connettore 100 pin, passo 0,5mm

IL MODULO "AUGUSTO" DI SHITEK

Shitek Technology è un'azienda italiana, specializzata nel telecontrollo GSM, che da anni sviluppa prodotti con le Wireless CPU di Wavecom. Recentemente ha realizzato un modulo GSM/GPRS denominato "Augusto", che integra una Wireless CPU WMP50 (o WMP100), 4 MByte di Flash e 2 MB di RAM, un porta-SIM, un connettore U.FL per antenna, un RTC, il tutto integrato in una scheda con tutti i pin (ball) della WCPU riportati sul bordo della scheda stessa e con possibilità di montaggio superficiale laterale ("castellation

... segue Tabella 1 - *Wireless CPU Wavecom*

Modello	Caratteristiche
Q26 Extreme	**Processore:** ARM946/DSP, 32 bit, 26÷104 MHz compatibile con sistema operativo Open AT® **Memoria:** Internal, 64 Mb Flash and 32 Mb PSRAM **I/O digitali:** fino a 45 GPIO (26 GPIO 2,8V, 19 GPIO 1,8 V), 2 SPI, 1 I²C, tastiera a matrice 5x5 **Porte di comunicazione:** 2 UART, USB 2.0, 1 16 bit parallel port **Interfacce analogiche:** 2 ADC, 1 DAC **Interfacce audio:** altoparlante e 2 microfoni, PCM digital interface (768 kHz, 16 bit data MSB first, Master) **Cellulare:** Tri-band UMTS/HSxPA (WCDMA/FDD), Quad-Band GSM 850/1900/2100 MHz (band I, II, V), Quad-band 850/900/1800/1900 **Dati cellulare:** GSM + GPRS multi slot class 12, EDGE (E-GPRS) multi slot Class 12, WCDMA, HSxPA **Voce cellulare:** Quad codec (FR/HR/EFR/AMR) per GSM e AMR per WCDMA; VDA2A **Package:** 40x32x6,55 mm, connettore 100 pin, passo 0,5mm,
Q52 Omni	**Processore:** ARM9, 32 bit, 104 MHz compatibile Open AT® **Memoria:** Internal, 64 Mb Flash and 32 Mb PSRAM **I/O digitali:** Up to 18 GPIO, SPI **Porte di comunicazione:** 2 UART **Interfacce analogiche:** 2 ADC, 1 DAC **Cellulare:** Global operation (quad band 800/900/1800/1900 MHz) **Dati cellulare:** standard GSM, SMS, Fax, CSD (circuit), GPRS cl 10 (packet) **Sezione satellitare:** Trasmissione: 148÷150,05 MHz; Ricezione: 137÷138 MHz; potenza 5 W **Localizzazione:** GPS (opzione) **Package:** 40x32x6,55 mm, connettore 40 Pin, passo 0,5mm, porta-SIM integrato

package") o mediante un comodo connettore pin strip passo 2 mm (**Fig. 1**).
Questa soluzione permette l'utilizzo delle potenti CPU Wireless anche agli hobbisti o a tutte quelle aziende che non hanno le possibilità tecniche o i numeri necessari. Nella **Tabella 1** sono riportate le principali caratteristiche della CPU WMP50 e del modulo Augusto.
Questo corso farà uso del modulo "Augusto" e della relativa demoboard EVKAUGUSTO, che andremo a realizzare nelle prossime puntate.

L'AMBIENTE DI SVILUPPO

Diamo uno sguardo ora al sistema di sviluppo di queste Wireless CPU (che per comodità, da ora in avanti, abbrevieremo in WCPU).
Wavecom Open-AT OS è la piattaforma di sviluppo tramite cui è possibile creare le proprie applicazioni; integra un sistema operativo real-time e un set di API completo per la gestione della WCPU.

Come sistema operativo real-time, Wavecom propone Open-AT e come ambiente di sviluppo (ossia l'IDE) il prodotto "M2M Studio" basato su Eclipse. Integrato nell'IDE, è a disposizione un semplice wizard per creare nuovi progetti, testare gli esempi forniti e creare applicazioni nuove partendo da tali esempi.
Open-AT è diventato negli anni uno standard de-facto per la programmazione di moduli GSM e conta oggi su un nutrito numero di sviluppatori e di un forum (*www.wavecom.com/forum*) vivace ed aggiornato dove potete trovare il supporto degli sviluppatori certificati wavecom e tanti suggerimenti, esempi ed aiuti immediati per qualsiasi problema incontriate nello sviluppo della vostra applicazione Open-AT.
Iniziamo dunque col procurarci una versione aggiornata della suite Open AT, che è gratuitamente scaricabile dal sito di Wavecom, previa registrazione. Per il download dovete seguire questo percorso: *www.wavecom.com* → **Products → Wireless CPU Range → Wireless Microprocessor Family → WMP50** (vedi **Fig. 2**).

Fig. 2 *- Download del software dal sito Wavecom*

Cliccate quindi sul tab "Downloads" quindi aprite la cartella:
Open AT Software Suite → Software Development Kit → Official Release.
Qui trovate il file eseguibile di installazione

dell'Open-AT.
L'ultima versione che è disponibile alla data di pubblicazione di questo articolo è la "Wavecom Open AT Software Suite v2.30".
La versione contiene l'Open-AT OS, l'IDE M2M Studio 1.0 e tutti gli esempi e i plug-in a corredo (vedere la **Tabella 2**).
Da questa posizione scarichiamo anche il firmware del sistema operativo da installare nella nostra WCPU. L'ultima versione ufficiale disponibile nel momento in cui viene pubblicato questo articolo è la **"R74_00-cus-wmp-02.wpk3"**.
A questo punto installiamo Open-AT V2.30 sul nostro PC, seguendo le istruzioni di installazione.
Per poter testare ed eseguire in modalità di debug gli esempi di codice forniti in questo corso, è necessario avere a disposizione l'evaluation kit di Wavecom "Wireless Microprocessor and Open AT Software Suite Development Kit", oppure la più semplice ed economica demoboard "EVKAUGUSTO", costruita da Shitek Technology, che andremo a realizzare nel prossimo articolo. La differenza tra la demoboard originale Wavecom e quella di Shitek è che mentre con la prima si possono utilizzare al 100% tutte le CPU della serie WMP, quella di Shitek è stata creata specificatamente per la WCPU WMP50. Le serie WMP50 hanno capacità di calcolo inferiore rispetto alle WMP100: 26 MHz/10 Mibps, contro i 104 MHz/88 Mibps delle WMP100. La demoboard EVKAUGUSTO consente di montare anche moduli WCPU della serie WMP100 (WMP100,150,120) che sono pin to pin compatibili limitatamente alle risorse hardware che sono condivise con la WMP50. Di fatto nella WCPU WMP50 si hanno a disposizione 11 GPIO,2 UART, 1 USB, 1 SPI/I2C, 2 INTERRUPT, 1 ADC, 1 RTC, risorse più che sufficienti per gli scopi di questo corso. Si ricorda infine che il codice scritto è compatibile con tutta la famiglia WMP.
Prima di lanciare l'IDE M2M Studio 1.0 assicuriamoci che la nostra Wireless CPU sia attiva e funzionante. Alloggiamo il modulo sulla demoboard, inseriamo una SIM nell'apposito alloggiamento, colleghiamo l'antenna e connettiamo la demoboard al PC attraverso la porta seriale; infine diamo l'alimentazione. Lanciamo il programma HyperTerminal

***Tabella 2** - Plug-in del software Wavecom*

Modello	Caratteristiche
C-GPS	Librerie plug-in per la gestione di un modulo GPS connesso alla UART della WCPU
GR	Plug-in per l'utilizzo delle WCPU serie Q64 che rimpiazzano la vecchia serie GR di Sony Ericssonn (acquistata da Wavecom)
LUA	Librerie per la programmazione in linguaggio LUA della WCPU (www.lua.org/pil)
WIP	Librerie per l'utilizzo dello stack TCP-IP

di Windows che troviamo in Programmi → Accessori → Comunicazioni. Se usiamo Windows Vista, occorre scaricarlo a parte dalla pagina web *http://www.hilgraeve.com/htpe/download.html*.
Settiamo la COM corretta a 115200, no parità, 8 bit, controllo di flusso Hardware e digitiamo AT seguito da un invio; a questo comando, la WCPU deve rispondere con OK:

```
>AT

OK
```

In seguito lanciamo i comandi ATI3 per vedere la release del FW:

```
>ATI3

R74_00gg.WMP50 2120060 052109 13:11
```

Digitiamo AT+WIND=255 per attivare i codici "WAVECOM INDICATE" che ci permettono di vedere passo-passo l'evolversi delle attività della WCPU, oltre che di catturare particolari eventi all'interno della nostra applicazione. Una trattazione completa dei comandi AT, compreso l'AT+WIND, sarà l'oggetto di una puntata successiva; per ora limitiamoci ad osservare cosa succede dopo aver dato i seguenti comandi AT (le righe di commento precedute da "//" non vanno digitate). Qui di seguito trovate i comandi impartiti e il loro significato.

```
>AT+IPR=115200      // imposta la velocità della seriale a 115200 (di default è attivo il riconoscimento automatico)

OK
```

```
>AT+WIND=255        // abilita gli eventi WIND

OK

>AT&W              // salva le impostazioni fatte fino
                   a questo punto

OK

>AT+CFUN=1         // resetta la WPCU

WIND:1             // WIND code che la presenza
                   della SIM

WIND:3             // WIND code che indica
                   la disponibilità a ricevere
                   comandi AT

WIND:4             // WIND code che indica
                   l'avvenuta registrazione alla rete
                   sim pronta ad essere letta/scritta
```

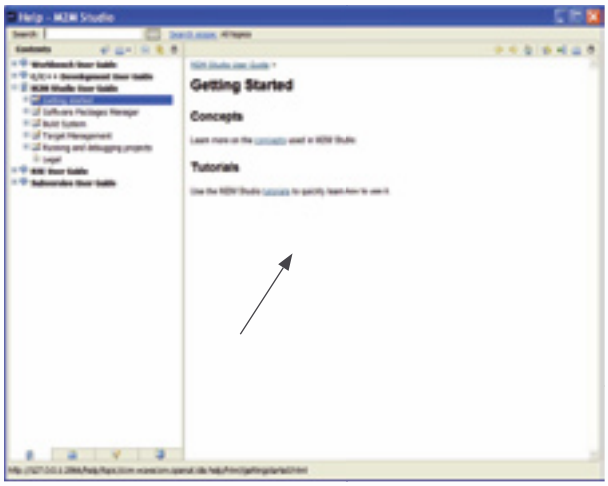


Fig. 4 *- Help dell'M2M Studio*

Fig. 3 *- M2M Studio Welcome*

A questo punto siamo sicuri che la nostra WCPU stia funzionando correttamente. Chiudiamo Hyperterminal.
Dopo aver installato l'ambiente di sviluppo M2M Studio, clicchiamo sull'icona con la rana per avviarlo.
All'inizio ci viene richiesto di definire una cartella predefinita dove salvare i progetti, che possiamo anche scegliere liberamente; a seguire appare la schermata di "Welcome" visibile nella **Fig. 3**.
Qui troviamo quattro opzioni corrispondenti alle seguenti possibilità: possiamo lanciare una breve panoramica sull'ambiente di sviluppo (Discover Open AT Concept), sfogliare la guida on-line (mediante la sezione First step with M2M Studio), oppure lanciare l'help in linea riguardante il set di API (vi accediamo facendo clic su Discover OpenAT APIs); infine, possiamo anche gestire i progetti creati o creare nuovi progetti semplicemente cliccando sul riquadro My Projects.
Per cominciare, facciamo clic su "First Step with M2M Studio": così facendo, abbiamo avviato il wizard che ci seguirà passo-passo nella creazione della nostra prima applicazione Open-AT. Una volta visualizzata la finestra di dialogo corrispondente clicchiamo su "tutorials" (**Fig. 4**) e siamo pronti per iniziare ad apprendere tutto quel che c'è da sapere sull'uso dell'ambiente di sviluppo.
Bene, per ora ci fermiamo qui; nella prossima puntata proseguiremo con il wizard, entreremo nel vivo delle funzionalità dell'M2M Studio e vedremo come sfruttare le possibilità da esso offerte.

Corso Wireless CPU

Utilizzo dell'M2M Studio di Wavecom

Entriamo nel vivo dell'ambiente operativo M2M Studio di Wavecom e vediamo come usarlo per iniziare a programmare le CPU Wireless.

dell'Ing.
NICOLA FRISO

Nella precedente puntata vi abbiamo fatto conoscere le CPU Wireless di Wavecom/Sierra Wireless elencando le varie famiglie disponibili in commercio distinte per prestazioni e funzionalità, ma anche iniziando a descrivere il sistema operativo Open AT e l'ambiente *M2M Studio* realizzati per esse. In particolare, ci siamo lasciati dopo aver spiegato come ottenere *l'M2M Studio* dal sito *www.wavecom.com*, in che modo inizializzare la Wireless CPU affinché sia pronta a interagire con esso e come accedere alle opzioni dello stesso *M2M Studio*. Abbiamo poi iniziato a descrivere l'uso del wizard. Ora è il momento di procedere riprendendo da dove ci siamo fermati, ossia dai *tutorial* del *wizard* accessibile dalla schermata Welcome dell'*M2M Studio*: una volta visualizzata la finestra di dialogo corrispondente, facciamo clic su *tutorials* e, nella finestra che si apre, clicchiamo sul link che consente di aprire i *cheat sheets* (che possiamo tradurre con "pagine

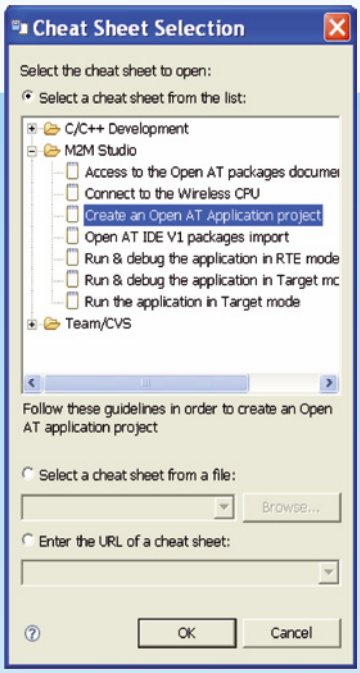
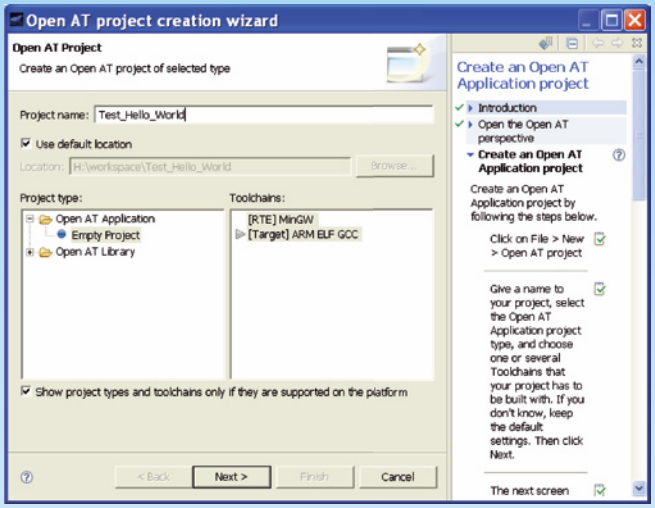


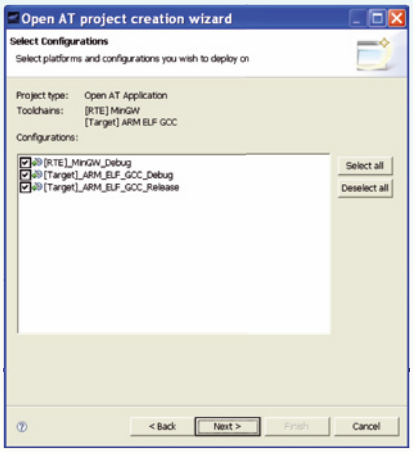
Fig. 1 *Cheat sheet selection.*

Fig. 2 - *La finestra di dialogo Project name del Wizard.*



furbe"...) i quali saranno la guida che ci permetterà, passo-passo, di creare il nostro primo progetto.
Si apre la finestra visibile nella **Fig. 1**. In essa selezioniamo *Create an Open AT Application project* e, a seguire, clicchiamo su OK. Seguiamo le istruzioni che compaiono sulla cheat page e clicchiamo su *Begin*.
A questo punto, seguendo i link proposti arriviamo alla schermata visibile nella **Fig. 2**. Diamo un nome al nostro primo progetto: ad esempio *Test_Hello_World*.
Nella finestra seguente (**Fig. 3**) lasciamo spuntate tutte le opzioni di configurazione. L'RTE (*Real Terminal Emulator*) è l'emulatore su PC sul quale è possibile testare l'applicazione in fase di debug attraverso il tool *miniGW_debug*. In alternativa si può eseguire il debug sul target (direttamente sulla wireless CPU) attraverso il tool *ARM_ELF_GCC_Debug* o programmare la WCPU attraverso il tool *ARM_ELF_GCC_release*.
Passiamo dunque alla finestra successiva, *Bynary project creation*, e selezioniamo l'esempio già pronto *Hello_World* (vedere **Fig. 4**). Clicchiamo su *Finish*: in pochi secondi il progetto viene creato.
Il *cheat sheet* ci indica che tutti i necessari passi sono stati compiuti (**Fig. 5**); ora clicchiamo sull'icona per ridurre la finestra di *cheat* (vedere freccia nella **Fig. 5**) e ci ritroviamo nell'ambiente di sviluppo *M2M Studio* con il codice del progetto caricato e pronto ad essere debuggato (**Fig. 6**). Sulla colonna di destra troviamo sempre i *cheat sheet* che ci fanno da guida, qualsiasi operazione vogliamo compie-

Fig. 3 - *La finestra Configurations del Wizard.*



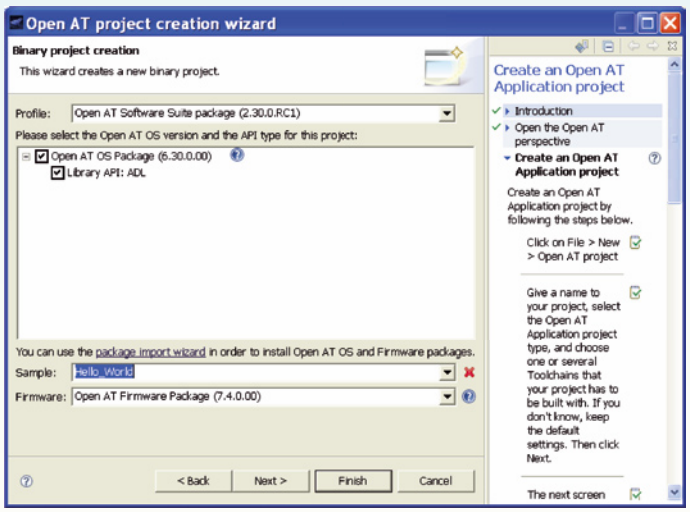


Fig. 4 *Wizard: la finestra Binary project creation.*

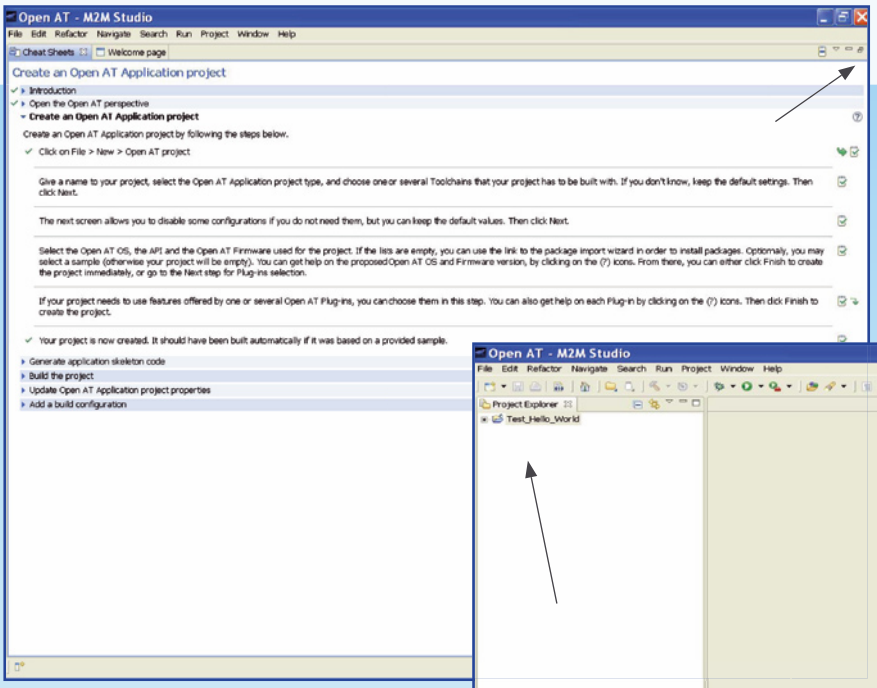


Fig. 5 - *I risultati della finestra Cheet Sheet.*

Fig. 6 - *Ambiente Open AT-M2M Studio.*

re. Se clicchiamo su *Test_Hello_World* possiamo esplorare il codice dell'applicazione *Hello World* (**Fig. 7**).
L'applicazione è composta da una direttiva *include adl_global.h* che deve essere sempre presente all'interno delle applicazioni Open AT e che serve per includere tutte le principali API della suite Open AT. A seguire viene riservata un'area di memoria di stack per le chiamate a funzione. L'applicazione è costituita da due funzioni, la funzione *HelloWorld_TimerHandler* e la funzione principale *adl_main* la quale deve essere sempre presente in ogni applicazione. La prima riga della funzione *adl_main* è un richiamo alla macro TRACE:

TRACE ((1, "Embedded : Hello World"));

L'effetto di questa macro è stampare la stringa nella finestra di shell del debugger. Il primo parametro è il livello di TRACE; sono possibili fino a 32 livelli di trace. Ogni livello di TRACE può essere attivato/disattivato a run time attraverso la finestra di *Remote Tasks Monitor* (**Fig. 8**) che si apre automaticamente quando lanciamo il debug. Questo è uno strumento molto

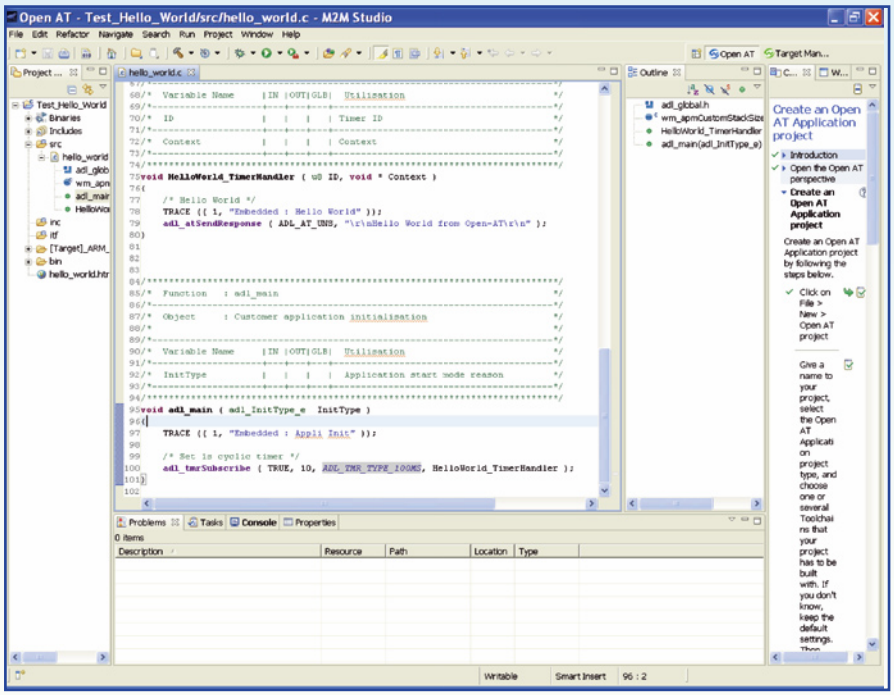


Fig. 7 - *Ambiente M2M Studio: il codice dell'applicazione.*

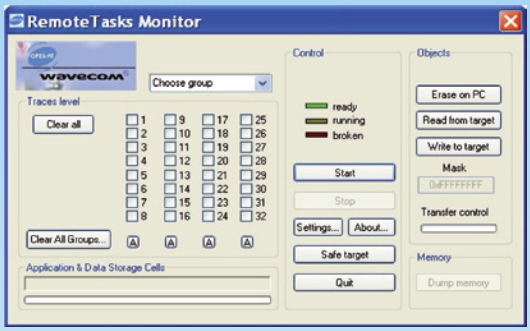


Fig. 8
La finestra Remote Tasks Monitor.

potente ed indispensabile per tener traccia del flusso del programma. Il secondo parametro della TRACE è la stringa da stampare. Sono consentiti, alla funzione *C printf*, anche parametri analoghi all'interno della stringa; ad esempio:

```
u16 var_int;
// dichiarazione di un
intero 16b bit

TRACE (( 1, "Stampa
un intero:%d",var_int ));
// stampa del contenuto della variabile var_int.
```

La seconda riga di codice all'interno della funzione *adl_main* inizializza un timer. Con questa funzione si attiva un evento timer, che scatta ogni secondo ed è associato alla funzione *HelloWorld_TimerHandler*. Il risultato è che ogni secondo il flusso di programma salta all'interno della funzione *HelloWorld_TimerHandler*. Il codice di esecuzione del timer stampa la stringa di TRACE *Embedded : Hello World* nella finestra di shell, la stessa stringa con la riga di codice:

```
adl_atSendResponse ( ADL_
AT_UNS, "\r\nHello World
from Open-AT\r\n" );
```

viene inviata sulla seriale di default UART 1 del modulo e quindi visualizzata sulla finestra di terminale. La differenza fra TRACE e *adl_atSendResponse* sta nel fatto che mentre con la macro TRACE la stringa è visualizzata solo sulla shell di debug, con la funzione *adl_atSendResponse* viene stampata anche sulla seriale e quindi la stessa stringa viene visualizzata anche in fase di run-time quando

Fig. 9 - *M2M Studio - Set RTE Debug.*

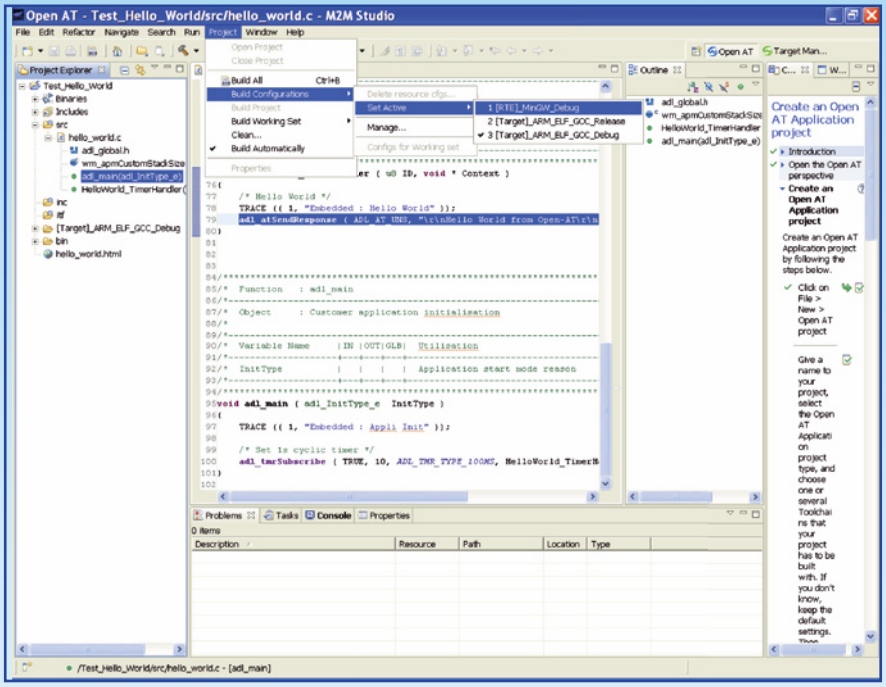


Fig. 10 - *M2M Studio Debug RTE.*

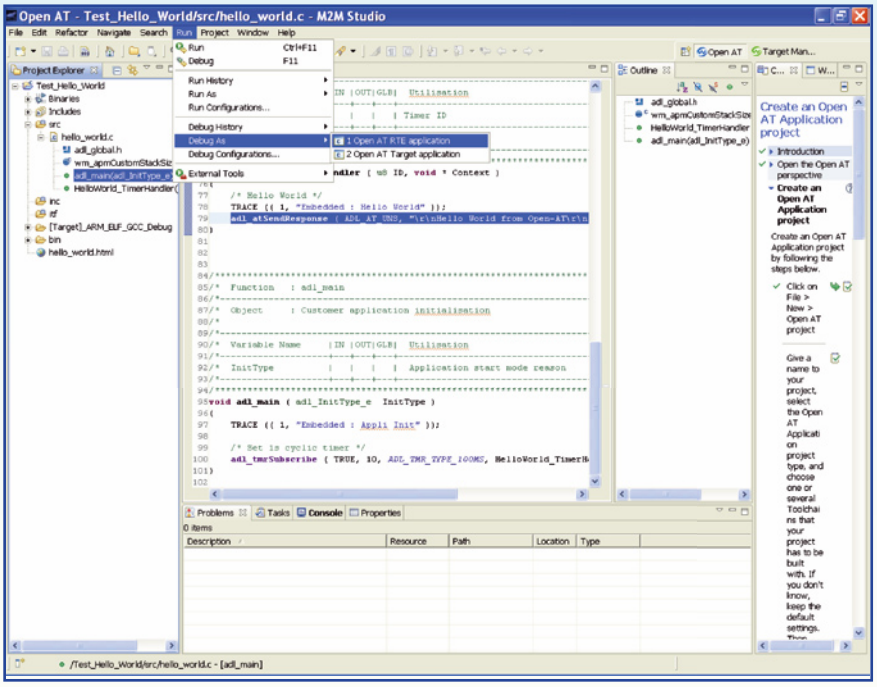
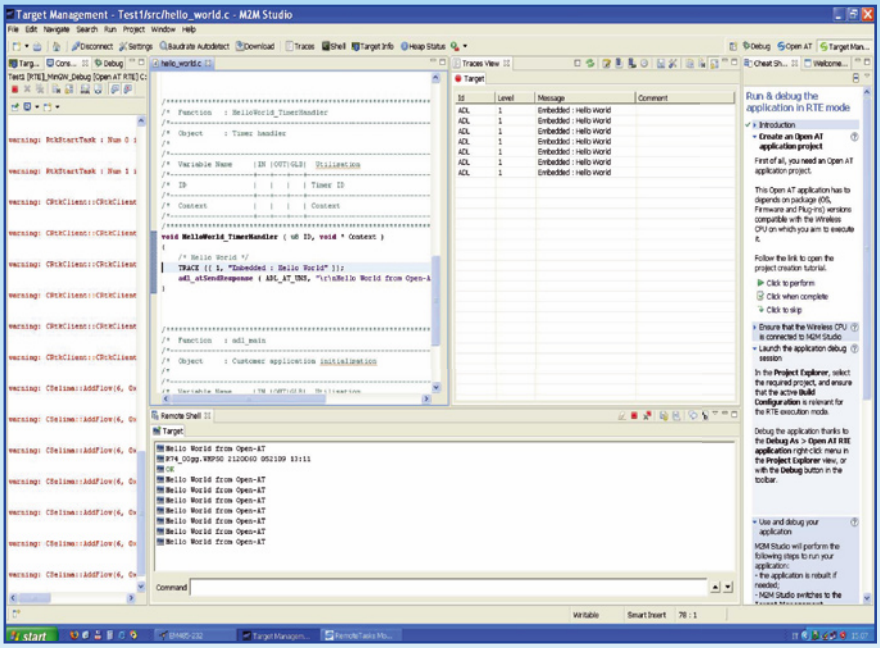


Fig. 11
Serial link Manager

Fig. 12- *La finestra di dialogo Target Management*



si avvia un applicativo tipo *hyper terminal* connesso alla UART 1 del modulo GSM.
Ora va detto che fin dall'inizio di questo corso abbiamo sottolineato un concetto importante: la funzione *adl_main* viene eseguita solo all'avvio della applicazione; non c'è alcun loop infinito tipo **while(1)** o **for(;;)** come succede nelle tipiche applicazioni per microcontrollori. Dovete infatti ricordare che stiamo lavorando all'interno di un sistema operativo. Tutte le risorse che si vogliono utilizzare, come timer, GPIO, interrupt ecc. vanno inizializzate (*sottoscritte*, nel gergo Open AT); ad ogni risorsa viene automaticamente assegnato un identificativo (ID) e viene richiesto di definire una funzione di *aggancio* (Handler) che viene messa in esecuzione dal S.O. nel momento opportuno. Nel caso del timer dell'esempio, tale funzione viene eseguita ogni secondo. Bisogna quindi pensare e sviluppare le applicazioni secondo questa filosofia. Si capisce quindi quanto sia importante, per noi, inserire le TRACE nel nostro programma, allo scopo di riuscire a seguirne il flusso in ogni momento. Ciò sarà più evidente nel seguito di questo corso, quando avremo modo di approfondire il concetto e di imparare a gestire nel modo corretto programmi più complessi che fanno uso di un elevato numero di eventi sottoscritti.
Bene, arrivati a questo punto proviamo a lanciare l'applicazione in modalità di Debug RTE; per farlo clicchiamo su *Project-> Build Configurations-> Set Active-> [RTE]_MinGW_Debug* per attivare la modalità di debug *Real Terminal Emulator* (**Fig. 9**).
Ora facciamo clic sul pulsante di compilazione oppure, dal menu, impartiamo il comando *Project-> Build All*; in basso, nella finestra *Console*, otteniamo il risultato della compilazione.
Nella finestra *Problem* troviamo gli eventuali errori e/o warning.
Adesso lanciamo il debug cliccando su *Run-> Debug As-> Open AT RTE Application* (vedere la **Fig. 10**).
Se la *target board* (Demoboard oppure il modulo "Augusto") è collegata correttamente al PC, si apre la finestra di *Remote Tasks Monitor* della **Fig. 8**. Se la UART cui è collegata la target board non è corretta, possiamo cambiarla cliccando sul *Serial Link manager* che si trova nella barra delle applicazioni sul desktop in basso a destra (vedere la **Fig. 11**).
Una volta aperta la COM corretta, avviamo il debugger cliccando sul pulsante *Start* della finestra *Remote Tasks Monitor*. Con questa finestra è possibile attivare/disattivare le TRACE a run-time oppure fermare l'applicazione ciccando su STOP. Selezioniamo *ADL** e la **Trace 1**. Il risultato di tale operazione, dopo aver impartito lo *Start*, è riportato in **Fig. 12**.
Sulla finestra di *Shell* e sulla finestra di *trace* vediamo comparire ogni secondo la stringa di *Hello World*.
Complimenti: avete appena fatto funzionare in debug la vostra prima applicazione sviluppata con Open AT! Adesso blocchiamo il flusso dell'applicazione cliccando su *Stop*. Poi ritorniamo all'editor premendo il pulsante

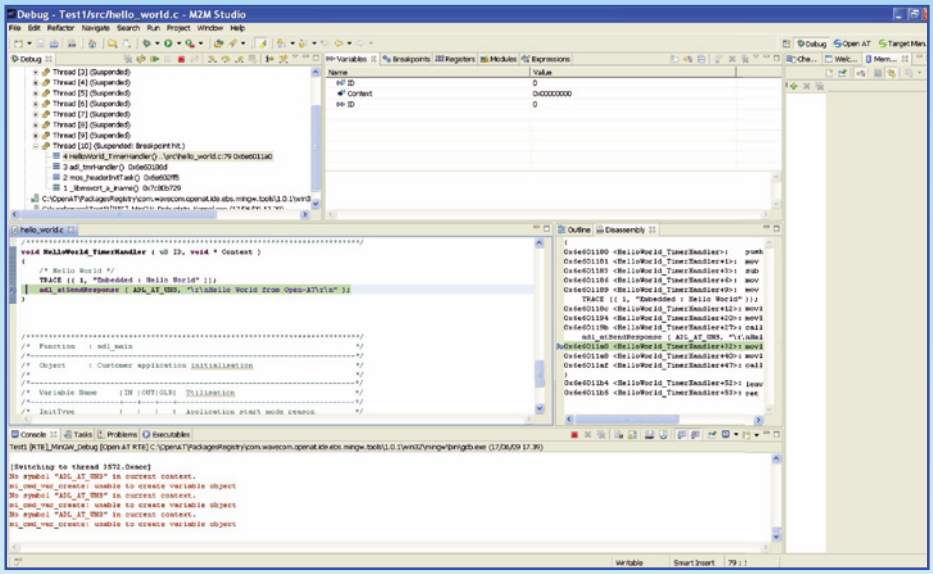


Fig. 13 - *Svolgimento del debug.*

"Open AT" in alto a destra e clicchiamo due volte sul bordo sinistro a fianco della riga di codice seguente allo scopo di inserire un breakpoint:

adl_atSendResponse (ADL_AT_UNS, "\r\nHello World from Open-AT\r\n");

Adesso facciamo clic sul pulsante di *Debug* (quello con l'insetto...) e lanciamo di nuovo il programma in modalità debug RTE. Il flusso del programma ora si interrompe ogni secondo, fermandosi sulla riga di codice indicata dal breakpoint (**Fig. 13**). Con il tasto funzione F6 proseguiamo passo-passo, debuggando riga per riga senza, quando viene incontrata una chiamata, entrare nelle funzioni. Con F5, invece, otteniamo lo stesso effetto entrando anche nelle chiamate a funzione. Con il tasto F8 riprendiamo il flusso normale. All'interno del debugger possiamo vedere anche il contenuto delle variabili, stringhe, classi o strutture che sono state dichiarate nel nostro codice. Finora abbiamo lavorato sempre in modalità DEBUG RTE, ma va detto che possiamo ripetere tutti i passi fin qui visti semplicemente selezionando la modalità debug TARGET (cliccare su *Project-> Build Configurations-> Set Active-> Target_ARM_ELG_GCC_DEBUG*). In questo caso il debug viene fatto direttamente sulla WCPU. La differenza tra le due modalità di debug sta principalmente nell'uso che viene fatto della memoria: nella modalità RTE la memoria programma e dati è emulata dal PC; sul target vengono inviati tutti i comandi necessari per muovere i GPIO e tutte le altre periferiche, ma il programma vero e proprio sta girando in modalità real time sul PC. Si ottiene così un debug molto rapido nell'esecuzione dei breakpoint e nella analisi dei contenuti di memoria. Invece nella modalità TARGET l'applicazione è realmente in esecuzione nella WCPU; quando si avvia questa modalità, infatti, il programma viene automaticamente caricato nella WCPU. In modalià TARGET si deve utilizzare il connettore JTAG invece della porta seriale; l'uso del JTAG è limitato ad applicazioni professionali e richiede il permesso esplicito della Wavecom e l'attivazione della porta JTAG, sempre da parte della Wavecom. Il modulo "Augusto" da noi utilizzato in questo corso è predisposto con un connettore JTAG e la WCPU è configurata per poter lavorare anche con il JTAG. Nelle prossime puntate vedremo come eseguire il debug delle nostre applicazioni utilizzando anche questo potente strumento. Finora abbiamo utilizzato il solo debugger; facciamo un ultimo sforzo e proviamo la nostra applicazione in modalità RELEASE, andando a scaricare ed eseguire l'applicazione direttamente sulla WCPU. Dalla finestra *Target Management* clicchiamo su "Download" e attendiamo che il programma venga scaricato. La CPU viene quindi automaticamente resettata e sulla shell viene inviato automaticamente il comando:

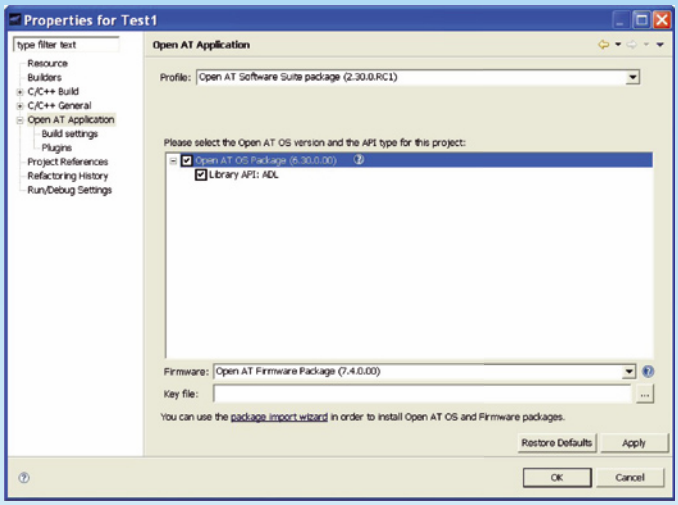
AT+WOPEN=1

che serve per mandare in esecuzione la predetta applicazione. Al massimo dopo 8 secondi (questo tempo è necessario al sistema operativo affinché carichi l'applicazione) dovrebbe comparire la scritta *Hello World from Open-AT* sulla *shell*, periodicamente ogni secondo. Il comando AT sopra indicato occorre digitarlo solo dopo aver scaricato una nuova applicazione nella target board. A questo punto possiamo anche spegnere e riaccendere la target board, perché il programma partirà comunque senza dover digitare alcun comando AT. Se chiudiamo l'*M2M Studio* e apriamo *Hyperterminal*, vedremo la scritta *Hello World from Open-AT* sulla finestra dello stesso *Hyper Terminal*, segno che la nostra applicazione sta girando sul target e non c'è più nulla che la leghi al PC, tranne il fatto che usa la seriale per inviare la stringa.
Se volete caricare altri esempi o altre applicazioni, occorre prima che cancelliate questa applicazione (ossia quella che sta girando); potete fare ciò con i seguenti comandi AT:

AT+WOPEN=3 // cancella la memoria flash object della applicazione
AT+WOPEN=4 // cancella la memoria flash A&D della applicazione

Alla trattazione della memoria sarà dedicata un'apposita puntata di questo corso; per ora accontentatevi di sapere che la "A&D" è la memoria Flash dove risiede l'applicazione e che la "object" è la memoria Flash, eventualmente utilizzata dalla nostra applicazione, destinata a salvare i dati.
Con la procedura descritta in questo articolo introduttivo potete testare altri esempi tra i molti disponibili in fase di creazione del progetto. Per lo studio del codice degli esempi conviene sempre avere sottomano la documentazione delle API ADL, che è disponibile on-line cliccando sul tasto destro nella finestra del *Project Explorer* di *M2M Studio*, quindi selezionando *Properties* e facendo clic sull'icona con il punto di domanda. Ciò permette di aprire la pagina con i collegamenti a tutte le guide Open AT (**Fig. 14**). La guida più importante è la *Open AT Development Guide*. La stessa guida la si può trovare anche in locale

Fig. 14 *- La finestra di dialogo Project Properties.*



nella cartella di installazione dell'Open AT: *C:\OpenAT\PackagesRegistry\com.wavecom.openat.ide.spm.lib.os.model\6.30.0.00\doc*
Il file è siglato *ADL_User_Guide.pdf*.
Per ogni applicazione di esempio invece, è disponibile un file html a corredo, nel quale vengono spiegati l'utilizzo e le funzioni di libreria utilizzate. I progetti di esempio con la relativa documentazione si possono trovare in locale nella seguente cartella: *C:\OpenAT\PackagesRegistry\com.wavecom.openat.ide.spm.lib.os.model\6.30.0.00\ADL\samples.*
Nella prossima puntata analizzeremo più nel dettaglio la pin-out e l'hardware della WCPU WMP50, andando a costruire una demoboard per essa, che utilizzeremo per tutti i nostri test. Chi volesse maggiori chiarimenti sulla materia, potrà consultare il file *ADL_User_Guide.pdf* e la Guida on-line *M2M Studio 1.0*, entrambi scaricabili dal sito Internet di Wavecom. Nel rimandarvi alla prossima puntata del corso, vi auguriamo buon lavoro e buon divertimento con Open AT!

Corso Wireless CPU

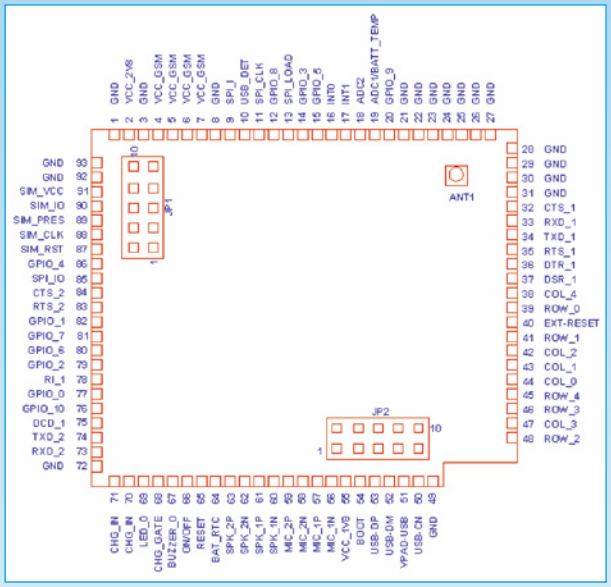
La demoboard “EVK Augusto”

Conosciamo la demoboard equipaggiata con WMP50, analizzandone l’hardware e vedendo come costruirla.

dell’Ing.
NICOLA FRISO

Nelle precedenti puntate di questo corso abbiamo introdotto la famiglia di Wireless CPU di Wavecom, esaminando in particolare i dispositivi WCPU siglati WMP50 e WMP100. Abbiamo anche configurato l’ambiente di sviluppo M2M Studio 1.0 e lanciato la nostra prima applicazione Open-AT “Hello World”. Adesso vogliamo focalizzare la nostra attenzione sugli strumenti hardware necessari a mettere alla prova le suddette wireless CPU: andremo a studiare la demoboard EVKAUGUSTO che ci servirà per testare le funzionalità del modulo Augusto sviluppato da Shitek Technology. Questo modulo è già stato introdotto nella puntata precedente e non è altro che una scheda di supporto per le Wireless CPU WMP50 che riporta sui quattro lati, su un connettore pin-strip passo 2mm, tutti i pin BGA presenti sulla WCPU. Il modulo Augu-

Fig. 1 - *Pin-out modulo Augusto.*



batterie 3,6 V forniti da una cella al litio ricaricabile oppure da tre celle NiMH o NiCd. Nella sezione POWER è riportato anche il circuito per gestire la ricarica del pacco batterie.

Sezione 2: GSM MODULE – Questa parte di schema riporta il modulo Augusto e i connettori passo 2,54 mm posizionati attorno al modulo per testare con l'oscilloscopio lo stato di tutti i pin oppure per collegare direttamente altri componenti esterni alla demoboard. Nella stessa sezione sono presenti dei trimmer per testare gli ingressi analogici, un buzzer collegato all'uscita BUZZER del modulo Augusto ed i pulsanti di ON/OFF, BOOT e RESET.

sto integra anche il porta SIM, il connettore d'antenna, la memoria combo Flash-RAM esterna, l'oscillatore RTC a 32,768 kHz, il connettore di programmazione/debug seriale, il connettore di programmazione/debug JTAG e i componenti di filtro necessari per separare la parte di alimentazione digitale della WCPU dalla sezione analogica RF.
Nella **Fig. 1** è illustrata la pin-out del modulo Augusto, mentre nella **Tabella 1** è riportata la descrizione delle funzionalità dei relativi pin; per maggiori dettagli consultate il data-sheet del modulo Augusto.
La demoboard EVKAUGUSTO, oltre al connettore dove inserire il modulo Augusto mette a disposizione tutte le periferiche utili per testare le proprie applicazioni.
Come si evince dallo schema a blocchi visibile nella **Figura 2**, la demoboard è divisa in sette sezioni, descritte qui di seguito.

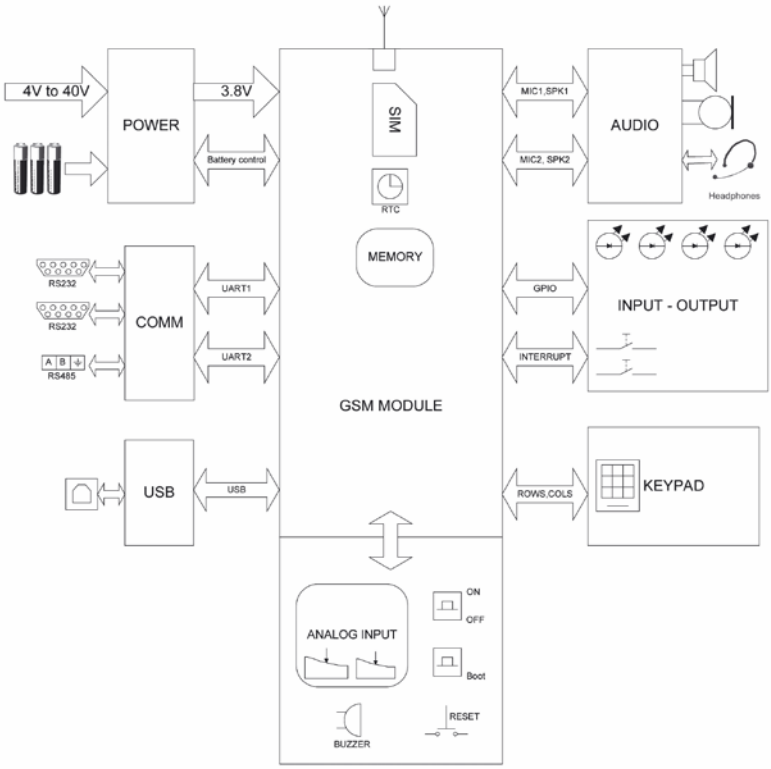


Fig. 2 - *Schema a blocchi EVK-AUGUSTO.*

Sezione 1: POWER – Questa è la sezione di alimentazione della scheda; comprende un DC/DC converter switching con ingresso da 4,5 V a 40 V e uscita impostata per fornire i 3,8 V necessari al modulo GSM. In alternativa è possibile alimentare la scheda con un pacco

Sezione 3: COMMUNICATIONS – Questa sezione contiene gli integrati e le porte di comunicazione seriale; nello specifico, sono presenti due porte seriali RS232, una completa a 8 fili (standard ITU-T V.24) e l'altra a 4 fili (TX, RX, CTS, RTS).

Sezione 4: USB – Questa parte di circuito è dedicata esclusivamente alla gestione della porta USB slave del modulo Augusto.

Sezione 5: KEYPAD - La demoboard dispone di un tastierino 3x2 tasti collegati a matrice sui pin predisposti alla gestione tastiera della wireless CPU montata sul modulo Augusto.

Sezione 6: AUDIO – Sono stati inseriti due connettori audio collegati tramite opportuni filtri ai due canali audio del modulo Augusto. I due connettori sono predisposti per il collegamento di altoparlante e microfono. Uno dei canali audio è predisposto per realizzare un sistema vivavoce sfruttando il cancellatore d'eco integrato nella WCPU montata sul modulo Augusto.

Sezione 7: INPUT-OUTPUT
In questa parte di circuito, per verificare il funzionamento dei pin di GPIO (General Purpose Input-Output) e interrupt, sono stati predisposti 2 pulsanti per testare i due pin di interrupt e 4 uscite a transistor collegate ad altrettanti led, pilotate da 4 pin di I/O generici della WCPU.
Andiamo ora ad analizzare nel dettaglio le sezioni in cui è stato suddiviso il circuito, con la premessa che i componenti si riferiscono agli schemi elettrici che trovate nel nostro sito e che, per ragioni di spazio, non abbiamo potuto pubblicare qui.

***Tabella 1** - Descrizione pin modulo Augusto.*

Pin Name	Number	Description	I/O	Voltage Domain
GND	1,3,8,21,22,23,24, 25,26,27,28,29,30, 31,49,72,92,93	Reference ground		
VCC_GSM	4,5,6,7	Power supply	I	3.2 V to 4,5 V 3,8 V raccomandated
VCC_2V8	2	Output Digital Power Supply 2,8V (max 15mA)	O	2,74 V to 2,86 V
VCC_1V8	55	Output Digital Power Supply 1,8V (max 60mA)	O	1,76 V to 1,94 V
ON/~OFF	66	ON/~OFF Min Time ON > 80 ms	I	VILmax = VCC_GSM x 0,2 VIHmin = VCC_GSM x 0,8
~RESET	65	Reset (active low) Min Time Reset 40 ms	I/O	1V8 open-drain with internal pull-up
~EXT-RESET	40	External Reset (active low) for external devices Tmin = 35ms after 122us from RESET	O	1V8 push-drain
BOOT	54	Download mode selection Set to VCC_1V8 to enter in boot mode	I	1V8
BUZZER_0	67	Buzzer output (max 100mA)	O	VCC_GSM Open Drain
BAT_RTC	64	Back-up power supply for RTC. Left open if RTC is not used. If VBATT is present the back-up battery is charged by internal 2,5V regulator.	I/O	Inp Voltage: 1,85 V to 3,0 V, Inp current: 3,3 µA Out Voltage: 2,45 V Out Current: 2 mA
CHG_IN	71,70	Charger input voltage	I	4,8 V minimum for Imax=800 mA 6V maximum fot I=0 mA
CHG_GATE	68	Charge transistor control output	O	Max 10m A
LED_0	69	Open drain output signalling led	O	Max 8 mA
SIM_VCC	91	SIM power supply, supported 1,8 V and 3,0 V SIM		Vtyp=1,8V or Vtyp=3,0V Max 10 mA
SIM_IO	90	SIM data	I/O	V_{OH}=0,9xSIMVCC V_{OL}=0,4V V_{IH}=0,7xSIMVCC V_{IL}=0,4V
SIM_PRES	89	SIM presence	I	1,8V
SIM_CLK	88	SIM clock	O	3,25 MHz max freq. 2,9V/1,8 V
~SIM_RST	87	SIM reset	O	2,9/1,8 V
CTS_1	32	Clear to send (UART 1) (PC view)	O	2,8 V
RXD_1	33	Receive serial data (UART1) (PC view)	O	2,8 V
TXD_1	34	Transmit serial data (UART1) (PC view)	I	2,8 V
RTS_1	35	Ready to send (UART 1) (PC view)	I	2,8 V
DTR_1	36	Data terminal ready (UART 1) (PC view)	I	2,8 V
DSR_1	37	Data set ready (UART 1) (PC view)	O	2,8 V
DCD_1	75	Data carrige detect (UART 1) (PC view)	O	2,8 V
RI_1	78	Ring indicator (UART 1)	O	2,8 V
CTS_2	84	Clear to send (UART 2) (PC view)	O	1,8 V
RXD_2	83	Receive serial data (UART2) (PC view)	O	1,8 V
TXD_2	74	Transmit serial data (UART2) (PC view)	I	1,8 V
RTS_2	73	Ready to send (UART 2) (PC view)	I	1,8 V
USB-DP	53	USB Differential data positive	I/O	V_{Min}=3 V V_{Typ}=3,3 V V_{Max}=3,6 V $I_{Vpadusb}$=8 mA

POWER

Con riferimento allo schema elettrico "Power Supply", abbiamo un ingresso di tipo jack J1, cui seguono un transil (D5) messo a protezione dai transitori di tensione, il diodo D3 di protezione dall'inversione della polarità e i condensatori di ingresso (C38,C39,C37) dello stadio switching. L'integrato U7 è un regolatore switching di tipo step-down LM2596S-ADJ della National, utilizzato per ricavare la tensione di 3,8 V necessaria ad alimentare correttamente la wireless CPU WMP50 o WMP100 montata sul modulo Au-

... prosegue Tabella 1 - *Descrizione pin modulo Augusto.*

Pin Name	Number	Description	I/O	Voltage Domain
USB-DM	52	USB Differential data negative	I/O	V_{Min}=3V V_{Typ}=3,3V V_{Max}=3,6V $I_{Vpadusb}$=8 mA
VPAD-USB	51	USB Power Supply	I	V_{Min}=3 V V_{Typ}=3,3 V V_{Max}=3,6 V $I_{Vpadusb}$=8 mA
USB-CN	50	USB Connect (high or low speed)	O	V_{Min}=3V V_{Typ}=3,3 V V_{Max}=3,6 V $I_{Vpadusb}$=8 mA
USB-DET	10	USB Detection	I	1,8 V
SPI_LOAD	13	SPI load	O	2,8 V
SPI_CLK	11	SPI serial clock	O	2,8 V
SPI_I	9	SPI serial data input (only input)	I	2,8 V
SPI_IO	85	SPI serial data input and output	I/O	2,8 V
GPIO_0	77	General purpose input/output	I/O	2,8 V
GPIO_1	82	General purpose input/output	I/O	2,8 V
GPIO_2	79	General purpose input/output	I/O	2,8 V
GPIO_3	14	General purpose input/output	I/O	2,8 V
GPIO_4	86	General purpose input/output	I/O	2,8 V
GPIO_5	15	General purpose input/output	I/O	2,8 V
GPIO_6	80	General purpose input/output	I/O	Open-Drain
GPIO_7	81	General purpose input/output	I/O	Open-Drain
GPIO_8	12	General purpose input/output	I/O	2,8 V
GPIO_9	20	General purpose input/output	I/O	2,8 V
GPIO_10	76	General purpose input/output	I/O	2,8 V
INT_0	16	External interrupt 0	I	2,8 V
INT_1	17	External interrupt 1	I	2,8 V
ADC1/ BATT_TEMP	19	Analog to digital converter 1 or monitor external temp. of battery	I	0÷2V 10 bit resolution
ADC2	18	Analog to digital converter 2	I	0 to 2V 10 bit resolution
ROW_0	39	Row scan keypad	I/O	1,8 V
ROW_1	41	Row scan keypad	I/O	1,8 V
ROW_2	48	Row scan keypad	I/O	1,8 V
ROW_3	46	Row scan keypad	I/O	1,8 V
ROW_4	45	Row scan keypad	I/O	1,8 V
COL_0	44	Column scan keypad	I/O	1,8 V
COL_1	43	Column scan keypad	I/O	1,8 V
COL_2	42	Column scan keypad	I/O	1,8 V
COL_3	47	Column scan keypad	I/O	1,8 V
COL_3	48	Column scan keypad	I/O	1,8 V
MIC_1N	56	Microphone 1 negative input	I	Analog (vedere sezione audio)
MIC_1P	57	Microphone 1 positive input	I	Analog (vedere sezione audio)
MIC_2N	58	Microphone 2 negative input	I	Analog (vedere sezione audio)
MIC_2P	59	Microphone 1 positive input	I	Analog (vedere sezione audio)
SPK_1N	60	Speaker 1 negative input	O	Analog (vedere sezione audio)
SPK_1P	61	Speaker 1 positive input	O	Analog (vedere sezione audio)
SPK_2N	62	Speaker 2 negative input	O	Analog (vedere sezione audio)
SPK_2P	63	Speaker 2 positive input	O	Analog (vedere sezione audio)

gusto. La tensione di uscita è impostata dalla rete di feedback formata dalle resistenze R72 ed R73. L'integrato accetta un range di tensioni di ingresso da 4 a 40 V ed eroga fino a 3 A, con una frequenza di switching pari a 150 kHz. Questo chip ben si presta per alimentare carichi di tipo GSM, perché eroga una corrente più che sufficiente (3 A) per sopportare i picchi di corrente da 2 A che si verificano durante la fase di comunicazione dati o voce con la rete GSM/GPRS. Inoltre la bassa frequenza di switching consente una maggiore tolleranza sul ripple ammesso nella tensione di uscita (vedere l'apposito riquadro e il datasheet del modulo Augusto).

Per l'integrato si raccomanda di utilizzare condensatori di ingresso e di uscita a bassa resistenza serie. Nel nostro caso abbiamo utilizzato due condensatori di ingresso low-esr C38 e C39 di tipo elettrolitico e un condesatore di uscita tantalio low-esr; un altro condensatore del genere è già montato sul modulo Augusto, tra il pin di VCC_GSM e GND.

Il led DL8 rimane acceso se sono presenti i 3,8 V di alimentazione del modulo Augusto.

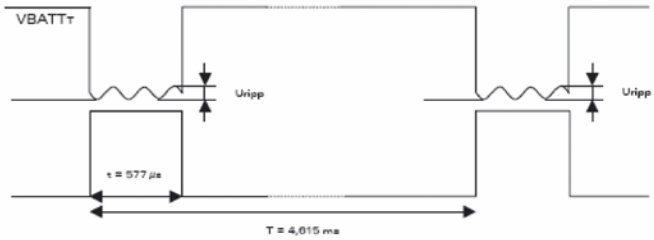
Nello schema è riportato anche un altro jack, siglato J2, connesso con il pin CHG-IN del modulo Augusto. Collegando a questo ingresso una tensione di 5 V proveniente, ad esempio, da un alimentatore per carica di cellulari (6V-800mA al massimo), si attiva il circuito di ricarica della batteria. In questa modalità occorre collegare al connettore CN5 una batteria. L'algoritmo interno può essere impostato per caricare due tipi di batterie: NiCd e NiMH oppure al litio.

Per quanto riguarda la cella al litio da 3,6 V, deve essere dotata di circuito di protezione interno contro sovratensioni e sovracorrenti e di un sensore di temperatura interno al pacco batterie, di tipo NTC da 10 kohm a 25 °C. Il circuito di ricarica della batteria funziona in questo

modo: nel momento in cui si collegano i 5 V a J2 il diodo D6 manda alto il pin 5 di ON/OFF dello switching U7. In questo modo anche se la tensione di ingresso MAIN POWER su J1 è rimasta inavvertitamente collegata, lo switching viene spento e la VGSM è data solo dalla batteria collegata a CN5 oppure a JP13 (sono connettori di tipo diverso, posti in parallelo). La WCPU tramite il pin VCC_GSM controlla la tensione di alimentazione collegata con la batteria.
Se si usa un pacco batteria con sonda di temperatura integrata, tramite R78 viene polarizzata la sonda NTC del pacco batterie e il segnale ricavato dal partitore R78 con il parallelo di R79 e della sonda NTC, viene inviato al pin di ingresso di controllo BAT_TEMP. I valori di R78 e R79 sono calcolati per avere al massimo 2 V sul pin BAT_TEMP.
Il pin CHG-GATE controlla in corrente la base del transistor tipo PNP di potenza Q13. È tramite questo transistor che viene fornita la corrente necessaria alla ricarica della batteria. Quando la tensione della batteria è compresa tra 2,8 V e 3,2 V si attiva la modalità di pre-carica, in cui la corrente fornita alla batteria è costante e pari a 50 mA. In questa modalità il transistor Q13 non viene pilotato e la corrente costante è fornita direttamente dalla WCPU attraverso il pin VBATT_BB, collegato alla VCC_GSM del modulo Augusto. Durante la fase di pre-carica il led presente sul modulo Augusto (collegato al pin LED0) lampeggia. Quando, invece, la tensione è compresa tra 3,2 V e 4,3 V (massimi) l'algoritmo di ricarica gestisce la curva di tensione e corrente a seconda del tipo di batteria che si sta utilizzando. Con il comando AT+WBCM si possono impostare tutti i parametri relativi alla ricarica della batteria compreso il tipo di batteria utilizzato (NiCd/NiMH oppure al litio).
Se la tensione del pacco batterie è inferiore a 2,8 V, l'algoritmo di ricarica non viene eseguito, perché la batteria viene considerata completamente scarica; il modulo Augusto non funziona. Facciamo notare che il circuito di ricarica proposto (che è anche quello consigliato dalla Wavecom e riportato nel data-sheet) è utile solo nel caso in cui l'applicazione che stiamo sviluppando si rivolge ad un dispositivo portatile alimentato a batteria, come ad esempio un palmare o un cellulare che viene di tanto in tanto ricaricato collegandolo ad una fonte di alimentazione esterna.
Se invece si desidera costruire un'applicazione che normalmente viene sempre alimentata

Alimentazione **modulo Augusto**

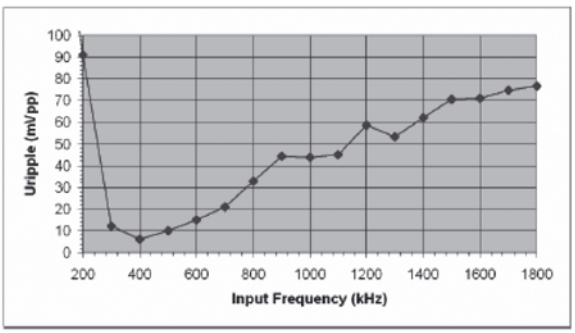
Segnale di alimentazione durante i burst di trasmissione:



In modalità di comunicazione GSM/GPRS classe 2, il modulo emette burst della durata di 577 µs ogni 4,615 ms.
In modalità di comunicazione GSM/GPRS classe 10, il modulo emette burst della durata di 1.154 µs ogni 4,615 ms.
Il range di tensione ammesso VBATT = VCC_GSM varia da 3,2V a 4,5V. Durante i burst la tensione non deve scendere al di sotto dei 3,2V pena lo spegnimento del modulo. Tensioni superiori a 4,5V possono invece danneggiare il modulo.
Il massimo ripple ammesso dipende dalla frequenza di switching ed è riportato nella tabella seguente.

Maximal voltage ripple (Uripp) versus Frequency

Freq. (kHz)	U_{ripp} Max (mVpp)	Freq. (kHz)	U_{ripp} Max (mVpp)	Freq. (kHz)	U_{ripp} Max (mVpp)
<100	100	800	33	1500	70
200	91	900	44	1600	71
300	12	1000	44	1700	75
400	6	1100	45	1800	77
500	10	1200	59	>1900	80
600	15	1300	53		
700	21	1400	62		

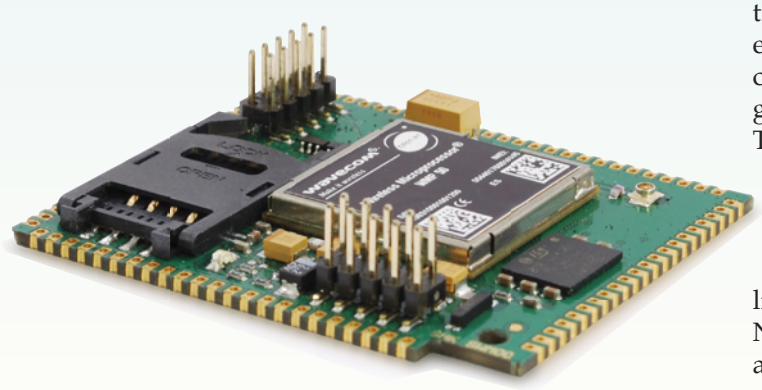


for f<100kHz Uripp Max = 100mVpp
for f>1800kHz Uripp Max = 80mVpp

da una fonte esterna fissa e, solo quando questa viene a mancare si passa all'alimentazione a batteria, allora tale circuito non è indicato; occorre far gestire la carica della batteria da un integrato esterno e non dall'algoritmo di ricarica implementato all'interno del modulo. Per maggiori dettagli su questa modalità di ricarica si consulti l'application note di Wavecom "EXTERNAL BATTERY CHARGING SOLUTION WM_DEV_MOD_APN_002" scaricabile dal sito Wavecom.

GSM MODULE

Al centro dello schema è riportato il modulo Augusto U6, ad esso è collegato un pulsante di reset P9 al pin di RESET con un varistore VR14 che serve a limitare le sovratensioni dovute ad eventuali scariche elettrostatiche ESD. Sono presenti due interruttori a slitta miniaturizzati: M2 per gestire il pin di ON/OFF e M3 per gestire il pin di BOOT.
Il pin BUZZER0 è un'uscita open-drain pilotata in PWM che consente di gestire direttamente un buzzer. Il jumper JP8 in serie alla linea BUZZER0 è stato inserito qualora si voglia staccare il cicalino montato sulla demoboard ed utilizzarne uno esterno. Con il comando AT+WTONE è possibile generare toni sul buzzer di diversa frequenza. I due ingressi analogici ADC1 e ADC2 sono collegati a due partitori di tensione R70-TR1 ed R71-TR2. La tensione si può variare tramite i trimmer da 0 a 2 V. Il jumper JP11 consente di collegare il segnale ADC1/BAT_TEMP al trimmer oppure alla sonda NTC della batteria mentre, se non viene montato, è possibile collegare un segnale esterno (0V÷2V) direttamente sul pin ADC1.
Il jumper JP12 inserito in serie alla linea ADC2 serve se si desidera scollegare il trimmer ed utilizzare un segnale esterno (0÷2 V) da applicare direttamente al pin dell'ingresso analogico ADC2. Se Augusto monta una WCPU WMP50 è disponibile un solo ingresso analogico ADC1, mentre se invece monta una WCPU WMP100, allora sono disponibili entrambi gli ingressi analogici.
I connettori strip passo 2,54 mm siglati JP6, JP7, JP9, JP10 sono stati posizionati tutto intorno al modulo Augusto e servono per accedere direttamente ai pin del modulo e per collegare eventuali dispositivi esterni alla demoboard.
Una nota importante: qualora si colleghino componenti o dispositivi esterni alla demoboard, occorre accertarsi che i valori di tensione ai pin del modulo Augusto non superino le specifiche riportate nel data-sheet. La massima tensione ammessa per quasi tutti i pin è pari a 2,8 V; se si supera questo valore si rischia il danneggiamento della WCPU montata sul modulo.



COMMUNICATIONS

In questa sezione dello schema sono riportate le porte di comunicazione seriale collegate alla UART1 e UART2 del modulo Augusto. La porta USB è stata disegnata su una sezione a parte per questioni di spazio e leggibilità dello schema. La WCPU (WMP50 o WMP100) montata sul modulo Augusto dispone di due porte seriali. La UART 1 accetta tensioni comprese tra 0 e 2,8 V, mentre la UART 2 va da 0 a 1,8 V. I segnali TXD1, RXD1, CTS1, RTS1 della porta UART1 sono portati anche sul connettore JP1 di Augusto e servono anche per il debug o la programmazione della WCPU. La porta UART1 è una seriale completa conforme allo standard ITU-T V.24 e tutti i suoi segnali sono collegati all'integrato U1 (MAX3237) che è un transceiver RS232 con 5 pin di trasmissione e 4 pin di ricezione, al quale è collegata una connessione seriale standard DB9 per il collegamento con un PC.
Tramite questa porta, il modulo Augusto può essere visto ed utilizzato da un PC come un modem wireless.
La tensione di alimentazione dei transceiver U1 e U3 è fornita dai due regolatori lineari LDO (Low Drop-Out) LP2951 della National. U2 fornisce i 3 V necessari per alimentare il MAX3237, mentre U4 fornisce la

tensione di 1,8 V necessaria per alimentare il transceiver U3 (LTC2804).
Il funzionamento dei due circuiti di pilotaggio di U1 e U3 è identico, quindi vediamo quello del solo U1: il transistor Q1 serve per pilotare il pin 3 SHD (attivo alto) di spegnimento di U2. Questo segnale viene mantenuto a livello alto da R16, collegata tra VCC_GSM e il collettore di Q1.
Quando il modulo Augusto si accende e riconosce che la sua tensione di alimentazione VCC_GSM è stabile attiva il regolatore lineare interno alla WCPU, che genera i 2,8 V collegati, tramite R18, con la base di Q1.
Il transistor allora entra in conduzione e manda a livello basso il pin SHD; U2 si accende e genera i 3 V necessari per alimentare la periferica di comunicazione. Tale procedura garantisce che non vi sia tensione ai pin delle porte UART del modulo Augusto prima della sua accensione; questa precauzione andrebbe presa per tutti i pin della WCPU, ai quali non deve essere applicata tensione senza che prima la WCPU non sia correttamente alimentata. Il led DL1 accesso indica la presenza dei 3 V necessari per alimentare U1.
Il segnale EN_UART1 è gestito, in modo analogo, dal transistor Q2 e serve per abilitare in modo automatico U1 quando la WCPU genera la tensione VCC_2V8.
L'interruttore M1 è collegato al pin SHDN (attivo basso) di U1 e consente lo spegnimento manuale dell'integrato. Il led DL3 acceso indica che SHDN, pin 14 di U1, è alto e il MAX3237 è attivo e funzionante.
La porta UART2 lavora nel range 0÷1,8 V ed è collegata con i segnali TXD2, RXD2, CTS2, RTS2 tramite U3, che è un transceiver della Linear Technology siglato LTC2804, operante a 1,8 V. Così facendo, la UART2 è resa disponibile su un connettore seriale DB-9 standard per il collegamento con il PC o ad altri dispositivi.
L'interruttore M4 agisce sul pin di ON/OFF di U3 e permette lo spegnimento manuale dell'integrato; lo stesso interruttore è collegato alla base del transistor Q4. Quando sulla base di quest'ultimo è presente il livello logico alto, si accende il led DL2, che indica che la UART 2 è attiva e funzionante.

***Tabella 2** - Comandi AT.*

Tipo	Comando	Risposta
Carica batterie	AT+WBCM? Legge lo statto della batteria.	0,0,4200,3200,4700,1438,601,1,2000,0 OK
	AT+WBCM=4,0 Imposta l'algoritmo di ricarica per celle NiCd,NiMH	OK
	AT+WBCM=4 Legge la modalità di ricarica impostata	+WBCM: 4,0 OK La modalità è NiCd/NiMh
	AT+WBCM=5 Verifica la connessione con il carica batterie	+WBCM: 5,1 Il carica batterie è collegato
	AT+WBCM=1,1 Inizia la ricarica con l'indicazione di carica attiva	OK +CME ERROR: 563 Carica non partita causa errore, ma codici di indicazione attivi
	AT+WBCM=1,0 Inizia la ricarica senza indicazione di carica	OK
	AT+WBCM=2 Chiede lo stato della batteria durante la carica	+WBCI: 2,4110 OK La tensione di batteria è 4.11V
		+WBCI:1 La tensione ha raggiunto il massimo livello. La batteria è considerata carica.
	AT+WBCM=0,1 Abilita l'indicazione di carica	OK
	AT+WBCM=3,0,4200,3500,4650...1,500,200 Configura i parametri di ricarica (in questo caso per la NiCd/NiMH)	+WBCI: 3,4195 La tensione di batteria è 4.195V OK
Buzzer	AT+WTONE=1,1,300,-1510,50 Suona un tono	OK
	AT+WTONE=0 Stop tono	OK
Keypad	AT+WHCNF=0,2 Attivazione keypad	+WHCNF: 0,1 OK Keypad è attivo
	AT+CMER=,1 Abilita segnalazione tasto premuto	OK
		+CKEV: 4,1 +CKEV: 4,0 Il tasto 4 è stato premuto e rilasciato

USB

Il modulo Augusto riporta sul pin-strip tutti i segnali della WCPU necessari per gestire una connessione USB secondo lo standard USB 2.0. Precisiamo che la WCPU permette solo una connessione USB di tipo slave (cioè Augusto viene visto come una periferica dal PC),

.. prosegue Tabella 2 - *Comandi AT.*

Tipo	Comando	Risposta
Interrupts	AT+WIPC=? Visualizza le configurazioni disponibili	+WIPC:(0-3),("INT0","INT1","INT2"),(0-2),(0-2),(1-7) OK
	AT+WIPC=1,"INT0",1,2,1	OK INT0 configurato sul fronte di salita con antirimbalzo
	AT+WIND=2048 Abilita indicazione degli interrupt	OK +WIND: 12,"INT0" Interrupt su INT0
	AT+WIPC=0,"INT0"	OK Rilascia l'INT0
GPIO	AT+WIOM=2 Legge lo stato dei pin di GPIO	+WIOM: "GPIO1",4 +WIOM: "GPIO2",4 +WIOM: "GPIO3",4 +WIOM: "GPIO4",3 +WIOM: "GPI1",4 +WIOM: "GPO1",4 OK
		GPIO1,GPIO2,GPIO3,GPI1 e GO1 sono disponibili, GPIO4 è già utilizzato
	AT+WIOM=1,"GPIO1",0	OK Alloca GPIO1 come ingresso
	AT+WIOM=1,"GPIO2",1	OK Alloca GPIO2 come uscita (valore iniziale 0)
	AT+WIOM?	+WIOM: "GPIO1",0 +WIOM: "GPIO2",1,0 OK
	AT+WIOR="GPIO05" Legge il valore di GPIO5	+WIOR: 0 OK Il valore è basso
	AT+WIOW="GPIO02",0 Imposta basso GPIO02	OK Il valore è scritto

**Si consulti il data-sheet Wavecom "AT Commands Interface Guide for Open AT® Firmware v7.4" per la descrizione dettagliata dei comandi AT.*

quindi il connettore USB standard previsto sulla demoboard è quello di tipo B (CN6 nello schema).
La tensione di alimentazione standard dell'USB è 5 V e viene fornita al pin 1 di CN6; la WCPU montata sul modulo Augusto lavora a 3 V, per cui è stato inserito l'integrato U8 (LP2951) che genera i 3 V necessari per alimentare correttamente la sezione USB della WCPU. Il led DL9 segnala la presenza dei 3 V. Il partitore R82-R87 porta il segnale al pin USB-DET, che serve alla WCPU per rilevare la connessione USB.
Su ogni linea dati differenziale DP e DM sono stati inseriti un filtro (R89, C46 e R91, C47) e dei diodi (D8) di protezione contro le scariche elettrostatiche (ESD). Secondo lo standard USB 2.0 la WCPU WMP50 dispone poi di un pin USB-CN per utilizzare la comunicazione ad alta velocità (fino a 12 Mbit/s). Quando il segnale USB-CN va a livello alto tramite il transistor NPN Q15, si attiva il mosfet di tipo P siglato Q14, che manda in pull-up, tramite la resistenza R92, la linea positiva DP della USB. Questo viene riconosciuto dallo standard USB 2.0 come funzionamento in full-speed.

KEYPAD

In questa sezione dello schema sono presenti 6 tasti disposti a matrice e collegati sugli ingressi dedicati del modulo Augusto ROW0, ROW1, ROW2 e COL0, COL1.
Su ogni tasto è collegato anche un varistore per limitare i transitori di tensione dovuti a scariche elettrostatiche (ESD) che possono avvenire quando si preme il tasto.
La gestione della matrice di tasti è affidata all'algoritmo di controllo interno alla WCPU, che si occupa della scansione della matrice per rilevare qual è il tasto premuto. Non occorre aggiungere alcun condensatore di anti-rimbalzo, poiché la sua funzione è già svolta nella WCPU.
Per leggere dal collegamento seriale il tasto premuto, si possono utilizzare i comandi AT+WHCNF e AT+CMER.

AUDIO

Le WCPU WMP50 e WMP100 dispongono di due canali audio separati, entrambi disponibili sul pin-strip di I/O del modulo Augusto. Ogni canale è comprensivo di un ingresso microfonico e di un'uscita altoparlante. Il canale 2 è solitamente utilizzato per creare un sistema vivavoce con un microfono e uno speaker separati e opportunamente posizionati all'interno di un autoveicolo.
Il canale 1 invece è normalmente collegato ad una cornetta o ad un kit auricolare.
I segnali audio sono disponibili ai connettori di tipo RJ-45 siglati CN1 e CN2.
La scelta del tipo di filtraggio da adottare sui canali audio è molto importante, soprattutto in un'applicazione rumorosa come quella in cui è presente un modulo GSM. In particolare,

la principale fonte di rumore nella banda audio è proprio il GSM stesso: il rumore è chiamato TDMA ed è dovuto alla demodulazione del segnale GSM, che comporta dei burst ogni 4,615 ms a 216,7 Hz più le armoniche.
Per quanto riguarda il microfono, sia per il canale 1 che per il canale 2 è stato scelto un circuito differenziale, in quanto è più immune al rumore rispetto alla soluzione single-ended. Per i due canali microfonici, la WCPU accetta comunque anche quest'ultimo tipo di configurazione, comunque sconsigliata da Wavecom. La principale differenza tra il MIC1 e il MIC2 è data dal fatto che il canale 2 è già polarizzato internamente a 2 V, mentre il microfono collegato al canale 1 necessita di polarizzazione esterna. Per quanto riguarda i canali dedicati agli altoparlanti (speakers SPK1,SPK2) il canale 2 può essere configurato come differenziale o singolo; nella demoboard è stata scelta la soluzione differenziale. Il canale 1, invece, accetta solo la configurazione single-ended,

Bibliografia:

- Datasheet Augusto rev.01 ***www.shitek.eu***
- User Guide Demoboard EVK Augusto ***www.shitek.eu***
- Datasheet Wavecom WMP50
  http://www.wavecom.com->Pruduct->Download
- Application Notes Wavecom
  http://www.wavecom.com->Developers->Application Notes

come si vede, nello schema, dalla presenza del condensatore di disaccoppiamento C3. Le impedenze ammesse per l'altoparlante con le configurazioni scelte sono di 8 ohm per l'SPK2 e di 32 ohm per l'SPK1.
Per le caratteristiche specifiche dell'altoparlante e del microfono si rimanda al data-sheet del modulo Augusto o al data-sheet di Wavecom.

INPUT-OUTPUT

In questa sezione dello schema sono presenti 2 tasti (P1 e P2) collegati alle basi dei transistor, rispettivamente, Q7 e Q10. Alla pressione del tasto i transistor agiscono come interruttori e portano a livello basso le linee di interrupt INT0 e INT1. In questo modo è possibile testare il funzionamento dei due ingressi di interrupt della WCPU.

Questa sezione permette anche di testare 4 degli 11 GPIO disponibili sul modulo Augusto. I pin GPIO0, GPIO1, GPIO2, GPIO3 tramite gli switch DIP1 possono essere collegati alle resistenze di base dei transisor Q8, Q9, Q11, Q12 e pilotare i rispettivi led DL4, DL5, DL6, DL7. Il dip-switch DIP1 serve per collegare (e scollegare) fisicamente i pin di GPIO e di interrupt con (dai) i circuiti della demoboard disegnati in questa sezione dello schema, ciò permette di utilizzare questi pin anche collegandoli a circuiti esterni alla demoboard.
I comandi AT per configurare gli ingressi di interrupt e i GPIO sono riportati nell'apposito riquadro (Comandi AT) in queste pagine.

MONTAGGIO DEL CIRCUITO

Questa demoboard è molto compatta ed utilizza componenti quasi esclusivamente di tipo SMD. Consigliamo di prendere in considerazione il montaggio manuale solo se avete già esperienza e pratica del montaggio manuale di circuiti SMD.
Alcuni componenti SMD hanno passo inferiore al millimetro e le resistenze sono di tipo 0603, per cui consigliamo vivamente di munirvi di un saldatore con punta sottile e di aiutarvi nella saldatura con del flussante; questo per prevenire eventuali sbavature di stagno difficili poi da individuare a occhio nudo.
Se decidete di autocostruirvi la demoboard, raccomandiamo di utilizzare esclusivamente i componenti elencati nella lista, soprattutto per quel che riguarda i condensatori del circuito switching di alimentazione C38, C39, C40, C41, che devono essere di tipo low-esr.
Al solito, montate per primi tutti i componenti SMD a partire dalle resistenze e dai componenti più bassi: diodi, condensatori, integrati; in seguito montate tutti i componenti PTH (connettori, pulsanti, ecc.). Si consiglia inoltre di procedere nel montaggio per gradi, partendo dalle sezioni di alimentazione e verificando le tensioni generate dai vari stadi di alimentazione man mano che si avanza con il lavoro.
Il modulo Augusto va alloggiato solo dopo aver verificato che tutte le tensioni presenti sui connettori strip JP6, JP7, JP8, JP9 rientrano nelle specifiche elencate nel datasheet del modulo Augusto, riportate nella **Tabella 1**.

Corso Wireless CPU

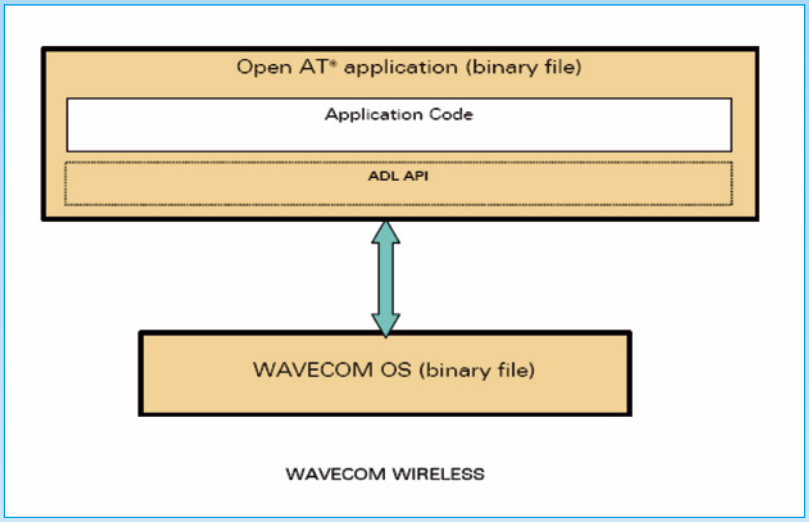
Introduzione alle librerie API di Open AT.

Entriamo nel vivo della programmazione imparando ad usare le librerie ad alto livello ADL, a gestire le funzioni e a dichiarare le variabili, mediante un esempio pratico.

dell'Ing.
NICOLA FRISO

In questa puntata del corso introduciamo le librerie API (Application Programming Interface) per lo sviluppo di applicazione con il sistema operativo Open AT di Wavecom. Per chi si fosse perso le puntate precedenti, ricordiamo che Open AT è un sistema operativo che comprende un set di librerie scritte in C che permettono di interfacciarsi ad alto livello con le periferiche hardware delle Wireless CPU di Wavecom. Le funzioni messe a disposizione dalle librerie permettono altresì di gestire connessioni dati GSM e GPRS, invio e ricezione di SMS e molto altro ancora. L'insieme di librerie ad alto livello che Wavecom mette gratuitamente a disposizione dello sviluppatore prende il nome di ADL (Application Developer Layer); questo "strato" di software

Fig. 1 - *Livelli software Open AT.*



fornisce allo sviluppatore una serie di servizi per l'accesso a tutte le capacità e periferiche delle Wireless CPU. La **Fig. 1** tratta dall' "ADL User Guide for Open AT" di Wavecom, mostra chiaramente come il set di API ADL funzioni da interfaccia verso i servizi messi a disposizione dal sistema operativo sottostante, che gestisce a basso livello la WCPU.
La nostra applicazione sarà dunque formata da codice C che richiama queste librerie ADL. Durante lo sviluppo del codice è fondamentale tener conto del fatto che abbiamo a che fare con un sistema operativo vero e proprio, per cui il flusso logico nello sviluppo del programma non è lineare e non è quello tipico a cui molti sono abituati quando si programma con i microcontrollori; sul nostro codice non troverete una vera e propria funzione main con un ciclo infinito tipo **while(1)** o **for(;;)** al cui interno si richiamano le diverse funzioni. C'è invece un unico punto di ingresso del programma (entry-point) chiamato "adl_main", dove eseguire le eventuali inizializzazioni, il quale viene eseguito una volta sola all'avvio della applicazione.
Tutto il resto del codice è gestito ad eventi ed a "task", i quali, nella terminologia dei sistemi operativi, sono pezzi di codice che svolgono una specifica funzione e che vengono eseguiti come se fossero in parallelo, alternandosi uno all'altro. Alcuni task più prioritari possono interromperne altri meno prioritari, ma per l'utente è come se il tutto fosse eseguito allo stesso momento.
Fortunatamente con le librerie ADL, sebbene sia possibile, non è necessario gestire a basso livello i task e le loro priorità; tutto è così molto più semplice di quanto possa sembrare.
Ogni funzione o periferica che si vuole abilitare (ad esempio Timer, ingressi/uscite, porta seriale, interrupt ecc…) deve essere prima attivata, nel gergo Open AT, si dice che deve essere "sottoscritta"; ecco perché per ogni interfaccia che si vuole abilitare c'è la sua funzione di "subscribe".
Dopo essere stata utilizzata, ogni funzione può essere disattivata, richiamando, allo scopo, la corrispondente funzione di de-sottoscrizione "unsubscribe". Ad ogni sottoscrizione il sistema assegna anche un identificativo ID dell'evento, che è un numero intero assegnato dal sistema nel momento della sottoscrizione e che identifica l'evento quando si scatena.
Nella sottoscrizione dell'evento occorre anche passare come parametro anche il puntatore alla funzione che gestirà l'evento, quando si scatenerà.
In pratica, una volta che si è eseguita la sottoscrizione, quando si verifica l'evento il flusso del programma salta automaticamente dentro la funzione di gestione dell'evento, al cui interno lo sviluppatore avrà inserito il proprio codice di gestione. Una volta entrati in questa filosofia di programmazione, diventa molto semplice e veloce sviluppare le nostre applicazioni.

Un esempio di questo concetto di programmazione lo abbiamo già visto nella seconda puntata del corso, con il programma "Hello World". In questo esempio abbiamo attivato un Timer con la funzione **adl_tmrSubscribe** richiamata all'interno dell'entry-point **adl_main**, che deve essere sempre presente

all'interno delle nostre applicazioni (M2M Studio lo crea automaticamente quando si avvia un nuovo progetto). A riguardo guardate il **Listato 1** che trovate qui accanto.
Il prototipo della funzione **adl_tmrSubscribe** è il seguente:
adl_tmr_t *adl_tmrSubscribe(bool bCyclic, u32 TimerValue, adl_tmrType_e TimerType, adl_tmrHandler_t Timerhdl)
Qui di seguito trovate la descrizione dei parametri della funzione.
bool bCyclic = valore booleano; TRUE corrisponde ad un timer ciclico, mentre FALSE attiva un timer di tipo "one shot" in cui allo scadere del tempo impostato il timer si ferma e termina automaticamente.
u32 TimerValue = valore temporale impostato (in multipli di 100 ms oppure in unità di "tick").
adl_tmrType_e TimerType = tipo di timer, i valori ammessi sono due:

- **ADL_TMR_TYPE_100MS** ; base del tempo del timer pari a 100ms;
- **ADL_TMR_TYPE_TICK**; base del tempo del timer pari a un tick (18,5 ms).

Notate che il tick permette di avere temporizzazioni più precise.
Esistono anche delle funzioni di conversione da ms a tick e viceversa (vedere, sulla "ADL UserGuide for Open AT" di Wavecom, le macrofunzioni) che sono, rispettivamente:
ADL_TMR_MS_TO_TICK(MsT),ADL_TMR_100MS_TO_TICK(MsT)
Per la conversione da millisecondi a tick e: **ADL_TMR_S_TO_TICK(SecT),ADL_TMR_MN_TO_TICK(MnT))**
Per la conversione da tick a millisecondi.
L'ultimo parametro della funzione da analizzare è **adl_tmrHandler_t Timerhdl**, che è il puntatore alla funzione che gestisce il timer.
Quindi alla funzione **adl_tmrSubscribe** viene passato il primo parametro TRUE per impostare un timer ciclico che si attiva ogni 10x100 ms = 10s e che ad ogni attivazione "salta" alla funzione HelloWorld_TimerHandler.
Vediamo ora un esempio un po' più complesso: salviamo il nostro progetto "Hello_World" con un nuovo nome "Allarme" e andiamo a realizzare un'applicazione che alla pressione di un tasto genera un allarme. Utilizziamo un pin della tastiera "keypad" per rilevare la

Listato 1 - Codice dell'applicazione **"Hello World".**

```
/**************************************************************************/
/*  File       : Hello_World.c                                            */
/*------------------------------------------------------------------------*/
/*  Object     : Customer application                                     */
/*                                                                        */
/*  contents   : Customer main procedures                                 */
/*                                                                        */
/*  Change     :                                                          */
/**************************************************************************/
/*

#include "adl_global.h"


/**************************************************************************/
/*  Mandatory variables                                                   */
/*------------------------------------------------------------------------*/
/*  wm_apmCustomStackSize                                                 */
/*------------------------------------------------------------------------*/
/**************************************************************************/
const u16 wm_apmCustomStackSize = 1024*3;



/**************************************************************************/
/*  Function   : HelloWorld_TimerHandler                                  */
/*------------------------------------------------------------------------*/
/*  Object     : Timer handler                                            */
/*                                                                        */
/*------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                         */
/*--------------------+---+---+---+-------------------------------------*/
/*  ID                |   |   |   | Timer ID                            */
/*--------------------+---+---+---+-------------------------------------*/
/*  Context           |   |   |   | Context                             */
/*--------------------+---+---+---+-------------------------------------*/
/**************************************************************************/
void HelloWorld_TimerHandler ( u8 ID, void * Context )
{
    /* Hello World */
    TRACE (( 1, "Embedded : Hello World" ));
    adl_atSendResponse ( ADL_AT_UNS, "\r\nHello World from Open-AT\r\n" );
}



/**************************************************************************/
/*  Function   : adl_main                                                 */
/*------------------------------------------------------------------------*/
/*  Object     : Customer application initialisation                      */
/*                                                                        */
/*------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                         */
/*--------------------+---+---+---+-------------------------------------*/
/*  InitType          |   |   |   |  Application start mode reason      */
/*--------------------+---+---+---+-------------------------------------*/
/**************************************************************************/
void adl_main ( adl_InitType_e  InitType )
{
    TRACE (( 1, "Embedded : Appli Init" ));

    /* Set 1s cyclic timer */
    adl_tmrSubscribe ( TRUE, 10, ADL_TMR_TYPE_100MS, HelloWorld_TimerHandler );
}
```

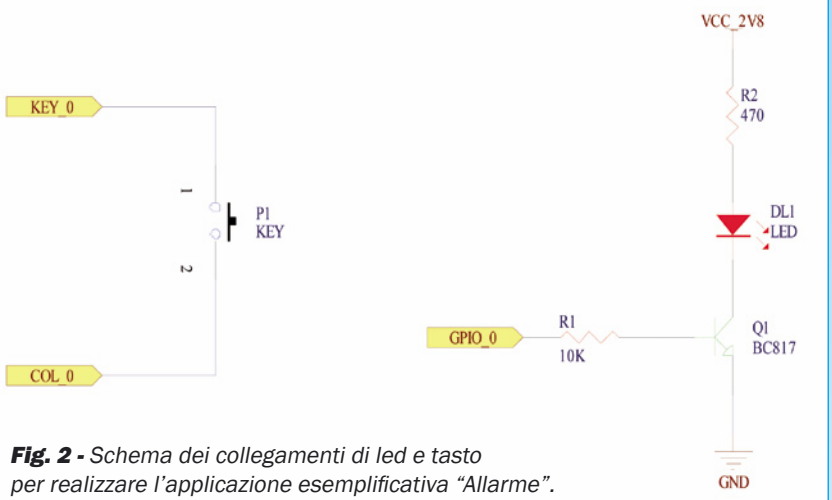


Fig. 2 *- Schema dei collegamenti di led e tasto per realizzare l'applicazione esemplificativa "Allarme".*

pressione del tasto e un pin di GPIO, configurato come uscita, per pilotare il led, il quale deve lampeggiare per 10 secondi e poi smettere; inoltre alla pressione del tasto generiamo un SMS di allarme verso un numero preimpostato.
Tutto questo ci permetterà di analizzare nel dettaglio, oltre ai timer anche le funzioni che gestiscono la tastiera, i pin di GPIO e quelle relative all'invio degli SMS.
Per realizzare questo esperimento possiamo utilizzare la demoboard EVKAUGUSTO per il modulo Augusto di Shitek Technology; in alternativa, se non disponiamo dell'EVKAUGUSTO possiamo collegare il modulo Augusto ai pin GPIO0, KEY0,COL0 secondo lo schema riportato nella **Fig. 2**.
Analizziamo riga per riga il codice dell'applicazione: nella prima parte troviamo l'include "adl_global.h" che deve essere sempre presente per caricare tutte le definizioni, le costanti predefinite e i prototipi della libreria ADL. A seguire troviamo una variabile di sistema che alloca la memoria per lo stack delle chiamate a funzione. Da questo punto in avanti inseriamo il codice della nostra applicazione "Allarme".
Definiamo prima di tutto quanti pin di GPIO intendiamo usare come uscita; in questo caso si tratta di un solo pin: GPIO0. Per fare ciò dichiariamo il tipo di struttura predefinita adl_ioDefs_t, tramite la quale è possibile definire la direzione di tutti i pin di GPIO e il loro stato iniziale.
In questo caso abbiamo definito un solo pin di uscita GPIO0 (ADL_IO_GPIO | 0); la direzione di uscita è fissata dalla costante predefinita ADL_IO_DIR_OUT e lo stato iniziale è basso (ADL_IO_LEV_LOW). Così facendo, nella struttura Gpio_Config di tipo adl_ioDefs_t abbiamo inserito una maschera di 16 bit, mettendo in OR le varie costanti predefinite; la maschera, in questo caso, dice al sistema operativo che al momento della sottoscrizione dei GPIO deve settare il GPIO0 come uscita a livello basso.
Utilizzando la stessa struttura potevamo definire come uscite o ingressi anche altri pin: se, ad esempio, volevamo definire anche il pin GPIO1 come uscita a livello alto e il GPIO2 come ingresso, bastava aggiungere alla struttura le maschere di bit relative a questi pin, ad esempio:

```
#define NUM_GPIO 3                    // number of GPIOs sets

static adl_ioDefs_t Gpio_Config[NUM_GPIO] =
{

        {ADL_IO_GPIO | 0 | ADL_IO_DIR_OUT | ADL_IO_LEV_LOW},  // set GPIO0 as output low
        {ADL_IO_GPIO | 1 | ADL_IO_DIR_OUT | ADL_IO_LEV_HIGH}, // set GPIO1 as output high
        {ADL_IO_GPIO | 2 | ADL_IO_DIR_IN}                     // set GPIO2 as input
};
```

La riga successiva, ossia "s32 myGpioOut_Handle" dichiara un intero a 32 bit con segno (vedere il riquadro **Tipi di variabili**) che sarà il numero identificativo (detto "handle") il quale servirà alle funzioni che gestiranno questa uscita. In modo analogo, nelle due righe successive definiamo un handle associato agli SMS ed uno associato al timer che farà lampeggiare il led.
"tim_count" è una variabile globale di tipo intero a 8 bit senza segno, che utilizzeremo come contatore di numero di volte che viene richiamata la funzione che gestisce il timer.

Le ultime due righe della parte di dichiarazione delle variabili definiscono due stringhe ASCII contenenti il numero di telefono cui inviare l'SMS di allarme e il testo dell'SMS.
Stabilite quali sono le variabili e strutture globali utilizzate, andiamo a vedere il contenuto della funzione di ingresso **adl_main**: qui troviamo l'inizializzazione della variabile contatore tim_count e, a seguire, le chiamate alle funzioni utili per sottoscrivere e attivare il pin di uscita GPIO0, per attivare la tastiera e per inizializzare la SIM del modulo. Vediamole nel dettaglio.
La funzione **adl_ioSubscribe** restituisce un intero maggiore-uguale a zero se la sottoscrizione dei pin di GPIO passati come parametro riesce. Se ciò non avviene, può restituire vari tipi di codice di errore sul cui significato per ora possiamo soprassedere, rimandando alla guida ADL di Wavecom. I parametri passati sono il numero di pin di GPIO da configurare, la struttura di definizione dei pin che abbiamo visto sopra, più altri tre parametri, impostati a zero, che servono qualora si fossero settati dei GPIO come ingressi e si volesse attivare un polling automatico per interrogarne lo stato. Vedremo più avanti in un capitolo del corso dedicato agli I/O anche questa comoda funzione. Per ora lasciamo questi parametri a zero.
La riga successiva è una chiamata alla funzione **adl_atCmdCreate** che serve per richiamare dall'interno della nostra applicazione qualsiasi comando AT. In questo caso abbiamo inoltrato il comando "AT+CMER=,1", il quale abilita la generazione di un evento ogni volta che si preme un tasto sulla tastiera. L'evento viene segnalato dalla generazione sulla seriale di una stringa del tipo "+CKEV:

Listato 2 - Codice dell'applicazione **"Allarme"**.

```
/*****************************************************************************/
/*  File       : Allarme.c                                                   */
/*---------------------------------------------------------------------------*/
/*  Object     : Customer application                                        */
/*                                                                           */
/*  contents   : Customer main procedures                                    */
/*                                                                           */
/*  Change     :                                                             */
/*****************************************************************************/
/*
 * ----------------------------------------------------------------------------
 *  Date     | Author | Revision       | Description
 * ----------+--------+----------------+-------------------------------------
 *  29.09.09 | NicolaF| 0.1            | * Tested version
 * ----------+--------+----------------+-------------------------------------
*/

#include "adl_global.h"


/*****************************************************************************/
/*  Mandatory variables                                                      */
/*---------------------------------------------------------------------------*/
/*  wm_apmCustomStackSize                                                    */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
const u16 wm_apmCustomStackSize = 1024*3*3;

#define NUM_GPIO 1              // number of GPIOs sets

static adl_ioDefs_t Gpio_Config[NUM_GPIO] = {

          {ADL_IO_GPIO | 0 | ADL_IO_DIR_OUT | ADL_IO_LEV_LOW}      // set GPIO0 as output low
};

static s32 myGpioOut_Handle;              // handle to GPIO0 out

static s8 sms_handle;                     // handle for sms event

static adl_tmr_t* tim_handle;             // timer handle for led

static u8 tim_count;                      // timer tim_counter

static ascii phone_number[] = "+393289531658";         // phone number to send sms
static ascii sms_text[] = "Allarme";                              // text of sms




/*****************************************************************************/
/*  Function   : Led_TimerHandler                                            */
/*---------------------------------------------------------------------------*/
/*  Object     : Timer handler                                               */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                            */
/*--------------------+---+---+---+----------------------------------------*/
/*  ID                |   |   |   | Timer ID                               */
/*--------------------+---+---+---+----------------------------------------*/
/*  Context           |   |   |   | Context                                */
/*--------------------+---+---+---+----------------------------------------*/
/*****************************************************************************/
void Led_TimerHandler ( u8 ID, void * Context )
{
   adl_ioDefs_t Gpio_to_write = ADL_IO_GPIO | 0 ;

   tim_count++;

          // Read output and toggle led
          if(adl_ioReadSingle( myGpioOut_Handle,&Gpio_to_write))
                    // Reset output
                    adl_ioWriteSingle ( myGpioOut_Handle, &Gpio_to_write, FALSE );
          else
                    // Set output
                    adl_ioWriteSingle ( myGpioOut_Handle, &Gpio_to_write , TRUE );

          if(tim_count > 20)    // Stop timer after 10s
             {
                       // Reset output
                       adl_ioWriteSingle ( myGpioOut_Handle, &Gpio_to_write , FALSE );
                       // init timer counter
                       tim_count = 0;
                       // Stop timer led
                       adl_tmrUnSubscribe ( tim_handle, Led_TimerHandler, ADL_TMR_TYPE_100MS);
             }

}


/*****************************************************************************/
/*  Function   : CKEV_Handler
 */
/*---------------------------------------------------------------------------*/
/*  Object     : CKEV_Handler                                                */
/*                                                                           */
/*---------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                            */
/*--------------------+---+---+---+----------------------------------------*/
/*  gpio_handle       |   |   |   |                                        */
```

Listato 2 ... continuazione

```
/*-------------------+---+---+---+---------------------------------------*/
/*  gpio_result      |   |   |   |
*/
/*-------------------+---+---+---+---------------------------------------*/
/*************************************************************************/
bool CKEV_Handler(adl_atUnsolicited_t *paras)
{
            ascii key_nb[3];        // key number
            ascii key_st[3];        // key status (ON=1 OFF=0)

            u8 kn,ks;

            TRACE (( 1, "CKEV" ));


            wm_memset(&key_nb,0x00,sizeof(key_nb));
            wm_memset(&key_st,0x00,sizeof(key_st));


            wm_strGetParameterString ( key_nb, paras->StrData, 1 );
            kn = wm_atoi(key_nb); // key number
            TRACE (( 1, "KEY_NB=%d",kn));

            wm_strGetParameterString ( key_st, paras->StrData, 2 );
            ks = wm_atoi(key_st); // key status
            TRACE (( 1, "KEY_ST=%d",ks));

            if(!tim_count)          // active alarm only if previus alarm is terminated
            {
                    //if(!wm_strcmp(res,"0"))
                    if(kn==0 && ks==0) // key number 0 release
                    {
                        // Send sms
                        TRACE (( 1, "Alarm activated...send sms" ));
                        adl_smsSend (sms_handle, phone_number, sms_text, ADL_SMS_MODE_TEXT);

                        // Set 500ms cyclic  led timer
                        TRACE (( 1, "Start led timer" ));
                        tim_handle = adl_tmrSubscribe ( TRUE, 5, ADL_TMR_TYPE_100MS, Led_TimerHandler );
                    }
            }

            return TRUE;
}
/*************************************************************************/
/*  Function    : SMS_Control_f
                                                                         */
/*-------------------------------------------------------------------------*/
/*  Object      : SMS_Control_f
                                                                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/
/*  Variable Name      |IN |OUT|GLB|  Utilisation                        */
/*-------------------+---+---+---+---------------------------------------*/
/*  Event                          |   |   |   |                         */
/*-------------------+---+---+---+---------------------------------------*/
/*  Nb                               |   |   |   |
        */
/*-------------------+---+---+---+---------------------------------------*/
/*************************************************************************/
void SMS_Control_f(u8 Event,u16 Nb)
{
            if(Event == ADL_SMS_EVENT_SENDING_OK)
            {
                        TRACE((1,"SMS SEND OK"));
                        adl_atSendResponse (ADL_AT_RSP, "\r\nSMS SEND OK\r\n");
            }
            if(Event == ADL_SMS_EVENT_SENDING_ERROR)        // can be here if no credit
            {
                        TRACE((1,"SMS SENDING ERROR"));
                        adl_atSendResponse (ADL_AT_RSP, "\r\nSMS SENDING ERROR\r\n");
            }
}
/*************************************************************************/
/*  Function    : SMS_Handler_f
                                                                         */
/*-------------------------------------------------------------------------*/
/*  Object      : SMS_Handler_f
                                                                         */
/*                                                                         */
/*-------------------------------------------------------------------------*/
/*  Variable Name      |IN |OUT|GLB|  Utilisation                        */
/*-------------------+---+---+---+---------------------------------------*/
/*  SmsTel                         |   | X |   |                         */
/*-------------------+---+---+---+---------------------------------------*/
/*  SmsTimeOrLength    |   | X |   |
*/
/*-------------------+---+---+---+---------------------------------------*/
/*          SmsText                          |          | X |   |

*/
/*-------------------+---+---+---+---------------------------------------*/
/*************************************************************************/
bool SMS_Handler_f(ascii *SmsTel, ascii *SmsTimeOrLength, ascii *SmsText)
{
            TRACE((1,"SMS ricevuto:"));
            TRACE((1,SmsText));
            TRACE((1,SmsTel));
            return TRUE;
}

/*************************************************************************/
```

<key>,<press>" in cui <key> è il codice del tasto premuto e <press> è lo stato del tasto (1 premuto, 0 rilasciato). Occorre quindi intercettare questa stringa per capire quando il tasto viene premuto. La chiamata:

adl_atUnSoSubscribe("+CKEV:",(adl_atUnSoHandler_t)CKEV_Handler);

fa proprio questo. Per l'esattezza, sottoscrive un evento di tipo "Unsolicited" che riguarda la tastiera: ogni volta che viene generata una stringa "+CKEV:" il flusso del programma salta alla funzione (gestita dall'utente) "CKEV_Handler" che deve andare a leggere i successivi due parametri <key> e <press> per capire quale tasto è stato premuto. Normalmente tutti gli eventi "Unsolicited" (ossia indesiderati...) per impostazione predefinita, sono disattivati. L'ultima riga del main sottoscrive i servizi relativi alla SIM che comprendono l'invio e la ricezione degli SMS, ma anche la gestione del PIN; in questo caso non abbiamo volutamente gestito il PIN per non complicare il codice. Il programma quindi funziona solo se si disabilita il PIN della SIM prima di inserire quest'ultima nel modulo Augusto.

La funzione **appSimHandler** restituisce delle informazioni circa lo stato della SIM.

Se è inizializzata e pronta all'uso, restituisce ADL_SIM_STATE_FULL_INIT, mentre se è in attesa di PIN ritorna ADL_SIM_STATE_PIN_WAIT; se il pin è errato, la funzione restituisce ADL_SIM_STATE_PIN_ERROR, e infine, se la SIM è rimossa o inserita, restituisce rispettivamente ADL_SIM_EVENT_REMOVED e ADL_SIM_EVENT_INSERTED.

Quando la SIM è inizializzata, sottoscriviamo il servizio SMS in modo da essere in grado in seguito di ricevere e trasmettere SMS.

La riga di codice seguente:

sms_handle = adl_smsSubscribe(SMS_Handler_f,SMS_Control_f,ADL_SMS_MODE_TEXT);

sottoscrive gli SMS. Durante la sottoscrizione vengono passati tre parametri, che sono:

SMS_Handler_f = puntatore ad una

funzione gestita dall'utente in cui l'applicazione salta quando riceve un SMS;
SMS_Control_f = puntatore ad una funzione gestita dall'utente in cui l'applicazione salta quando ha inviato un SMS;
ADL_SMS_MODE_TEXT = indica il modo in cui gli sms devono essere ricevuti o inviati.
In quest'ultimo caso nel parametro è stato impostato il modo testo; è anche possibile scegliere il modo ADL_SMS_MODE_PDU, ricordando che PDU è un protocollo binario nativo del sistema GSM ed è con questo protocollo che gli SMS vengono scambiati nella rete. Per semplicità utilizzeremo sempre il modo testo. Tornando al flusso del programma, dopo aver eseguito il main, la SIM si è inizializzata e l'applicazione è in attesa di un tasto premuto. Alla pressione del tasto si esegue la funzione "CKEV_Handler".
In questa funzione si analizzano i parametri <key> e <press> con la funzione di libreria wm_strGetParameterString che copia il primo (<key>) e secondo (<press>) parametro nell'array "key_nb" e "key_st" rispettivamente. Il primo parametro è il numero del tasto premuto mentre il secondo è lo stato zero o uno, del tasto. Prima di generare l'allarme verifichiamo lo stato del timer di allarme. Se il contatore è uguale a zero, cioè non ci sono altri allarmi precedenti in corso si genera l'sms di allarme e si fa partire il timer per il lampeggio del led. L'allarme viene generato quando lo stato "key_st" del tasto è zero, che significa far partire l'allarme quando il tasto viene rilasciato. Quindi si genera l'SMS di allarme e si fa partire il timer per il lampeggio del led. La funzione **adl_smsSend** invia l'SMS; quando il messaggio è stato realmente inoltrato alla rete GSM, il flusso del programma salta alla funzione SMS_Control_f che avevamo definito in fase di sottoscrizione degli SMS. Se tutto è andato a buon fine si verifica l'evento ADL_SMS_EVENT_SENDING_OK e il programma genera sulla porta seriale la stringa "SMS SEND OK".

Listato 2 ... continuazione

```
/*  Function    : appSimHandler                                                 */
/*--------------------------------------------------------------------------*/
/*  Object      : Handler to receive SIM related events                     */
/*                                                                          */
/*  Return      :                                                           */
/*                                                                          */
/*                                                                          */
/*--------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                          */
/*--------------------+---+---+---+---------------------------------------*/
/*                     |   |   |   |                                       */
/*--------------------+---+---+---+---------------------------------------*/
/****************************************************************************/
void appSimHandler (u8 Event)
{
  TRACE((1,"APP SIM EVENT HANDLER"));
  switch (Event)
  {
    case ADL_SIM_STATE_FULL_INIT:

              TRACE (( 1, "ADL_SIM_STATE_FULL_INIT"));

              // SMS subscribe
                sms_handle  =  adl_smsSubscribe(SMS_Handler_f,SMS_Control_f,ADL_SMS_MODE_
TEXT);
              TRACE((1,"sms_handle= %d",sms_handle));
      break;

    case ADL_SIM_STATE_PIN_WAIT:
              TRACE((1,"\r\nADL_SIM_STATE_PIN_WAIT\r\n"));
      adl_atSendResponse (ADL_AT_RSP, "\r\nADL_SIM_STATE_PIN_WAIT\r\n");
      break;

    case ADL_SIM_STATE_PIN_ERROR:
              TRACE((1, "\r\nADL_SIM_STATE_PIN_ERROR\r\n"));
      adl_atSendResponse (ADL_AT_RSP, "\r\nADL_SIM_STATE_PIN_ERROR\r\n");
      break;

    default:
      TRACE (( 1, "SIM Event = %d", Event ));
      break;
  }
  return;
}
/****************************************************************************/
/*  Function    : adl_at_CMER_response_handler
                                  */
/*--------------------------------------------------------------------------*/
/*  Object      : adl_at_CMER_response_handler                              */
/*                                                                          */
/*--------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                          */
/*--------------------+---+---+---+---------------------------------------*/
/*  gpio_handle       |   |   |   |                                       */
/*--------------------+---+---+---+---------------------------------------*/
/*  gpio_result       |   |   |   |                                        */
/*--------------------+---+---+---+---------------------------------------*/
/****************************************************************************/
void adl_at_CMER_response_handler(adl_atResponse_t *at_str_pars)
{
           // nothing to do
            TRACE (( 1, "CMER" ));

}
/****************************************************************************/
/*  Function    : adl_main                                                  */
/*--------------------------------------------------------------------------*/
/*  Object      : Customer application initialisation                       */
/*                                                                          */
/*--------------------------------------------------------------------------*/
/*  Variable Name     |IN |OUT|GLB|  Utilisation                          */
/*--------------------+---+---+---+---------------------------------------*/
/*  InitType          |   |   |   |  Application start mode reason        */
/*--------------------+---+---+---+---------------------------------------*/
/****************************************************************************/
void adl_main ( adl_InitType_e  InitType )
{
           s8 res;

           //adl_ioDefs_t Gpio_to_write1 = ADL_IO_GPIO | 0 ;

           TRACE (( 1, "Embedded : Appli Init" ));

           tim_count = 0;        // init timer counter

    // Subscribe GPIO0 output
          myGpioOut_Handle = adl_ioSubscribe(NUM_GPIO,Gpio_Config,0,0,0);
          TRACE (( 1, "handler returns %d", myGpioOut_Handle ));

    // Subscribe keypad
           adl_atCmdCreate("AT+CMER=,1",FALSE,(adl_atRspHandler_t)adl_at_CMER_response_
handler,"*",NULL);

    // Subscribe the '+CKEV: unsolicited response
    adl_atUnSoSubscribe("+CKEV: ",(adl_atUnSoHandler_t)CKEV_Handler);

    // Subscribe to SIM service to send and receive sms
    res = adl_simSubscribe ( (adl_simHdlr_f) appSimHandler, NULL);
}
```

Tipi di **variabili**

I tipi base sono definiti nel file di sistema **wm_types.h**

u8	8 bit senza segno (corrisponde al tipo "unsigned char" del C)
s8	8 bit con segno (corrisponde al tipo "signed char" del C)
u16	16 bit senza segno(corrisponde al tipo "unsigned short" del C)
s16	16 bit con segno (corrisponde al tipo "signed short" del C)
u32	32 bit senza segno (corrisponde al tipo "unsigned int" del C)
s32	32 bit con segno (corrisponde al tipo "signed int" del C)
u64	64 bit senza segno (corrisponde al tipo "unsigned long" del C)
s64	64 bit con segno (corrisponde al tipo "signed long" del C)
bool	booleano true o false
ascii	carattere 8 bit (corrisponde al tipo "char" del C)

Nel frattempo il timer per il lampeggio del led gestito dalla funzione **Led_TimerHandler** è partito e, ad ogni richiamo della funzione, commuta lo stato del pin di uscita GPIO_0. Per farlo commutare prima, leggiamo lo stato del pin con la funzione **adl_ioReadSingle**; se è ad uno lo mettiamo a zero mediante la funzione **adl_ioWriteSingle**, viceversa se lo stato era basso. Si noti che alle funzioni di Read e Write dobbiamo passare l'handle "myGpioOut_Handle" che avevamo ottenuto sul main (in fase di sottoscrizione del pin di GPIO) e il puntatore alla struttura "adl_ioDefs_t Gpio_to_write", cui abbiamo assegnato il valore "ADL_IO_GPIO | 0", che corrisponde al GPIO_0. Il timer incrementa il contatore tim_count ad ogni chiamata (ogni 500 ms); quando il contatore arriva a 20 (pari a 10 s) il timer viene disattivato con la funzione **adl_tmrUnSubscribe** e il pin di GPIO_0 viene portato a zero logico. Si noti come anche per la funzione di de-sottoscrizione del timer **adl_tmrUnSubscribe** abbiamo dovuto passare ad essa l'handle ricavato in fase di sottoscrizione. Bene, con questo abbiamo finito; ora non vi resta che compilare il progetto e attivare il debug sull'M2M Studio (vedere la seconda puntata del corso) per testare il codice. Nel nostro sito *www.elettronicain.it* trovate il file "allarme.zip" con il progetto per M2M Studio e il codice sorgente dell'applicazione.

Corso Wireless CPU

I comandi AT

In questa puntata conosciamo ed impariamo ad usare i comandi AT speciali delle wireless CPU di Wavecom, utili per realizzare svariate funzioni.

dell'Ing.
NICOLA FRISO

Nelle puntate precedenti abbiamo preso familiarità con le wireless CPU serie WMP di Wavecom, abbiamo visto come configurare l'ambiente di sviluppo M2M Studio per la programmazione delle WCPU, ed abbiamo anche scritto la nostra prima applicazione utilizzando le librerie API "Open-AT" di Wavecom. L'applicazione mandava un SMS di allarme e faceva lampeggiare un led in corrispondenza della pressione di un tasto; l'abbiamo potuta testare fisicamente grazie al modulo "Augusto" e alla demoboard EVKAUGUSTO di Shitek Technology, descritti nelle scorse puntate. Prima di addentrarci ulteriormente nello studio delle librerie API di Open AT è necessario conoscere almeno le basi del funzionamento dei comandi AT.

I comandi AT rappresentano il protocollo di comunicazione standard dei moduli GSM, ma anche di qualsiasi altro modem. Esistono dei comandi AT standard comuni a tutti i modem GSM (secondo la specifica ETSI GSM) ma anche dei comandi AT non standard "aggiuntivi" introdotti dal produttore del modulo GSM; in questo caso Wavecom. La grande novità introdotta da Open AT (da qui l'origine del nome) risiede nella possibilità, offerta al programmatore, di definire e creare nuovi comandi AT personalizzati; vedremo come fare nel corso di questa puntata. Nel seguito di questo articolo sono riportati i principali comandi AT di uso più comune. I comandi AT in generale hanno tutti una formattazione standard che è riportata nella

***Tabella 1** - Formato dei comandi AT.*

Comando	Effetto
AT+<cmd>=?	Visualizza i parametri ammessi per quel comando
AT+<cmd>?	Visualizza le impostazioni attuali per quel comando
AT+<cmd>	Esegue il comando AT esteso senza parametri
AT+<cmd>=<parameters>	Esegue il comando AT esteso con parametri
AT<cmd>	Esegue il comando AT senza parametri

Tabella 1. In tutti i comandi **<cmd>** è l'identificativo del comando e **<parameters>** la lista dei parametri previsti (separati dalla virgola se è presente più di un parametro).

Listato 1 - Richiamo del comando AT+CREG all'interno di un'applicazione.

```
void adl_main( adl_InitType_e InitType )
{
adl_atCmdCreate("AT+CREG?",FALSE,(adl_atRspHandler_t)adl_at_
CREG_response_handler,"*",NULL);
}
void   adl_at_CREG_response_handler(adl_atResponse_t   *at_str_
pars)
{
        ascii res[]={0,0};
        if(at_str_pars->RspID == ADL_STR_OK)
                return;
        if(at_str_pars->RspID == ADL_STR_ERROR ||
           at_str_pars->RspID == ADL_STR_CME_ERROR ||
           at_str_pars->RspID == ADL_STR_CMS_ERROR)
        {
                TRACE((1,"RILEVATO       ERRORE=%d",at_str_pars-
>RspID));
                return;
        }
        wm_strGetParameterString ( res, at_str_pars->StrData, 2
);      // legge parametri di registrazione
        if(res[0]=='1')
        {
                TRACE((1,"Registrato presso operatore locale"));
        }
        else if (res[0]=='5')
        {
                TRACE((1,"Registrato in roaming verso altro ope-
ratore"));
        }
        else
        {
                TRACE((1,"Non ancora registrato. "));
        }
}
```

Ovviamente i comandi AT sono molti di più di quelli riportati in questa puntata del corso ma elencarli tutti esula dagli scopi di questo articolo; chi volesse approfondire l'argomento potrà trovare tutti i comandi standard, ad esempio, nel manuale di cui al punto [1] della Bibliografia. Per poterli provare è sufficiente collegare la porta seriale del PC con la UART1 del modulo Augusto che è presente sulla demoboard EVK AUGUSTO. In alternativa è necessario costruirsi un'interfaccia RS232 come quella riportata nel data-sheet del modulo Augusto prodotto dalla Shitek Technology (*www.shitek.eu*). Nell'apposita sezione di questa puntata sono riportati anche alcuni comandi AT introdotti da Wavecom, che avevamo già descritto nella prima puntata del corso, comandi che sono utili per il download e l'avvio della propria applicazione Open AT all'interno del modulo.
Vi invitiamo a prendere familiarità con i comandi AT in quanto molto spesso occorrerà richiamarli all'interno delle nostre applicazioni Open AT. Le librerie ADL introdotte nella scorsa puntata infatti, permettono di fare molte cose ma non tutto, e spesso è necessario utilizzare proprio

i comandi AT per eseguire certe particolari funzioni (si veda ad esempio la gestione della tastiera presentata nella scorsa puntata). Facciamo presente, inoltre, che le librerie Open AT rappresentano un layer software sottostante ai comandi AT, il che vuol dire che quando inviamo un comando AT, questo non fa altro che richiamare delle funzioni di libreria ADL oppure altre funzioni ancora a più basso livello. A tal proposito segnaliamo che è disponibile un'applicazione Open AT "Internet plug-in" scritta da Wavecom, dove sono stati definiti dei nuovi comandi AT che permettono di gestire lo stack TCP/IP ad alto livello. Con questi specifici comandi è possibile inviare/ricevere mail, creare una connessione FTP, HTTP,UDP e molto altro. Esempi d'uso di questo plug-in si possono trovare nel manuale "AT command user guide for Wavecom IP v4.01" scaricabile dal sito Wavecom. È sufficiente scaricare dal sito l'applicazione **gcc_WIPSoft_256KB.wpb.dwl**, fare il download con il comando AT+WDWL oppure con il software DWLWIN, e avviarla con il comando AT+WOPEN=1, per avere a disposizione questa serie aggiuntiva di comandi AT.

Listato 2 - Richiamo del comando AT+WIND e rilevazione dell'evento WIND 4.

```
void adl_main( adl_InitType_e InitType )
{
// Send  AT+WIND=255
adl_atCmdCreate("AT+WIND=255",FALSE,(adl_atUnSoHandler_t)adl_at_
WIND255_Handler,ADL_STR_OK,NULL);
// Subscribe WIND 4
adl_atUnSoSubscribe("+WIND: 4",(adl_atUnSoHandler_t)Wind4_Handler);
}
bool adl_at_WIND255_Handler(adl_atUnsolicited_t *paras)
{
        TRACE((1,"WIND 255 OK"));
        return TRUE;
}
bool Wind4_Handler(adl_atUnsolicited_t *paras)
{
        adl_atUnSoUnSubscribe("+WIND: 4",(adl_atUnSoHandler_t)Wind4_
Handler);
        TRACE((1,"WIND 4 OK"));
          return TRUE;
}
```

Nella scorsa puntata abbiamo già visto come richiamare un comando AT all'interno della nostra applicazione; ora ci limitiamo pertanto a fornire i passi essenziali. Con la funzione **adl_cmdCreate** si lancia il comando AT. Il prototipo della funzione è il seguente:

*s8 adl_atCmdCreate (ascii * atstr, u16 rspflag, adl_atRspHandler_t rsphdl, …)*

I parametri corrispondenti hanno il significato descritto qui di seguito:
atstr = si tratta della stringa contenente il comando AT da inviare;
rspflag = mettendo FALSE a questo flag si lascia come porta per l'invio del comando AT la porta di default (Open AT virtual port); una impostazione più avanzata, che esula dagli scopi di questo articolo, permette di impostare una porta diversa tramite la macro *ADL_AT_PORT_TYPE(_port,_flag)* (per ulteriori dettagli si consulti l'"ADL User Guide");

Tabella 2 - *Opzioni della funzione adl_atCmdSubscribe.*

Tipo	Valore	Descrizione
ADL_CMD_TYPE_PARA	0x0100	Ammessi comandi AT del tipo: AT+cmd=x,y Il numero di parametri ammessi deve essere messo in OR con questa maschera: ADL_CMD_TYPE_PARA | 0xXY dove X è il numero massimo di parametri ammessi e Y il numero minimo di parametri ammessi.
ADL_CMD_TYPE_TEST	0x0200	Ammessi comandi AT del tipo: AT+cmd=?
ADL_CMD_TYPE_READ	0x0400	Ammessi comandi AT del tipo: 'AT+cmd?'
ADL_CMD_TYPE_ACT	0x0800	Ammessi comandi AT del tipo: 'AT+cmd'
ADL_CMD_TYPE_ROOT	0x1000	Sono ammessi tutti i comandi che iniziano con la stringa definita ma senza il carattere ; finale che serve per il parsing di coamndi AT concatenati. Quindi se la stringa è "AT-" tutti i comandi AT del tipo "AT-cmd1", "AT-cmd2" sono ammessi.

Listato 3 – Applicazione AT+LED.

```
#include "adl_global.h"
const u16 wm_apmCustomStackSize = 1024;
#define NUM_GPIO 1                // number of GPIOs sets
static adl_ioDefs_t Gpio_Config[NUM_GPIO] = {
        {ADL_IO_GPIO | 0 | ADL_IO_DIR_OUT | ADL_IO_LEV_LOW}
// set GPIO0 as output low
};
static s32 myGpioOut_Handle;              // handle to GPIO0 out
static adl_tmr_t* tim_handle;             // timer handle for led
static u8 timer_led;                      // timer led 0 to 1000ms
/***********************************/
/*  Function   : Led_TimerHandler */
/***********************************/
void Led_TimerHandler ( u8 ID, void * Context )
{
   adl_ioDefs_t Gpio_to_write = ADL_IO_GPIO | 0 ;

   TRACE (( 1, "Led timer" ));

   // Read output and toggle led
   if(adl_ioReadSingle( myGpioOut_Handle,&Gpio_to_write))
      // Reset output
        adl_ioWriteSingle  (  myGpioOut_Handle,  &Gpio_to_write,
FALSE );
   else
      // Set output
        adl_ioWriteSingle  (  myGpioOut_Handle,  &Gpio_to_write  ,
TRUE );
}
/***********************************/
/*  Function   : cmdATLED_Handler */
/***********************************/
void cmdATLED_Handler (adl_atCmdPreParser_t *param)
{
  u16 time_ms;
  timer_led = wm_atoi(ADL_GET_PARAM(param,0));
  TRACE (( 1, "AT+LED ricevuto: Stop led timer" ));
   adl_tmrUnSubscribe(  tim_handle,Led_TimerHandler,ADL_TMR_TYPE_
100MS);          // Stop timer led

  if(timer_led == 0 || timer_led > 255)
        {
        TRACE (( 1, "Stop led timer" ));
        return;
        }
  time_ms = timer_led * 10;  // trasformo in centinaia di ms (unità
base del timer)
  TRACE (( 1, "AT+LED Start led timer:%d s",timer_led ));
    tim_handle  =  adl_tmrSubscribe(  TRUE,time_ms,ADL_TMR_TYPE_
100MS,Led_TimerHandler );
}
/***********************************/
/*  Function   : adl_main         */
/***********************************/
void adl_main( adl_InitType_e InitType )
{
        TRACE (( 1, "Subscribe GPIO_0"));
        // Subcribe GPIO0 output
        myGpioOut_Handle   =   adl_ioSubscribe(NUM_GPIO,Gpio_Con-
fig,0,0,0);
        TRACE (( 1, "handler returns %d", myGpioOut_Handle ));
        TRACE (( 1, "AT+LED Subscribe" ));
        // Sottoscrive il comando AT+LED=<parametro>, la maschera
ADL_CMD_TYPE_PARA|0X0011 indica che è un comando del tipo AT+cmd=X
con un solo parametro ammesso
        adl_atCmdSubscribe("at+led",cmdATLED_Handler,ADL_CMD_
TYPE_PARA|0X0011);
}
```

rsphdl = funzione associata al comando AT; deve essere definita all'interno del codice ed è con questa funzione che si cattura la risposta al comando AT;
… = lista di parametri di tipo stringa che deve essere in grado di rilevare la funzione di risposta. La lista deve terminare con NULL; il carattere "*" all'interno della lista indica che la funzione definita in **rsphdl** riceve tutti i parametri di risposta (in alternativa si possono elencare le stringhe di risposta che la funzione definita in **rsphdl** è in grado di rilevare).

Sempre riguardo al parametro **…** , notate che quando avviene la risposta al comando AT la funzione **rsphdl** può essere richiamata più volte, una per ogni stringa di risposta generata (normalmente è richiamata due volte, una volta per la risposta contenente i parametri e un'altra volta per ricevere l'OK).
È allora possibile definire solo alcune stringhe di risposta ammesse e ignorare le altre.
Quella relativa al parametro **…** è comunque una funzionalità avanzata; per i nostri scopi useremo "*" oppure ADL_STR_OK qualora ci interessasse rilevare solo l'OK.
Nel **Listato 1** è riportato un esempio che richiama il comando AT+CREG e ne rileva la risposta.
Con questo pezzo di codice è possibile quindi determinare lo stato di registrazione alla rete del modulo GSM.
In modo analogo a quanto visto nel **Listato 1**, si può eseguire qualsiasi altro comando AT dall'interno delle proprie applicazioni Open AT.
Nel **Listato 2** è riportato un altro esempio di un frammento di codice utile per rilevare l'evento WIND 4. Questo evento ci dice che il modulo è registrato alla rete e che la SIM è pronta all'uso. Quindi solo dopo aver rilevato il WIND 4 sarà possibile leggere e scrivere in SIM, fare chiamate, inviare SMS ed altro ancora, attraverso la propria applicazione Open AT.
Veniamo infine alla questione più interessante: la possibilità di creare i propri comandi AT personalizzati. Si supponga ad esempio di voler aggiungere alla propria

applicazione un comando AT per far lampeggiare un led con periodo da 0 (spento) a 255 s. Il led è collegato al pin GPIO_0 del modulo Augusto configurato come uscita (vedi puntata precedente). Vogliamo creare un comando AT del tipo:

AT+LED=<tempo>

dove <tempo> è un numero, in secondi, da 0 a 255 che rappresenta il periodo in cui il led rimane acceso. Il duty-cycle è fisso al 50%. Il codice dell'applicazione è riportato nel **Listato 3**. Il codice fa uso della funzione di libreria **adl_atCmdSubscribe** per creare il comando AT, i parametri da passare a questa funzione sono nell'ordine:
- la stringa del comando AT;
- la funzione di risposta al comando AT;
- le opzioni che definiscono il tipo di comando AT secondo la formattazione standard descritta precedentemente e il numero di parametri ammessi (vedere **Tabella 2**).

Per quanto riguarda le funzioni che gestiscono i timer e i GPIO, avendole già introdotte nella scorsa puntata, abbiamo usato parte dello stesso codice dell'esempio "Allarme" visto nel numero precedente, solo che questa volta il tempo di lampeggio del led è un parametro settabile dall'utente con il comando AT+LED.

PRINCIPALI COMANDI AT

Bene, in conclusione di questa puntata, come accennato, descriviamo i principali comandi AT previsti nel protocollo GSM (comandi AT estesi) e quelli specifici delle wireless CPU, che sono tra i più utilizzati nelle varie applicazioni.

AT
Comando di interrogazione di base.
Esempio:
AT
OK

Comandi AT di configurazione

AT+IPR=<baudrate>
Imposta la velocità della porta seriale.
Esempio:
AT+IPR=115200
OK
AT+IPR?
+IPR: 115200
OK

ATE<parameter>
Attiva/Disattiva l'eco dei caratteri.
Esempio:
ATE0 Eco disattivo
OK
ATE1 Eco attivo
OK

AT+CFUN=<parameter>
Riavvio del modulo wireless CPU.
Esempio:
AT+CFUN=1
OK

ATI3
Legge la versione del firmware.
Esempio:
ATI3
R70_00gg.WMP100 2009124 012408 21:14
OK

AT&W
Salva le impostazioni correnti.
Esempio:
AT&W
OK

Comandi AT speciali introdotti da Wavecom

AT+WIND=<IndLevel>
Attivazione dei codici di "Wavecom Indication".
<IndLevel> è il codice dell'evento da attivare secondo la **Tabella 3** riportata a pagina seguente.

bit settato a 0: indication disattivata
bit settato a 1: indication attivata
Esempio:
AT+WIND=255 attiva i primi 8 bit di <IndLevel>
AT+WIND?
+WIND: 255
OK

Quando si genera un evento viene inviata la stringa seguente :

Tabella 3 - *Descrizione dei codici <IndLevel>.*

0	no unsolicited "+WIND: <event>" will occur
1 (bit 0)	activate hardware SIM insert / remove indications or SIM presence after software reset
2 (bit 1)	activate calling party alert indication
4 (bit 2)	activate idle state indication
8 (bit 3)	activate end of Wireless CPU® initialization (after PIN1/CHV 1 code entered) indication
16 (bit 4)	activate new call identifier indication
32 (bit 5)	activate call release indication
64 (bit 6)	activate network service available indication
128 (bit 7)	activate network lost indication
256 (bit 8)	activate audio ON indication
512 (bit 9)	activate SIM phonebooks reload status
1024 (bit 10)	activate SIM phonebooks checksum indication
2048 (bit 11)	activate interruption indication
4096 (bit12)	activate hardware Rack Open/Closed Indication
8192 (bit13)	activate NITZ indication
16384 (bit 14)	activate SMS service ready indication

WIND: <event>

dove <event> è il codice dell'evento. I codici degli eventi e la loro descrizione sono riportati nella **Tabella 4**.

AT+WDWL
Wavecom downloading (comando per programmare il modulo).
Dopo questo comando il modulo risponde con dei caratteri NAK ed è in attesa del download della applicazione. Il download può avvenire via hyper-terminal, inviando il file **.dwl** o **.wpb** in formato 1K X-Modem (vedi prima puntata del corso).

AT+WOPEN=<parameter>
Controllo dell'avvio/cancellazione della applicazione.
Esempio:

AT+WOPEN=1 Attiva l'avvio automatico dell'applicazione
OK
AT+WOPEN=0 Disattiva l'avvio automatico dell'applicazione
OK
AT+WOPEN=3 Cancella la memoria flash dati
AT+WOPEN=4 Cancella la memoria flash programma

Comandi AT riguardanti la gestione della SIM

AT+CPIN
Imposta pin o verifica lo stato della SIM
Esempio:
AT+CPIN?
+CPIN: <code>
dove <code> è lo stato della SIM riportato nella **Tabella 5**.

Se il PIN della SIM viene disabilitato oppure è già stato introdotto correttamente la risposta è READY.

AT+CPBR=<parameter>
Legge la rubrica della SIM.
Esempio:
Legge la prima voce in rubrica
AT+CPBR=1
+CPBR: 1,"112",129,"SOS"
OK

Legge le voci da 1 a 3
AT+CPBR=1,3
+CPBR: 1,"34111223344",129,"Primo nome"
+CPBR: 2,"12345678595",129,"Secondo nome"
+CPBR: 3,"01564888888",129,"Terzo nome"
OK

AT+CPBW =<idx>,<number>,<type>,<text>
Scrive in rubrica nella posizione <idx> il numero <number> associato al nome <text>. <type> è un parametro 129 numero standard oppure 145 numero con il prefisso internazionale ('+' davanti).
Esempio:
Scrive alla posizione 5
AT+CPBW=5,"0491234567",129,"Nicola"
OK
Cancella la posizione 1

AT+CPBW=1
OK

AT+CGMR=<index>
Legge gli SMS ricevuti memorizzati in SIM. <index> è la posizione in SIM dell'SMS da leggere.
Esempio:
AT+CMGR=1
+CMGR: "REC UN-READ","0146290800","98/10/01,18:22:11+00"
ABCdefGHI
OK

AT+CMGW=<number>
Scrive SMS in SIM avente il numero <number>. Dopo l'invio (<CR>) occorre scrivere il messaggio e quindi premere CTRL-Z.
Esempio:
AT+CMGW="+39123456" <CR>
> Ciao a tutti <CRTL-Z>
+CMGW: 1
OK

AT+CMGS=<number>
Invia un SMS verso il numero <number>. Dopo l'invio (<CR>) occorre scrivere il messaggio e quindi premere CTRL-Z.
Esempio:
AT+CMGW="+39123456" <CR>
> Ciao a tutti <CRTL-Z>
+CMGS: 1
OK

AT+CMGD=<index>
Cancella un SMS salvato nella posizione <index> dalla SIM.
Esempio:
AT+CMGD=1
OK

ATD <numero> effettua una chiamata dati
ATD <numero>; effettua una chiamata voce
ATH termina una chiamata

Comandi AT riguardanti la rete

AT+CSQ
Verifica la forza del segnale radio (valore RSSI del campo da 0 assente a 31 ottimo, 99 non rilevabile) e il bit error rate BER (da 0 a 7, 99 non rilevabile).

Tabella 4 - *Descrizione codice eventi.*

Codice evento	Evento
0	the SIM presence pin has detected a "SIM removed"
1	the SIM presence pin has detected a "SIM inserted"
2	the calling party is alerting
3	the product is ready to process AT commands (except phonebooks, AOC, SMS), at init or after AT+CFUN=1
4	the product is ready to process all AT commands, end of phonebook init or swap (FDN to ADN)
5	a call <idx> has been created (after ATD or +CCWA...)
6	a call <idx> has been released, after a NO CARRIER, a "+CSSU: 5" indication, or after the release of a call waiting
7	the network service is available for an emergency call
8	the network is lost
9	the audio channel is opened
10	reload status of each SIM phonebook after init phase (after power on or SIM insertion)
11	checksum of SIM phonebooks after initialization
12	an interruption has occurred
13	the rack has been detected as closed
14	the rack has been detected as opened
15	the Wireless CPU® has received a NITZ information message from the network
16	16 SMS and SMS CB services are initialized

Esempio:
AT+CSQ
+CSQ: 17,1
OK

AT+CREG?
Verifica stato di registrazione alla rete. Risposta del tipo:
+CREG: <mode>,<stat>
dove <mode> è l'impostazione di configurazione del comando fatta con AT+CREG=<mode>, il default è 0 mentre <stat> è lo stato di registrazione dove:
0=non registrato

Tabella 3 - *Descrizione dei codici <IndLevel>.*

READY	ME is not writing for any password
SIM PIN	PIN 1/CHV 1 is required
SIM PUK	PUK1 is required
SIM PIN2	PIN 2/CHV 2 is required
SIM PUK2	PUK2 is required
PH-SIM PIN	SIM lock (phone-to-SIM) is required
PH-NET PIN	network personalization is required
PH-NETSUB PIN	network subset is required
PH- SERVPROV PIN	service provider is required
PH- CORPORATE PIN	corporate is required

1=registrato
2=in fase di ricerca operatore
3=registrazione negata

4=non usato
5=registrato in roaming
Esempio:
AT+CREG?
+CREG: 0,1
OK

Per ulteriori dettagli sui comandi AT qui esposti e su altri comandi AT vi invitiamo a consultare i documenti richiamati nella bibliografia.

Bibliografia:

[1] "AT Commands Interface Guide for Open AT® Firmware v7.4"
[2] "AT command user guide for Wavecom IP v4.01"
[3] "ADL User Guide for Open AT® OS v6.1"

Tutti questi documenti sono scaricabili previa registrazione dal sito Internet della Wavecom, www.wavecom.com.

Corso Wireless CPU

La gestione delle periferiche

Impariamo ad usare Timer, GPIO, Interrupt, UART e USB, bus SPI e I²C di cui sono dotati i moduli WMP50 e WMP100 della Wavecom.

dell'Ing.
NICOLA FRISO

In questa puntata introduciamo le periferiche della Wireless CPU WMP50/WMP100, montata sul modulo Augusto di Shitek Technology. È vero che alcune periferiche come i Timer e i GPIO (General Purpose Input Output) sono già state descritte nelle puntate precedenti del corso, ma in questa sede ci limiteremo a richiamarne le funzioni essenziali; in ogni caso per migliorare la comprensione della materia vi invitiamo a rivedere gli esempi di codice a corredo del corso riportati nelle puntate dei fascicoli numero 142 e numero 143 della nostra rivista.
Altre periferiche di input/output e temporizzazione come gli interrupt vengono qui introdotte per la prima volta. Inoltre, nel corso di questa puntata descriveremo alcuni semplici esempi di codice riguardanti la gestione delle porte di comunicazione UART, USB, SPI e I²C.

Tabella 1 - *Elenco (non esaustivo) degli esempi inclusi nell'ambiente di sviluppo Wavecom M2M Studio.*

Categoria	Nome Esempio	Descrizione
Periferiche HW	ADC RTC DAC Port_manager UART_access Appli_WD Hardware_WD ExternalStorage_IIC ExternalStorage_parallel ExternalStorage_SPI Irq_measure Signal_generator	- Gestione convertitore ADC - Gestione Real Time Clock - Gestione convertitore DAC - Gestione porte UART - Accesso alla UART a basso livello - Gestione software di un watchdog - Gestione hardware di un watchdog - Collegamento memoria EEPROM I^2C - Collegamento memoria parallela - Collegamento memoria EEPROM SPI - Gestione interrupt - Gestione IRQ e GPIO
Stack TCP-IP	Application_Download DOTA Ping_GPRS POP3 SMTP http_get	- Downolad remoto dell'applicazione - Gestione collegamento GPRS - Gestione mail protocollo POP3 - Gestione mail protocollo SMTP - Gestione comandi GET/POST protocollo HTTP
SMS & Data connection	Duplex_Data Telemetry SMS_Automation Remote_terminal	- Chiamata tra due dispositivi - SMS di telemetria - Ricezione SMS - Gestione connessione dati con libreria FCM
Audio	Call_monitoring DTMF_Decoding Dtmf_keyboard Sound_Demo PCM_record_and_play	- Monitor chiamate in ingresso - Decodificatore toni DTMF - Generatore toni DTMF da tastiera - Generazione suoni - Gestione periferica PCM
Sistema Operativo	Queque Multitasking_Message Multitasking_Semaphore Multitasking_Timers	- Gestione code - Gestione messaggi - Gestione semafori - Gestione timer
Generale	Hello_World Dhrystone Bug	- Primo esempio di partenza - Algoritmo di benchmark della WCPU - Simulazione errori

I TIMER

Come per tutte le altre risorse messe a disposizione dal sistema operativo Open AT, ogni volta che si ha bisogno di un timer occorre sottoscriverlo attraverso la sua specifica funzione di libreria ADL (Application Development Layer) **adl_tmrSubscribe.** Occorre poi definire una variabile chiamata *handle* ("gancio") di tipo **adl_tmr_t**, per rilevare il valore di ritorno della funzione di sottoscrizione. Infine dobbiamo definire una funzione di "aggancio" (handler) in cui il programma salta al termine della temporizzazione. Il prototipo della funzione di sottoscrizione dei timer è il seguente:

*adl_tmr_t *adl_tmrSubscribe(bool bCyclic, u32 TimerValue, adl_tmrType_e TimerType, adl_tmrHandler_t Timerhdl)*

I parametri da passare alla funzione **adl_tmrSubscribe** sono quelli elencati qui di seguito.

- bCyclic TRUE=il timer è ciclico (allo scadere riparte di nuovo); FALSE=il timer si ferma al termine del tempo impostato.
- TimerValue Numero di periodi dopo il quale il timer termina.
- TimerType Unità di misura dei periodi (può essere espresso in ms, quindi ADL_TMR_TYPE_100MS pari a 100ms o in ticks ADL_TMR_TYPE_TICK, 1 tick = 18,5 ms)
- Timerhdl Funzione che gestisce il timer allo scadere del tempo.

La funzione **ald_tmrUnSubscribe** serve invece per fermare il timer. Un esempio pratico che utilizza queste funzioni lo si trova nelle precedenti puntate del Corso; altri esempi sono inclusi nell'ambiente di sviluppo M2M Studio (vedi **Tabella 1**) e nel capitolo "Timers" che trovate nel testo di cui al punto [1] della bibliografia. Per completezza introduciamo anche un altro nuovo modo di sottoscrivere un timer con la funzione **adl_tmrSubscribeExt**. Questa funzione, il cui prototipo è il seguente:

*adl_tmr_t *adl_tmrSubscribeExt (adl_tmrCyclicMode_e CyclicOpt, u32 TimerValue, adl_tmrType_e TimerType, adl_tmrHandler_t Timerhdl, void * Context, bool Strict);*

ha più parametri rispetto alla precedente, soprattutto per ciò che riguarda la modalità di funzionamento ciclico del timer. Vediamo il significato di questi parametri.

- CyclicOpt permette di settare diversi tipi di timer ciclico, definiti dalle costanti riportate nella

Tabella 2 qui accanto.

- TimerValue Numero di periodi dopo il quale il timer termina.
- TimerType Unità di misura dei periodi (può essere espresso in ms ADL_TMR_TYPE_100MS pari a 100ms o in ticks ADL_TMR_TYPE_TICK, 1 tick = 18,5 ms).
- Timerhdl Funzione che gestisce il timer allo scadere.
- Context Definisce il contesto del particolare task in cui viene usato il timer (se non è stato definito e avviato un task, vale NULL).
- Strict FALSE=il timer non sveglia la CPU dalla modalità di basso consumo *Slow Idle Mode*, perciò qualora si entrasse in tale modalità il timer viene ritardato; TRUE=il timer è in grado di svegliare la CPU dalla modalità di basso consumo *Slow idle mode* (per le modalità di utilizzo della WCPU si rimanda ai testi richiamati ai punti [3] e [4] della bibliografia).

Un semplice esempio tratto dal testo [1] della bibliografia che utilizza entrambi i metodi di sottoscrizione "normale" ed "ext" , è riportato nel **Listato 1**. Vi invitiamo a provarne il funzionamento. In questo caso il primo timer sottoscritto termina dopo il suo periodo di funzionamento poiché il parametro *bCyclic* è stato impostato FALSE. Il secondo timer, invece, occorre bloccarlo manualmente con la funzione **adl_tmrUnSubscribe**, perché il parametro *CyclicOpt* è stato settato come ADL_TMR_CYCLIC_OPT_ON_RECEIVE.

I GPIO

Il modulo Augusto di Shitek Technology mette a disposizione undici linee di input/output chiamate GPIO, definite nella **Tabella 3** a seconda del tipo di WCPU WMP50 o WMP100 che monta il modulo. L'intervallo di tensione ammesso per tutti i pin di GPIO è 0÷2,8 V. Nelle scorse puntate abbiamo già descritto la struttura dati **adl_ioDefs_t** definita nelle librerie ADL, che viene utilizzata per la configurazione dei pin di GPIO. Una volta dichiarata e inizializzata questa struttura, la stessa viene richiamata come parametro in fase di sottoscrizione dei GPIO. I pin saranno effettivamente operativi solo dopo aver sottoscritto questa struttura. Le funzioni per la scrittura e lettura di un GPIO sono la **adl_ioWriteSingle** e la **adl_ioReadSingle,** che abbiamo già visto. Esempi pratici di configurazione e utilizzo dei GPIO sono già stati esposti nelle due puntate

***Tabella 2** - Opzioni di configurazione di un timer ciclico.*

ADL_TMR_CYCLIC_OPT_NONE	"One shot timer" il timer termina automaticamente alla fine del periodo impostato.
ADL_TMR_CYCLIC_OPT_ON_EXPIRATION	Timer ciclico che necessita di essere de-sottoscritto al termine del periodo impostato immediatamente prima dello scatenarsi dell'evento di fine temporizzazione (salto nella funzione di handler). (*)
ADL_TMR_CYCLIC_OPT_ON_RECEIVE	Timer ciclico che necessita di essere de-sottoscritto al termine del periodo impostato immediatamente all'inizio dello scatenarsi dell'evento di fine temporizzazione (cioè dentro alla funzione di handler). (*)

(*) La differenza tra i due comportamenti è che nel caso ON_EXPIRATION, se l'applicazione è occupata a fare altro non è garantita la durata minima del timer. Nel secondo caso (ON_RECEIVE), poiché lo stop avviene esattamente quando si scatena l'evento, la durata è garantita.

Listato 1

```
adl_tmr_t *tt, *tt2;
u16 timeout_period = 5; //in 100 ms steps;
void Timer_Handler(u8 Id, void * Context)
{
//We do not unsubscribe to the timer because it has
//'naturally'expired
adl_atSendResponse(ADL_AT_RSP, "\r\Timer timed out\r\n");
}

void Timer_Handler2(u8 Id, void * Context)
{
//Unsubscribe from the timer resource
adl_tmrUnSubscribe (tt2, Timer_Handler2);
}

//main function
void adl_main (adl_InitType_e adlInitType)
{
//We set up a one-shot timer
tt = adl_tmrSubscribe (FALSE,timeout_period,
ADL_TMR_TYPE_100MS,Timer_Handler);
//We set up a cyclic timer
tt2 = adl_tmrSubscribeExt (ADL_TMR_CYCLIC_OPT_ON_RECEIVE,
timeout_period,ADL_TMR_TYPE_100MS,Timer_Handler2,NULL,FALSE);
}
```

Tabella 3 - *Nome dei pin di I/O nel WMP50 e nel WMP100.*

WMP 50 pin out naming	WMP 100 pin out naming
INT0	INT1
INT1	INT3
GPIO0	GPIO19
GPIO1	GPIO20
GPIO2	GPIO21
GPIO3	GPIO22
GPIO4	GPIO23
GPIO5	GPIO24
GPIO6	GPIO26 (SCL pin in I²C bus)
GPIO7	GPIO27 (SDA pin in I²C bus)
GPIO8	GPIO32
GPIO9	GPIO35
GPIO10	GPIO44
ADC1/BAT_TEMP	BAT_TEMP
ADC2	AUX_ADC0

precedenti del corso.
Nel **Listato 2** è riportato un esempio di codice tratto dal testo di cui al punto [1] della bibliografia, che riassume quanto già visto finora ed aggiunge anche la gestione ad eventi dei GPIO (esegue il polling automatico su tali pin senza l'ausilio di timer). Vediamo dunque cosa fa l'esempio del **Listato 2**, in particolare per ciò che riguarda la gestione degli eventi legati al cambio di stato dei pin di GPIO. Le dichiarazioni iniziali le abbiamo già analizzate nella scorsa puntata e definiscono quali e quanti pin di GPIO utilizzare ed eventualmente anche il loro stato iniziale. In questo caso sono stati definiti due gruppi di GPIO: il primo comprende i pin GPIO1 e GPIO2 che saranno utilizzati in polling, mentre il secondo contempla il solo pin GPIO3, che sarà sottoscritto senza polling. Per i due gruppi di GPIO vengono dichiarati quindi due identificativi "handle" distinti: *MyGpioHandle1* e *MyGpioHandle2*. Viene dichiarato anche un handle per la gestione dell'evento di cambio di stato sui pin GPIO1 e GPIO2, chiamato *MyGpioEventHandle*. Al termine delle dichiarazioni si trova la funzione che gestisce il cambio di stato qui chiamata *MyGpioEventHandler*. Il prototipo della funzione è il seguente:

*void MyGpioEventHandler (s32 GpioHandle, adl_ioEvent_e Event, u32 Size, void * Param)*

I parametri della funzione sono:
- GpioHandle Handle della risorsa GPIO;
- Event Identificatore dell'evento (può essere solo ADL_IO_EVENT_INPUT_CHANGED cioè un evento di cambio stato dei pin);
- Size Numero di elementi della tabella **Param** (parametro successivo);
- Param Tabella di lettura dello stato dell'evento (è un puntatore alla struttura **adl_ioDefs_t**).

La funzione *MyGpioEventHandler* è stata scritta per poter essere in grado di rilevare qualsiasi cambio di stato associato al gruppo di GPIO sottoscritto. Questo complica un po' il codice, ma permette di usare la stessa funzione in tutte le situazioni. Lo switch/case iniziale verifica che l'evento sia di tipo "cambio stato"; poi viene fatto un ciclo *for* per scorrere tutti gli elementi della tabella di tipo *adl_ioDefs_t* dove sono stati inseriti i GPIO da monitorare. Attraverso la riga di codice:

*if ((ADL_IO_TYPE_MSK &((adl_ioDefs_t *) Param)[My_Loop])&& ADL_IO_GPO)*

si va a vedere se la tabella contiene qualche pin di GPO (General Purpose Output) che è cambiato di stato. Esistono infatti moduli Wavecom in cui alcuni pin sono configurabili solo come uscita e il codice è scritto per essere usato con tutta la famiglia dei prodotti Open AT. Se non ci si trova in questo caso, si salta all'*else* dove si è certi che il cambio di stato riguarda un pin di GPIO. La TRACE è uguale per i due casi e si occupa di visualizzare in debug il nuovo stato del pin.
Se ad esempio GPIO1 passa da 0 ad 1, sulla trace di debug di M2M Studio, si vedrà la stringa "GPIO 1 new value: 1".
Nella istruzione TRACE il numero di GPIO viene ricavato mettendo in AND logico la tabella **Param** con la maschera predefinita ADL_IO_NUM_MSK. Lo stato del pin viene ricavato allo stesso modo con in più una AND con la maschera ADL_IO_LEV_HIGH. La successiva funzione *MyFunction* sottoscrive gli

I/O e richiama la funzione **adl_ioEventSubscribe** per sottoscrivere anche l'evento associato con la funzione *MyGpioEventHandler*. A seguire, il pin GPIO1 viene settato alto; poi si legge lo stato di tutti i pin sottoscritti e infine vengono de-sottoscritti i GPIO e l'evento. L'intera funzione *MyFunction* è solo un esempio che potete richiamare per intero o in parte all'interno delle vostre applicazioni. Ulteriori dettagli sui servizi associati ai pin di GPIO si possono trovare, al capitolo "GPIO Service", nel testo di cui al punto [1] della bibliografia e negli esempi inclusi con l'M2M Studio riportati nella **Tabella 1**.

INTERRUPT

La gestione degli interrupt è affidata anch'essa a delle funzioni di libreria ADL e a delle particolari strutture dati dedicate, già definite nella libreria. La più importante è la **adl_irqID_e**, che definisce tutti i tipi di interrupt disponibili. Ci sono interrupt legati all'audio, o allo scadere dei timer, interrupt di tipo capture-compare in grado di rilevare tempi tra un evento ed un altro, interrupt legati allo stato o al fronte di salita e/o discesa di un pin, ecc.
Descrivere tutte le possibili combinazioni di interrupt esula dagli scopi di questo corso e richiederebbe una puntata intera; vediamo invece, al solito, un esempio pratico con tutti i passi da fare per configurare ed attivare una risorsa di interrupt.
Il codice riportato nel **Listato 3** è relativo alla gestione di un interrupt di tipo ADL_IRQ_ID_EXTINT che è collegato ad un pin: in questo caso il pin INT0 della Wireless CPU (WCPU) WMP50 montata sul modulo Augusto. Questo è un particolare tipo di interrupt, ma le funzioni che gestiscono gli interrupt e quindi anche i vari passi

Listato 2

```
// Global variables & constants

// Subscription data
#define GPIO_COUNT1 2
#define GPIO_COUNT2 1
u32 My_Gpio_Label1 [ GPIO_COUNT1 ] = { 1 , 2 };
u32 My_Gpio_Label2 [ GPIO_COUNT2 ] = { 3 };

adl_ioDefs_t* MyGpioConfig1 [ GPIO_COUNT1 ] =
{ ADL_IO_GPIO | 1| ADL_IO_DIR_OUT | ADL_IO_LEV_LOW ,
{ ADL_IO_GPIO | 2| ADL_IO_DIR_IN };

adl_ioDefs_t* MyGpioConfig2 [ GPIO_COUNT2 ] =
{ ADL_IO_GPIO | 3| ADL_IO_DIR_IN };

// Gpio Event Handle
s32 MyGpioEventHandle;

// Gpio Handles
s32 MyGpioHandle1, MyGpioHandle2;

// GPIO event handler
void MyGpioEventHandler ( s32 GpioHandle, adl_ioEvent_e Event, u32 Size,
void * Param )
{
// Check event
switch ( Event )
{
 case ADL_IO_EVENT_INPUT_CHANGED :
 {
  u32 My_Loop;
  // The subscribed input has changed
  for ( My_Loop = 0 ; My_Loop < Size ; My_Loop++)
  {
   if (( ADL_IO_TYPE_MSK &((adl_ioDefs_t *)Param)[ My_Loop])&& ADL_IO_GPO)
   {
    TRACE (( 1, "GPO %d new value: %d",
          (((adl_ioDefs_t *)Param)[ My_Loop ] ) & ADL_IO_NUM_MSK ,
          ((((adl_ioDefs_t *)Param)[ My_Loop ]) & ADL_IO_LEV_MSK )
           && ADL_IO_LEV_HIGH ));
   }
   else
   {
    TRACE (( 1, "GPIO %d new value: %d",
    (((adl_ioDefs_t *)Param)[ My_Loop ] ) & ADL_IO_NUM_MSK ,
    ((((adl_ioDefs_t *)Param)[ My_Loop ] ) & ADL_IO_LEV_MSK ) &&
    ADL_IO_LEV_HIGH ));
   }
  }
}
break;
}
}
...
// Somewhere in the application code, used as an event handler
void MyFunction ( void )
{
// Local variables
s32 ReadValue;
// Subscribe to the GPIO event service
MyGpioEventHandle = adl_ioEventSubscribe ( MyGpioEventHandler );

// Subscribe to the GPIO service (One handle without polling,
// one with a 100ms polling process)
MyGpioHandle1 = adl_ioSubscribe ( GPIO_COUNT1, MyGpioConfig1, 0, 0,
0 );
MyGpioHandle2 = adl_ioSubscribe ( GPIO_COUNT2, MyGpioConfig2,
ADL_TMR_TYPE_100MS, 1, MyGpioEventHandle );

// Set output
adl_ioWriteSingle ( MyGpioHandle1, &Gpio_to_write1 , TRUE );

// Read inputs
ReadValue = adl_ioReadSingle (MyGpioHandle1, &Gpio_to_read1 );
ReadValue = adl_ioReadSingle (MyGpioHandle2, &Gpio_to_read2 );

// Unsubscribe from the GPIO services
adl_ioUnsubscribe ( MyGpioHandle1 );
adl_ioUnsubscribe ( MyGpioHandle2 );

// Unsubscribe from the GPIO event service
adl_ioEventUnsubscribe ( MyGpioEventHandle );
}
```

Listato 3

```
//Global variables
//use the PIN0 for the Ext Int
#define EXTINT_PIN0 0
//ExtInt service handle
s32 ExtIntHandle;
//IRQ service handle
s32 IrqHandle;
//ExtInt configuration: both edge detection without filter
adl_extintConfig_t extintConfig =
{ ADL_EXTINT_SENSITIVITY_BOTH_EDGE ,
ADL_EXTINT_FILTER_BYPASS_MODE,0,0, NULL };

//ExtInt interrupt handler
bool MyExtIntHandler (adl_irqID_e Source,adl_irqNotifyLevel_e
NotificationLevel,adl_irqEventData_t * Data)
{
//Read the input status
adl_extintInfo_t Status, * AutoReadStatus;
adl_extintRead (ExtIntHandle, &Status);
//Input status can also be obtained from the auto read option.
AutoReadStatus = (adl_extintInfo_t *) Data->SourceData;
return TRUE;
}

//Somewhere in the application code, used as event handlers
void MyFunction1 (void)
{
adl_extintCapabilities_t My_ExtInt_Capa;
adl_extintGetCapabilities (&My_ExtInt_Capa);
//Test if the Wireless CPU® have Ext Int pin
if (My_ExtInt_Capa.NbExternalInterrupt >= 1)
          {
          //Subscribes to the IRQ service
          IrqHandle = adl_irqSubscribe (MyExtIntHandler,ADL IRQ
          NOTIFY_LOW_LEVEL, ADL_IRQ_PRIORITY_HIGH_LEVEL,
          ADL_IRQ_OPTION_AUTO_READ);
          //Configures comparator channel
          ExtIntHandle = adl_extintSubscribe (EXTINT_PIN0,
          IrqHandle,0,&extintConfig);
          }
}

void MyFunction2 (void)
{
// Un-subscribes from the ExtInt service
adl_extintUnsubscribe (ExtIntHandle);
}
```

da compiere per configurarlo, sono molto simili per tutti i servizi di interrupt offerti dal sistema operativo. L'interrupt dell'esempio scatta quando si rileva un fronte di salita o di discesa sul pin. Non sono stati applicati filtri, ma esiste la possibilità di settare in automatico, ad esempio, un filtro anti-rimbalzo che fa scattare l'interrupt solo quando lo stato del pin è stabile per un certo numero di impulsi di clock impostabili. La prima riga di codice del **Listato 3** definisce quale pin di interrupt utilizzare; in questo caso abbiamo scelto INT0. Il modulo Augusto mette a disposizione 2 ingressi di interrupt (INT0 e INT1) entrambi ad alta impedenza, per cui nell'usarli si raccomanda di mettere loro una resistenza di pull-up. La procedura per l'inizializzazione e configurazione dei servizi di interrupt è molto simile a quello che abbiamo visto nelle puntate precedenti per gestire qualsiasi altra risorsa della WCPU; essa si svolge nei passi descritti qui di seguito.

- Si definisce un handle alla risorsa di interrupt EXTINT e un handle generico *IrqHandle* per i servizi di interrupt in generale.
- Si configura l'interrupt dichiarando la struttura: *adl_extintConfig_t extintConfig = { ADL_EXTINT_SENSITIVITY_BOTH_EDGE , ADL_EXTINT_FILTER_BYPASS_MODE , 0,0, NULL };* che definisce l'interrupt sensibile ad entrambi i fronti, senza filtri applicati. Come si vede, il nome dato alle strutture dati è già di per sè descrittivo della configurazione.
- Si deve quindi scrivere la funzione che qui abbiamo chiamato *MyExtIntHandler*. È la funzione cui il flusso di programma salta quando si genera l'evento di interrupt. La struttura predefinita di tipo **adl_extintInfo_t** dichiarata all'interno della funzione serve per leggere le informazioni relative all'interrupt che si è generato. La riga di codice: *AutoReadStatus = (adl_extintInfo_t *) Data->SourceData;* serve per riempire la struttura *adl_extintInfo_t* con le informazioni relative all'interrupt. In questo caso la struttura contiene solo lo stato del pin di interrupt ed è definita nella libreria ADL come segue:
  typedef struct
  {
  u8 PinState;
  } adl_extintInfo_t;
- La funzione chiamata "MyFunction1" deve essere richiamata da qualche parte all'interno del proprio programma (ad esempio nell'adl_main) e serve per sottoscrivere, configurare ed attivare fisicamente l'interrupt. Per prima cosa, con la funzione di libreria **adl_extintGetCapabilities** viene riempita una struttura dati predefinita che serve a verificare se quel tipo di interrupt è supportato dal sistema operativo e dalla WCPU. Questo viene fatto con l'if successivo: *if (My_ExtInt_Capa.NbExternalInterrupt >= 1)*. Una volta verificato che esiste tale risorsa di interrupt, si comunica al sistema operativo che si utilizza un servizio di interrupt ad alta priorità; ciò avviene con la funzione generica di sottoscrizione degli interrupt **adl_irqSubscribe.** A questa funzione di libreria si passa come parametro anche la nostra funzione *MyExtIntHandler* associata al particolare tipo di interrupt che si

vuole utilizzare (in questo caso EXTINT). La successiva sottoscrizione con la funzione **adl_extintSubscribe** è la sottoscrizione specifica degli interrupt di tipo EXTINT. Con questa funzione si configura l'interrupt specifico e lo si attiva.

- La funzione "MyFunction2" deve invece essere richiamata ogni volta che si vuole disattivare l'interrupt. Per far questo occorre de-sottoscriverlo con la funzione di libreria **adl_extintUnsubscribe**.

Questo che abbiamo descritto è solo un tipo particolare di interrupt, configurato in modo automatico ad alta priorità; vi invitiamo a prendere visione del documento richiamato al punto [1] nella bibliografia, al capitolo "IRQ Service" e successivi, per approfondire il discorso sui servizi di interrupt. Sicuramente vi troverete una configurazione adatta ai vostri scopi. A proposito, suggeriamo di trarre spunto dagli esempi di codice realizzati da Wavecom, disponibili con l'ambiente di sviluppo M2M Studio (vedi **Tabella 1**).

LE PORTE UART E L'USB

Abbiamo già visto nelle scorse puntate come inviare una stringa sulla porta seriale attraverso la funzione **adl_atSendResponse**. Vediamo ora, nel dettaglio, come configurare ed utilizzare gli UART. Le stesse strutture dati e funzioni di libreria sono utilizzate anche per gestire la comunicazione sulla porta USB. Per attivare una comunicazione bidirezionale sulla porta desiderata, utilizziamo un insieme di librerie che vanno sotto il nome di servizi **FCM** (Flow Control Manager) insieme a dei comandi AT specifici per configurare ed inizializzare la porta. In alternativa avremmo

Listato 4 (continua...)

```
#include "adl_global.h"


// header
adl_fcmCtrlHdlr_f FCM_ControlHandler(adl_fcmEvent_e Event);
adl_fcmDataHdlr_f FCM_DataHandler(u16 DataLen, u8* Data);

u32 wm_apmCustomStack [ 256 ];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

s8 FCMHandler; //declare FCM handler used for subscription


/*****************************************************************************/
/*  Function   : adl_main                                                    */
/*****************************************************************************/

void adl_main ( adl_InitType_e  InitType )
{
        s8 res;
        TRACE (( 1, "Embedded : Appli Init" ));


        //open uart2
        res = adl_atCmdCreate( "AT+WMFM=0,1,2", FALSE, (adl_atRspHandler_t)NULL,
NULL );
        if(res ==OK)
                TRACE((1,"INIT:uart2 opened"));
        else
                TRACE((1,"INIT:uart2 failed to open"));
        //set to 9600 baud
        res = adl_atCmdCreate( "AT+IPR=9600", ADL_AT_PORT_TYPE( ADL_AT_UART2,
FALSE ), (adl_atRspHandler_t)NULL, NULL );
        if(res ==OK)
                TRACE((1,"INIT:uart2 9600 baud"));
        else
                TRACE((1,"INIT:uart2 cannot change baud"));


        // disable flow control
        res = adl_atCmdCreate( "AT+IFC=0,0", FALSE, (adl_atRspHandler_t)NULL,
NULL );
        if(res ==OK)
                TRACE((1,"INIT:uart2 disable flow control"));
        else
                TRACE((1,"INIT:uart2 failed to disable flow control"));

        //subscribe to port
        TRACE((1,"INIT: subscribing to uart2"));
        //uart2_handle=adl_fcmSubscribe(ADL_PORT_UART2,uart2_ctrl_
handler,uart2_data_handler);

        //subsribe to FCM service
        //FCMHandler = adl_fcmSubscribe(ADL_FCM_FLOW_V24_UART2, FCM_ControlHan-
dler, FCM_DataHandler);
        FCMHandler = adl_fcmSubscribe(ADL_PORT_UART2, FCM_ControlHandler, FCM_
DataHandler);
}


//implementation of FCM_ControlHandler
adl_fcmCtrlHdlr_f FCM_ControlHandler(adl_fcmEvent_e Event)
{
        s8 s8r;
        switch (Event)
        {
        case ADL_FCM_RET_ERROR_GSM_GPRS_ALREADY_OPENNED :
                {
                        TRACE((1, "Flow GSM already Opened"));
                        break ;
                }
        case ADL_RET_ERR_PARAM :
                {
                        TRACE((1, "Open Flow GPRS Parameters Error"));
                        break ;
                }
        case ADL_RET_ERR_ALREADY_SUBSCRIBED :
                {
                        TRACE((1, "Flow GPRS already subscribed"));
                        break ;
                }
        case ADL_FCM_EVENT_FLOW_OPENNED :
                {
```

Listato 4 (continuazione)

```
                        TRACE((1, "ADL_FCM_EVENT_FLOW_OPENNED"));
                        adl_atSendResponse(ADL_AT_UNS,"fcmSubscribe  openned\
r\n");
                        s8r=adl_fcmSwitchV24State(FCMHandler,ADL_FCM_V24_
STATE_DATA);
                        TRACE(( 1, "adl_fcmSwitchV24State Ret= %d", s8r));
                        break ;
                }
        case ADL_FCM_EVENT_FLOW_CLOSED :
                {
                        TRACE((1, "ADL_FCM_EVENT_FLOW_CLOSED"));
                        break ;
                }
        case ADL_FCM_EVENT_RESUME :
                {
                        TRACE((1, "ADL_FCM_EVENT_RESUME"));
                        break ;
                }
        case ADL_FCM_EVENT_MEM_RELEASE :
                {
                        TRACE((1, "ADL_FCM_EVENT_MEM_RELEASE"));
                        break ;
                }
        case ADL_FCM_EVENT_V24_DATA_MODE:
                {
                        TRACE((1, "ADL_FCM_EVENT_V24_DATA_MODE"));
                        adl_atSendResponse(ADL_AT_UNS,"now   be   data_mode\
r\n");
                        s8r=adl_fcmSendData(FCMHandler,"abc",3);
                        TRACE(( 1, "adl_fcmSendData Ret= %d", s8r));
                        break ;
                }
        case ADL_FCM_EVENT_V24_AT_MODE:
                {
                        TRACE((1, "ADL_FCM_EVENT_V24_DATA_MODE"));
                        adl_atSendResponse(ADL_AT_UNS,"now be at_mode\r\n");
                        break ;
                }
        default :
                {
                        TRACE((1, "Embedded : FCM_ControlHandler Event not
processed"));
                        break ;
                }
        }
        return TRUE ;
}


//implementation of FCM_DataHandler
adl_fcmDataHdlr_f FCM_DataHandler(u16 DataLen, u8* Data)
{
        TRACE(( 1, "FCM_DataHandler"));

        adl_fcmSendData(FCMHandler,Data,DataLen);     // Data received: Echo to
uart

        DUMP( 1, Data, DataLen);
        return TRUE ;
}
```

potuto utilizzare i servizi messi a disposizione dalla libreria **adl_OpenDevice** (vedi il capitolo "ADL Open Device Service" e successivi nel documento [1] della bibliografia). Questa libreria risparmia di richiamare comandi AT per configurare la porta, ma per contro è molto più difficile da utilizzare, richiede più codice e la sua trattazione esula dagli scopi di questo corso. Vediamo al solito subito un esempio, che trovate riportato nel **Listato 4**; per ulteriori approfondimenti rimandiamo al testo di cui al punto [1] della bibliografia, al capitolo "FCM Service". Con queste funzioni di libreria si possono anche gestire i dati provenienti da collegamenti dati GSM o GPRS.

L'esempio del **Listato 4** riguarda la gestione della UART2; in modo analogo è possibile gestire la UART1 e l'USB (la porta aperta per impostazione predefinita è la UART1). Si dà per scontato che abbiate già familiarità con le porte UART e i principali parametri di configurazione come il bit rate, il numero di bit, il numero di bit di stop, il controllo di parità e il controllo di flusso.

Per prima cosa viene lanciato il comando **AT+WMFM=0,1,2**. Questo comando AT (vedi punto [2] della bibliografia) serve per aprire la porta UART2. I tre parametri sono rispettivamente:

- <type_of_action>: 0 = gestisce l'attivazione della porta, 1 = gestisce i dati in arrivo in una connessione dati GSM o GPRS;
- <mode>: 0 = chiude la porta oppure setta la modalità di ricezione dinamica dei dati di un collegamento dati GSM o GPRS, 1 = apre la porta oppure attiva il collegamento dati, 2 = legge lo stato della porta;
- <port_id>: 1=UART1, 1x=UART1 in modo dati DLC(Data link connection), 2=UART2, 2x=UART2 in modo DLC, 3=USB, 3x=USB in modo DLC, 4=porta dinamica solo lettura (type_of_action=1), 80=Riservato per applicazioni Open AT.

La modalità DLC non verrà trattata; vi basterà sapere che per ogni porta è possibile attivare quattro flussi dati virtuali (x = 1,2,3,4) per cui il nome delle porte in questo caso diventa *UART 11,12,13,14* per la UART1 e *UART 21,22,23,24* per la UART2. Il successivo comando **AT+IPR=9600** setta la velocità della porta aperta (9.600 bit/s); si noti come al secondo parametro della funzione **adl_atCmdCreate** viene passata la macro **ADL_AT_PORT_TYPE(ADL_AT_UART2, FALSE)** tramite la quale si comunica al sistema opera-

tivo di impostare tale velocità sulla UART2. La descrizione della funzione **adl_atCmdCreate** l'abbiamo già vista nella precedente puntata del corso dedicata ai comandi AT. L'ultimo comando AT che viene inoltrato al sistema è **AT+IFC=0,0,** e con esso si disabilita il controllo di flusso (RTS,CTS) sulla porta (è abilitato per impostazione predefinita). Qualora si volessero cambiare anche i parametri di bit di stop, lunghezza della parola dati e controllo di parità, si può utilizzare il comando **AT+IFC** con le stesse modalità sopra descritte. Con questi tre comandi abbiamo aperto la UART2 in modalità (8,N,1 default) a 9.600bit/s senza controllo di flusso. Otteniamo quindi l'handle identificatore della porta, con la riga di codice:

FCMHandler = adl_fcmSubscribe(ADL_PORT_UART2, FCM_ControlHandler, FCM_DataHandler);

che va a sottoscrivere il servizio FCM relativo alla porta UART2. Oltre all'identificativo della porta ADL_PORT_UART2 occorre passare come parametro anche la funzione di controllo chiamata nel codice *FCM_ControlHandler* che, come vedremo, si occupa di rilevare tutti gli eventi che occorrono in fase di configurazione della porta. Il terzo parametro è la funzione *FCM_DataHandler*, che gestisce la ricezione dei dati dalla porta. Entrambe sono funzioni definite dall'utente che devono essere presenti nel codice della applicazione. Vediamone il loro contenuto, esposto qui di seguito.

- La funzione *FCM_ControlHandler* è costituita da uno "switch" con una serie di "case" per ognuno degli eventi che si vuole gestire. Gli eventi possono essere l'apertura o chiusura della porta, il passaggio in modalità modem V24 con controllo di flusso, gli errori di riapertura di una porta già aperta, ecc. Il nome delle costanti associate a questi eventi è sufficientemente autoesplicativo; per ulteriori dettagli si rimanda al testo richiamato al punto [1] della bibliografia.
- La funzione di "aggancio" dei dati ricevuti qui chiamata *FCM_DataHandler* contiene nei suoi parametri la lunghezza del blocco dati ricevuto e il puntatore al buffer dei dati. In questo caso abbiamo semplicemente preso i dati ricevuti e fatto l'eco sulla stessa UART2 tramite la funzione di libreria **adl_fcmSendData**. A questa funzione vengono passati l'handle al servizio FCM precedentemente sottoscritto, il puntatore al buffer contenente i dati e la lunghezza del blocco dati che si vuole inviare sulla porta. Infine la riga di codice *DUMP(1,Data,DataLen)* è una riga di debug analoga alla macro *TRACE* già descritta nelle scorse puntate. Con *DUMP* è possibile visualizzare in modalità di debug il contenuto della memoria di un blocco dati, passando come parametri il puntatore al blocco e la dimensione del blocco.

IL BUS SPI E IL BUS I²C

La WCPU WMP50 dispone di una porta di comunicazione SPI, la WCPU WMP100 dispone di due porte SPI e una porta I²C. La WCPU WMP100 se montata sul modulo Augusto dispone solamente della porta SPI1 e della porta I²C1, secondo la pinout riportato in **Tabella 1**. Diamo per scontato abbiate già qualche familiarità con il mondo dei microcontrollori e che quindi conosciate il funzionamento di una porta SPI e di una porta I²C e i loro principali parametri di configurazione. Le librerie ADL di Open AT mettono a disposizione delle strutture dati predefinite che permettono di impostare facilmente tutti questi parametri. Una descrizione completa e dettagliata di tutte queste strutture dati e delle funzioni che gestiscono le porte, necessiterebbe di una puntata dedicata del corso. Per ragioni di spazio, semplicità e immediatezza, vediamo subito degli esempi pratici.
Il primo riguarda la configurazione del bus **SPI** del modulo. L'esempio, riportato nel **Listato 5**, usa anche le funzioni **adl_busRead** e **adl_busWrite** per leggere e scrivere byte dalla porta. Sempre con riferimento al **Listato 5**, per prima cosa si dichiara una struttura dati di tipo **adl_busSPISettings_t** chiamata in questo caso *"MySPIConfig"* tramite la quale si vanno ad impostare delle costanti di configurazione della porta (si rimanda al punto [1] della bibliografia capitolo "Bus Service" per l'elenco completo di tutti i parametri di configurazione disponibili). La descrizione della configurazione dell'esempio è, nell'ordine, la seguente:

- il clock è in modo 0 (stato di riposo a zero e dato valido sul fronte di salita);

Listato 5

```
// SPI Subscription data
const adl_busSPISettings_t MySPIConfig =
{
1, // No divider, use full clock speed
ADL_BUS_SPI_CLK_MODE_0, // Mode 0 clock
ADL_BUS_SPI_ADDR_CS_GPIO, // Use a GPIO to handle the Chip Select
signal
ADL_BUS_SPI_CS_POL_LOW, // Chip Select active in low state
ADL_BUS_SPI_MSB_FIRST, // Data are sent MSB first
ADL_IO_GPIO | 31, // Use GPIO 31 to handle the Chip Select
signal
ADL_BUS_SPI_LOAD_UNUSED, // LOAD signal not used
ADL_BUS_SPI_DATA_BIDIR, // 2 Wires configuration
API
ADL_BUS_SPI_MASTER_MODE, // Master mode
ADL_BUS_SPI_BUSY_UNUSED // BUSY signal not used
};
// Write/Read buffer sizes
#define WRITE_SIZE 5
#define READ_SIZE 3

// Access configuration structure
adl_busAccess_t AccessConfig =
{
0, 0 // No Opcode, No Address
};

// BUS Handles
s32 MySPIHandle;
// Data buffers
u8 WriteBuffer [ WRITE_SIZE ], ReadBuffer [ READ_SIZE ];


void MyFunction ( void )
{
// Local variables
s32 ReadValue;
u32 AddSize=0;
// Subscribe to the SPI1 BUS
MySPIHandle = adl_busSubscribe ( ADL_BUS_ID_SPI, 1, &MySPIConfig );
// Configure the Address length to 0 (rewrite the default value)
adl_busIOCtl ( MySPIHandle, ADL_BUS_CMD_SET_ADD_SIZE, &AddSize );
// Write 5 bytes set to '0' on the SPI & I2C bus
wm_memset ( WriteBuffer, WRITE_SIZE, 0 );
adl_busWrite ( MySPIHandle, &AccessConfig, WRITE_SIZE, WriteBuffer );
// Read 3 bytes from the SPI & I2C bus
adl_busRead ( MySPIHandle, &AccessConfig, READ_SIZE, ReadBuffer );
// Unsubscribe from subscribed BUS
adl_busUnsubscribe ( MySPIHandle );
```

- si utilizza un pin di GPIO come chip select (CS) del dispositivo SPI slave collegato;
- il CS è attivo a livello basso;
- i dati sul bus sono inviati partendo dall'MSB;
- il GPIO31 è il pin utilizzato come CS;
- il pin di LOAD dedicato allo scopo di CS non viene utilizzato;
- la configurazione è bifilare (clock e dati);
- la WCPU fa da master;
- il segnale di BUSY non viene utilizzato.

A seguire vengono definite le dimensioni del buffer di scrittura e di quello di lettura. Viene quindi dichiarata una struttura *AccessConfig* di tipo **adl_busAccess** che riporta i parametri di configurazione per l'accesso al bus. Questi parametri sono *Opcode* e *Address*. L'indirizzo (Address) fino a 32 bit, viene inviato nel bus prima di avviare la lettura o scrittura dei dati. L'*Opcode* è un altro parametro (max 32bit) che viene inviato sul bus prima dell'inizio della lettura o scrittura dei dati. In questo caso vengono messi a 0 perché non utilizzati.
A seguire viene definito l'handle utile nella fase successiva di sottoscrizione della porta. Viene quindi dichiarato un buffer per la scrittura dei dati e uno per la lettura. A questo punto abbiamo dichiarato tutte le strutture dati che ci servono. La funzione successiva *MyFunction* può essere richiamata in qualsiasi parte del nostro codice, oppure può essere anche smembrata in due o più funzioni: una (normalmente richiamata nell'adl_main) per effettuare la sottoscrizione della porta, un'altra può essere richiamata per la lettura e una ancora può essere utilizzata per creare una funzione di scrittura sulla porta. Vediamo nel dettaglio il codice della MyFunction:
con la funzione **adl_busSubscribe** sottoscriviamo la porta con le configurazioni dichiarate nella struttura *MYSPIConfig* precedentemente inizializzata. Con la funzione **adl_busIOCtl** definiamo la lunghezza dell'indirizzo della porta; in questo caso è zero. Questa funzione è utile se si utilizza la porta parallela (disponibile solo nel WMP100) per l'uso della quale si rimanda al punto [1] della bibliografia. A seguire, con la funzione della libreria standard "wm_stdio.h" **wm_memset**, si inizializza il buffer di scrittura a zero e con la funzione **adl_busWrite** si scrive sul buffer un numero di byte pari a quanto definito in WRITE_SIZE (cioè 5 byte).
Analogamente, con la funzione **adl_busRead** si leggono un numero di byte pari a READ_SIZE. Si noti come alle due funzioni di busWrite e busRead è necessario anche passare come parametro l'handle *MySPIHandle* definito in precedenza e collegato alla struttura *MYSPIConfig* in fase di sottoscrizione del bus. Inoltre è necessario passare come parametro anche la struttura *AcessConfig*. In questo modo la funzione di read/write è in grado di capire a quale porta accedere e come farlo.
L'ultima funzione **adl_busUnsubscribe**, come si intuisce dal nome, serve per de-sottoscrivere la porta, cioè chiude la porta quando questa non deve più essere utilizzata.

Le funzioni che abbiamo visto in questo esempio sono le stesse, che si tratti sia di un bus SPI, sia di un I²C oppure di un bus parallelo (che qui non verrà descritto).
Per lo studio della porta **I²C** proponiamo un esempio più significativo, utile per gestire una memoria EEPROM I²C standard. Il codice dell'esempio si può trovare nelle applicazioni fornite con l'ambiente di sviluppo M2M Studio. Il progetto si chiama "External-Storage_IIC" (vedi **Tabella 1**) e lo si carica allo stesso modo del progetto predefinito "HelloWorld" che abbiamo utilizzato nella prima puntata del corso. Il collegamento con il modulo Augusto (in versione WMP100) di Shitek Technology e la memoria EEPROM avviene attraverso le linee SCL(GPIO26) e SDA(GPIO27). Occorre poi predisporre una resistenza di pull-up da 4,7 kohm su entrambe le linee del bus. La velocità del bus, come si vede dal codice, è configurabile inizializzando una struttura di tipo **adl_busI2CSettrings_t** con una delle seguenti costanti predefinite:

ADL_BUS_I2C_CLK_STD	100kHz
ADL_BUS_I2C_CLK_FAST	400kHz
ADL_BUS_I2C_CLK_HIGH	3,4MHz

Nell'esempio è impostata a 400 kHz. L'esempio di Wavecom definisce tre nuovi comandi AT: AT+WRITE e AT+READ per scrivere e leggere dati dalla memoria EEPROM ed il comando AT+SETMODE per impostare la modalità asincrona o sincrona. In modalità asincrona è la WCPU che fa da master e genera il clock per la periferica I²C collegata; in modalità sincrona il segnale di clock è generato dalla periferica e la WCPU fa da slave.
Le modalità per definire nuovi comandi AT le abbiamo già viste nella scorsa puntata.
Invitiamo il lettore ad effettuare i necessari collegamenti con la memoria, prendere visione dell'esempio e testarne le funzionalità. Il codice è ben commentato ed è facile riconoscere al suo interno le stesse funzioni introdotte nel codice del **Listato 5**, solo che questa volta sono utilizzate per gestire un bus di tipo I²C.

CONCLUSIONI

In questo articolo abbiamo introdotto solo alcune delle periferiche hardware che sono a disposizione della WCPU, abbiamo trascurato ad esempio, l'**RTC** (Real Time Clock) e l'**ADC** (analog to digital interface) che peraltro, sono molto semplici da configurare, abbiamo trascurato i servizi audio come la generazione dei toni DTMF e la gestione dei canali audio. Invitiamo il lettore interessato a questi argomenti a consultare il manuale d'uso Open AT indicato in bibliografia al punto [1] e gli esempi relativi all'argomento riportati in **Tabella 1** che sono inclusi con l'M2M Studio.
Infine, per coloro che hanno già dimestichezza con i sistemi operativi per sistemi embedded, segnaliamo che le potenzialità del sistema operativo Open AT sono molto superiori a quanto può apparire dalla descrizione fin qui fatta. Con i servizi a basso livello del sistema operativo si possono gestire eventi in modalità real-time con le stesse caratteristiche che si hanno tipicamente a disposizione nei sistemi operativi utilizzati con i microcontrollori della famiglia ARM, ad esempio Linux Embedded, ma in modo molto più semplice e immediato. Questi servizi comprendono la gestione dei task, dei semafori, del passaggio dei contesti, delle priorità degli interrupt, ecc. La descrizione di queste strutture, tipiche di un sistema operativo real-time, si trova nel testo [1] della bibliografia ed esula dagli scopi di questo corso; sarà eventualmente, l'oggetto di successivi approfondimenti. Nella prossima puntata del corso verrà descritto l'utilizzo di un'altra periferica molto importante: la memoria, senza la quale le wireless CPU serie WMP non potrebbero funzionare, essendone prive. Il processore ARM montato a bordo della WCPU infatti, non ha la memoria flash né RAM integrata; occorre quindi predisporre una memoria esterna collegata al bus dati e bus indirizzi del processore. Il modulo Augusto di Shitek Technology monta già una memoria di tipo combo (flash più RAM insieme). Vedremo come gestirla nella prossima puntata.

Bibliografia:

[1] "ADL User Guide for Open AT OS v6.1.pdf" Documento allegato all'ambiente di sviluppo M2M Studio.
[2] "AT_Command_Interface_Guide_Open_AT_Firmware_7.4.pdf" Guida ai comandi AT scaricabile dal sito Wavecom *www.wavecom.com.*
[3] "MAN_000016_eng_(AUGUSTO)_ed1.1.pdf" Datasheet modulo Augusto scaricabile dal sito *www.shitek.eu*
[4] "WMP100_PTS_and_CDG_July_2007.pdf" Datasheet Wavecom WMP100 scaricabile dal sito *www.wavecom.com*

Corso Wireless CPU

La gestione delle periferiche

Conosciamo ed impariamo ad utilizzare una periferica fondamentale per il funzionamento della wireless CPU: la memoria. Settima puntata.

dell'Ing.
NICOLA FRISO

In questa puntata del corso presentiamo una periferica fondamentale per il funzionamento della wireless cpu: la memoria. Come abbiamo già avuto modo di dire nelle precedenti puntate del corso, le WCPU serie WMP50/WMP100 di Wavecom/Sierra Wireless sono delle CPU ARM9 di tipo ROMless quindi prive di memoria; sono dotate di un bus dati a 16 bit (linee D0÷D15) e di un bus indirizzi a 24bit (linee A0÷A24) con tutti i classici segnali di controllo WE,OE,WAIT,CS. La **Fig. 1** rappresenta i collegamenti tra la Wireless CPU montata sul modulo Augusto di Shitek Technology e una memoria di tipo combo (RAM + Flash) da 4 Mbyte di Flash e 2 MByte di RAM. La memoria montata sul modulo Augusto è della Nyumonix (M36W0R5040T8ZAQE/F). In alternativa, esistono memorie combo di taglia superiore: ad esempio 8 MB Flash/2MB RAM (Nymonix cod. M36W0R6040T3ZAQF) che sono pin-to-pin compatibili con le precedenti. Si può comunque scegliere di utilizzare le classiche memorie RAM e Flash separate, ovviamente utilizzando maggio-

Tabella 1 - *Suddivisione della memoria Flash.*

Flash totale	ROM (Application Code)	A&D (Application & Data Storage)	Flash Objects Data
32Mb (4MB)	200kB (min) 1600kB (max) (default: 832 kB)	0 (min) 1344kB (max) (default: 768kB)	128 kB
>=64Mb (8MB)	256kB (min) (1600+X)kB (max) (default: 832+X kB)	0 (min) (1344+X) kB (max) (default:768 kB)	128 kB

re spazio sul circuito stampato. Un esempio di collegamento con memorie di questo tipo è riportato nel datasheet Wavecom (vedere punto [1] in bibliografia).
Nello sviluppo della vostra applicazione dovete tenere conto che non tutta la memoria Flash e RAM è a disposizione della applicazione: considerando, ad esempio, la memoria combo montata sul modulo Augusto 4MB/2MB, l'ammontare di spazio realmente disponibile è 3MB per la Flash e 1,5MB per la RAM. Il resto della memoria è occupata dal sistema operativo Open AT. La memoria Flash a sua volta è suddivisa dal sistema operativo in tre parti distinte:

- ROM (Application code);
- A&D (Application & Data Storage volume);
- Flash Object Data.

La **Tabella 1** riporta le dimensioni minime, massime e di default per le tre parti in cui è suddivisa la memoria Flash. La tabella riporta i due casi: memoria Flash da 32 Mbit (4MB) e memoria da 64 Mbit (8MB). I valori di default indicati sono quelli proposti dal software di programmazione DWL Win in fase di programmazione del FW del sistema operativo. Con lo stesso software è possibile modificare manualmente le dimensioni di memoria dedicata al programma (Application Code) e le dimensioni della flash A&D. In alternativa, le stesse operazioni si possono eseguire tramite il comando AT+WOPEN (vedi **Tabella 2**). I valori predefiniti per la memoria da 32 Mbit (la stessa montata sul modulo Augusto) indicano che per il codice della applicazione sono realmente disponibili 832 kB mentre per la memoria A&D 768 kB. La Object Flash non è modificabile come dimensioni ed è sempre pari a 128 kB.
Nel caso 64 Mbit la quantità indicata con X è chiamato "spazio addizionale" ed è pari a X=(S-32)/8*1.024 dove S è la quantità totale di Flash. Ad esempio con S=64Mb, X è pari a 4.096 kB.
Vediamo ora nel dettaglio la descrizione delle tre aree di memoria in cui è suddivisa la Flash:

- **ROM (Application Code)**: è la memoria dedicata alla applicazione utente. Si tratta sempre di Flash (quindi è riscrivibile) anche se qui viene indicata come "ROM" (Read Only Memory). Questa area di memoria può essere cancellata completamente con il comando AT+WOPEN=4. Viene scritta attraverso il comando AT+WDWL (vedi puntate precedenti) oppure attraverso il software DWLWin oppure ancora attraverso la procedura di aggiornamento remoto DOTA che vedremo nella prossima ed ultima puntata del corso.
- **A&D (Application & Data Storage volume)**: è la memoria dedicata al salvataggio di dati oppure di file .DWL. I file con questa estensione possono essere file per l'upgrade da remoto del fw del s.o. oppure file di upgrade della nostra applicazione oppure ancora, file di configurazione .E2P della EEPROM.

Le funzioni messe a disposizione dalla libreria ADL per la gestione di questa area di memoria sono quelle elencate qui di seguito:

- adl_adSubscribe
- adl_adUnsubscribe
- adl_adWrite

Tabella 2 - *Funzionalità del comando AT+WOPEN.*

Comando	Descrizione
AT+WOPEN=0	Ferma l'applicazione
AT+WOPEN=1	Avvia l'applicazione
AT+WOPEN=2	Restituisce la versione dalla libreria Open AT
AT+WOPEN=3	Cancella la Object Flash (è permesso solo con l'applicazione in stop)
AT+WOPEN=4	Cancella l'applicazione (è permesso solo con l'applicazione in stop)
AT+WOPEN=5	Sospende l'applicazione fino a che non viene inviato AT+WOPENRES oppure viene rilevato un interrupt hardware
AT+WOPEN=6	Configura la memoria A&D (si veda [2] in bibliografica per maggiori dettagli)
AT+WOPEN=7	Richiede lo stato corrente dell'applicazione (se l'applicazione sta girando in modalità target o RTE)
AT+WOPEN=8	Configura la modalità Safe mode (si veda [2] in bibliografia per maggiori dettagli)

- adl_adInfo
- adl_adGetState
- adl_adFinalise
- adl_adDelete
- adl_adInstall
- adl_adRecompact
- adl_adGetCellList
- adl_adFormat
- adl_adEventSubscribe
- adl_adEventUnsubscribe
- adl_adGetInstallResult

Come abbiamo visto per tutte le risorse del s.o. Open AT, anche per utilizzare la A&D memory occorre sottoscrivere la risorsa; in questo caso si utilizza la funzione **adl_adSubscribe** per ottenere l'handle alla cella di memoria (CellHandle). Il prototipo di questa funzione è il seguente:

s32 adl_adEventSubscribe (adl_adEventHdlr_f Handler);

Il valore di ritorno di questa funzione è un numero identificativo chiamato *CellHandle* mentre la funzione *Handler* è definita come "call-back function", ed è la funzione cui salta l'applicazione quando si verifica un evento legato alla A&D memory. Gli eventi devono anch'essi essere sottoscritti con l'apposita funzione **adl_adEventSubscribe**. La **Tabella 3** riporta l'elenco dei possibili eventi. La funzione **adl_adUnsubscribe** invece, serve per de-sottoscrivere la cella di memoria.
Le informazioni circa la dimensione e lo stato della memoria A&D si possono ricavare con le funzioni **adl_adInfo** e **adl_adGetState** rispettivamente. La funzione **adl_adFinalise** serve per finalizzare la cella di memoria, il che significa fissarne le dimensioni e proteggerla rendendola accessibile in sola lettura.
La funzione **adl_adDelete** cancella la cella. La funzione **adl_adInstall** viene utilizzata quando si vuole installare un file .dwl, che può essere, ad esempio, una applicazione scaricata da remoto attraverso una connessione FTP. La funzione di installazione, al termine, resetta la WCPU, viene quindi passato al main un parametro ADL_INIT_DOWNLOAD_SUCCESS che permette di capire all'avvio dell'applicazione che il nuovo file si è installato correttamente. La funzione **adl_adRecompact** serve per ricompattare la Flash cancellando definitivamente le celle che erano state cancellate o de-sottoscritte. La funzione **adl_adGetCellList** ritorna la lista di tutte le celle sottoscritte.
La funzione che effettivamente scrive nella memoria Flash A&D è la **adl_adWrite** e ha il seguente prototipo:

*s32 adl_adWrite (s32 CellHandle, u32 Size, void * Data)*

I parametri di questa funzione sono:
- *CellHandle*: handle ottenuto in fase di sottoscrizione della cella.
- *Size:* dimensione dei dati da scrivere in byte.
- *Data*: puntatore al buffer contenente i dati.

In generale, il procedimento da seguire è il seguente: si sottoscrive la cella con la funzione **adl_adSubscribe** e si ottiene il *CellHandle*, si formatta la cella e al termine della formattazione si rileva l'evento ADL_AD_EVENT_FORMAT_DONE, si scrivono i dati sulla cella attraverso la funzione **adl_adWrite**, se la cella contiene un file .dwl lo si installa con la funzione **adl_adInstall**, al riavvio si rileva l'evento ADL_INIT_DOWNLOAD_SUCCESS che ci assicura che tutto è andato a buon fine.
Per la descrizione dettagliata di queste funzioni si rimanda alla ADL_User_Guide al

Tabella 3 - *Eventi generati dai servizi A&D*

Evento	Significato
ADL_AD_EVENT_FORMAT_INIT	L'evento è generato all'avvio della funzione adl_adFormat, indica che è stata richiesta la formattazione della memoria.
ADL_AD_EVENT_FORMAT_PROGRESS	Processo di formattazione in corso. L'evento viene generato diverse volte nel corso della formattazione.
ADL_AD_EVENT_FORMAT_DONE	Processo di formattazione terminato. Da questo momento tutta l'area A&D è stata cancellata ed è di nuovo disponibile per la scrittura.
ADL_AD_EVENT_RECOMPACT_INIT	L'evento è generato all'avvio della funzione adl_adRecompact, indica che è stata richiesta la ricompattazione della memoria.
ADL_AD_EVENT_RECOMPACT_PROGRESS	Processo di ricompattazione in corso. L'evento viene generato diverse volte nel corso della ricompattazione.
ADL_AD_EVENT_RECOMPACT_DONE	Processo di ricompattazione terminato. Da questo momento tutta l'area A&D è stata ricompattata e tutte le celle segnate come "erased" sono state effettivamente cancellate.
ADL_AD_EVENT_INSTALL	L'evento si genera quando la funzione adl_adInstall viene richiamata dall'applicazione.

Listato 1

```
// TEST MEMORIA OBJECT FLASH
// File: test_memoria.c
/*****************************************************************************/
/*  Include files                                                            */
/*****************************************************************************/
#include “adl_global.h”
#include “test_memoria.h”
/*****************************************************************************/
/*  Mandatory variables                                                      */
/*---------------------------------------------------------------------------*/
/*  wm_apmCustomStack                                                        */
/*  wm_apmCustomStackSize                                                    */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
u32 wm_apmCustomStack [ 256 ];
const u16 wm_apmCustomStackSize = sizeof ( wm_apmCustomStack );

/*****************************************************************************/
/*  Local variables                                                          */
/*****************************************************************************/
ascii *data_handle = “datahandle”;                    // handle for data
t_data_name DataName[MAX_DATA_NAME];
t_data_work DataWork[MAX_DATA_WORK];
/*****************************************************************************/
/*  Function   : SetDataName                                                 */
/*****************************************************************************/
void SetDataName(u8 indice,ascii *id,ascii *nome,ascii *cognome)
{
         wm_strcpy(DataName[indice].id,id);
         wm_strcpy(DataName[indice].nome,nome);
         wm_strcpy(DataName[indice].cognome,cognome);
}

/*****************************************************************************/
/*  Function   : SetDataWork                                                 */
/*****************************************************************************/
void SetDataWork (u8 indice, ascii *id,ascii *impiego, ascii *hh_start, ascii *hh_stop)
{
         wm_strcpy(DataWork[indice].id,id);
         wm_strcpy(DataWork[indice].impiego, impiego);
         wm_strcpy(DataWork[indice].hh_start, hh_start);
         wm_strcpy(DataWork[indice].hh_stop, hh_stop);
}

/*****************************************************************************/
/*  Function   :adl_main                                                     */
/*****************************************************************************/
void adl_main( adl_InitType_e InitType )
{
         s8 res;
         u8 i;
         ascii buffer[255];
         // Inizializza le strutture dati
         wm_memset(&DataName,0x00,sizeof(DataName));
         wm_memset(&DataWork,0x00,sizeof(DataWork));
         // Subscribe object in flash memory
         res = adl_flhSubscribe(data_handle,2);        // sottoscrive la object data flash con 2 ID.
         TRACE((1,”Creazione handle object data=%d”,res));
         if(res == ADL_RET_ERR_ALREADY_SUBSCRIBED)
         {
                  TRACE((1,”Oggetto in flash già presente\n”));
                  res = adl_flhRead(data_handle,0,sizeof(DataName),&DataName);     // Legge obj ID 0
                  TRACE((1,”Lettura memoria adl_flhRead =%d”,res));
                  // Stampa sulla seriale il contenuto dell’object flash DataName
                  adl_atSendResponse (ADL_AT_RSP,”DataName:\n\r”);
                  for(i=0;i< MAX_DATA_NAME;i++)
                  {
                           wm_memset(buffer,0x00,sizeof(buffer));
                           wm_sprintf (buffer,”%s %s %s\n\r”,&DataName[i].id,&DataName[i].nome,&DataName[i].cognome);
                           adl_atSendResponse (ADL_AT_RSP,buffer);
                  }
                  res = adl_flhRead(data_handle,1,sizeof(DataWork),&DataWork);     // Legge obj ID 1
                  TRACE((1,”Lettura memoria adl_flhRead =%d”,res));
```

capitolo “Application & Data Storage service” (punto [1] della bibliografia). Per un esempio pratico di utilizzo delle funzioni di gestione dei servizi A&D rimandiamo alla prossima puntata nella quale descriveremo l’applicazione Wavecom DOTA (Download Over The Air). Il DOTA viene utilizzato per aggiornare da remoto, attraverso un collegamento dati GPRS, il firmware della propria applicazione. In quella sede avremo modo di studiare il codice che fa uso delle funzioni sopra descritte e che gestisce tutto il processo di aggiornamen-

```
                // Stampa sulla seriale il contenuto dell'object flash DataWork
                adl_atSendResponse (ADL_AT_RSP,"DataWork:\n\r");

                for(i=0;i< MAX_DATA_WORK;i++)
                {
                        wm_memset(buffer,0x00,sizeof(buffer));
                        wm_sprintf (buffer,"%s %s %s %s\n\r",DataWork[i].id, DataWork [i].impiego, DataWork[i].hh_start, DataWork [i].hh_stop);
                        adl_atSendResponse (ADL_AT_RSP,buffer);
                }
        }
        else if(res == ADL_RET_ERR_PARAM || res == ADL_FLH_RET_ERR_NO_ENOUGH_IDS)
        {
                TRACE((1,"Errore sottoscrizione ID memoria"));
        }
        else if (res == OK)  // Nuova sottoscrizione riuscita
        {
                // Popola il database DataName
                SetDataName(0,"2454","Nicola","Friso");
                SetDataName (1,"2223","Mario","Rossi");
                SetDataName (2,"3454","Federico","Tosato");
                SetDataName (3,"2234","Marco","Bianchi");
                SetDataName (4,"0102","Francesco","Trota");
                // Popola il database DataWork
                SetDataWork(0,"2454","Progettista","9.00","17.30");
                SetDataWork (1,"2223","Impiegato amministrativo","9.00","17.30");
                SetDataWork (2,"0102","Impiegato amministrativo","9.00","17.30");
                SetDataWork (3,"2234","Tecnico specializzato","8.00","17.00");
                SetDataWork (4,"3454","Progettista","9.00","18.00");
                // Scrive un nuovo oggetto in object flash
                TRACE((1,"Creo nuovo oggetto in flash: DataName\n"));
                res = adl_flhWrite(data_handle,0,sizeof(DataName),&DataName);
                TRACE((1,"Creo nuovo oggetto in flash: DataWork\n"));
                res = adl_flhWrite(data_handle,1,sizeof(DataWork),&DataWork);
        }
}
```

```
// File: test_memoria.h
#ifndef __drvMEM_H__
#define __drvMEM_H__
/*************/
/* Constants */
/*************/
#define MAX_DATA_NAME 10
#define MAX_DATA_WORK 10
/*********/
/* Types */
/*********/
typedef struct  {
        ascii id[10];
        ascii nome[30];
        ascii cognome[30];
}t_data_name;

typedef struct  {
        ascii id[10];
        ascii impiego[30];
        ascii hh_start[6];
        ascii hh_stop[6];
}t_data_work;
/***************/
/* Variables  */
/***************/
extern t_data_name DataName[MAX_DATA_NAME];
extern t_data_work DataWork[MAX_DATA_WORK];
/***************/
/* Prototypes */
/***************/
void SetDataName(u8 indice,ascii *id,ascii *nome,ascii *cognome);
void SetDataWork(u8 indice, ascii *id,ascii *impiego, ascii *hh_start, ascii *hh_stop);
#endif
```

to. **Object Data**: questa è un'area di memoria speciale di dimensione 128kB completamente a disposizione della nostra applicazione adibita al salvataggio di dati. E' possibile definire fino ad un massimo di 2000 *object data* alla stesso tempo. Ad ogni object data viene assegnato un ID in fase di sottoscrizione. Le principali funzioni di libreria ADL adibite alla gestione di questa area di memoria sono di seguito elencate.

- adl_flhSubscribe
- adl_flhExist

- adl_flhWrite
- adl_flhRead
- adl_flhGetFreeMem
- adl_flhGetIDCount
- adl_flhGetUsedSize

Al solito la funzione **adl_flhSubscribe** è la prima ad essere richiamata e serve per ottenere l'handle al servizio. Ogni *Flash subscribe* sottoscrive un certo numero di oggetti in Flash definito dal parametro *NbPbjectRes* e identificati da un ID che va da *0* a *NbPbjectRes-1*. Il prototipo della funzione è il seguente:

s8 adl_flhSubscribe (ascii Handle, u16 NbObjectsRes)*

Ecco i valori di ritorno di questa funzione:

- OK in caso di successo;
- ADL_RET_ERR_PARAM se sono stati inseriti parametri errati;
- ADL_RET_ERR_ALREADY_SUBSCRIBED se l'handle è già stato sottoscritto;
- ADL_FLH_RET_ERR_NO_ENOUGH_IDS se non ci sono abbastanza ID disponibili per allocare l'handle;
- ADL_RET_ERR_SERVICE_LOCKED se la funzione è chiamata all'interno di un interrupt a basso livello (è proibita la chiamata all'interno di questo contesto).

Attraverso la funzione **adl_flhExist** si determina se l'handle e l'ID passati come parametro esistono già in Flash. La funzione ritorna OK se l'oggetto non esiste altrimenti ritorna le dimensioni in byte dell'oggetto o altri codici di errore nel caso di parametri errati o ID fuori dal range ammesso. La funzione **adl_flhWrite** serve per scrivere dati associati ad un determinato ID ed Handle passati come parametro. Gli altri parametri sono il puntatore al buffer contenente i dati da scrivere e le dimensioni in byte del blocco di dati. Il prototipo della funzione è il seguente:

s8 adl_flhWrite (ascii Handle, u16 ID, u16 Len, u8 *WriteData)*

I valori di ritorno sono:

- OK in caso di successo;
- ADL_RET_ERR_PARAM in caso di parametri errati;
- ADL_RET_ERR_UNKNOWN_HDL se l'handle non è stato sottoscritto;
- ADL_FLH_RET_ERR_ID_OUT_OF_RANGE se l'ID è fuori dal range definito in fase di sottoscrizione dell'handle;
- ADL_RET_ERR_FATAL in caso di fatal error (si genera anche l'evento ADL_ERR_FLH_WRITE).
- ADL_FLH_RET_ERR_MEM_FULL se la memoria *object Flash* è piena;
- ADL_FLH_RET_ERR_NO_ENOUGH_IDS se l'oggetto non può essere creato poiché non vi sono più ID disponibili;
- ADL_RET_ERR_SERVICE_LOCKED se la funzione è chiamata all'interno di un servizio di interrupt a basso livello (è proibita la chiamata all'interno di questo contesto).

La funzione **adl_flhRead** serve per leggere il contenuto della memoria, il suo prototipo, riportato di seguito, è molto simile alla funzione di write; il parametro ReadData è il buffer dove vengono copiati i dati letti dalla memoria.

s8 adl_flhRead (ascii Handle, u16 ID, u16 Len, u8 *ReadData)*

I valori di ritorno sono analoghi a quelli definiti per la funzione di write.
La funzione **flhGetFreeMem** ritorna le dimensioni in byte della memoria Flash object rimasta libera. La funzione **adl_flhGetIDCount** ritorna il numero di ID sottoscritti. La funzione **adl_flhGetUsedSize** ritorna le dimensioni in byte degli ID passati come parametro e associati ad un determinato handle (viene richiesto l'ID di start e l'ID di stop, la funzione ritorna le dimensioni in byte del totale di Flash occupata da questi oggetti).
Il **Listato 1** riporta un esempio di codice Open AT per sottoscrivere un *Object data*, scrivere al suo interno e leggere i dati in esso contenuti. Il codice di esempio definisce una struttura dati che viene chiamata *DataName* (in questo caso una scheda con un codice identificativo seguito dal nome e cognome di una persona); un'altra struttura dati chiamata *DataWork* riporta il codice identificativo, la mansione e l'orario di lavoro della persona. Il codice sottoscrive due *object data ID* con la funzione **adl_flhSubscribe** che ci servirà poi per memorizzare in Flash le due strutture dati. L'applicazione può modificare la struttura dati in RAM *DataName* e *DataWork*, poi con la funzione **adl_flhWrite** scrive i dati sull'*object data*, mentre tramite la funzione **adl_flhRead** viene letta la struttura. L'area di memoria *Object Flash* può quindi

essere utilizzata come una memoria dati ad oggetti ove salvare ad esempio delle particolari configurazioni della nostra applicazione o delle strutture di tipo database, o altro ancora. E' il codice della libreria ADL che si occupa di gestire a basso livello la cancellazione e ri-scrittura dei settori in cui è suddivisa la memoria Flash. Ricordiamo infatti, che si tratta pur sempre di Flash che ha dei limiti fisici sul numero di cancellazioni e ri-scritture possibili. La memoria può essere scritta e cancellata fino ad un massimo di circa 100.000 volte. La nostra applicazione deve tener conto di questo limite. L'area di memoria *Object Data* può essere cancellata per intero con il comando AT+WOPEN=3. Un'ultima nota riguarda la gestione della *Flash object* in modalità di debug della propria applicazione (modo RTE – Real Time Emulator). Occorre tener conto che in questa modalità come memoria *Object Data* non viene utilizzata fisicamente la memoria Flash collegata alla WCPU, ma la memoria viene emulata sul PC dove sta girando l'ambiente di sviluppo e debug M2M Studio (vedi la prima puntata del corso). In particolare il contenuto della memoria si trova nella sotto-cartella di progetto "rte\objects". Cancellando i file con estensione .obc presenti all'interno di questa cartella, si cancella il contenuto della memoria *object data* in fase di debug RTE. La prossima puntata del corso presenta il servizio Wavecom DOTA (*Download Over The Air*) che è di fondamentale importanza qualora si intenda fornire la propria applicazione della funzionalità di aggiornamento da remoto, trattandosi di un codice di programma che gira su una wireless CPU è quasi d'obbligo dotare la propria applicazione di questa modalità di funzionamento.

Bibliografia:

[1] "ADL User Guide for Open AT OS v6.1.pdf" Documento allegato all'ambiente di sviluppo M2M Studio.
[2] "AT_Command_Interface_Guide_Open_AT_Firmware_7.4.pdf" Guida ai comandi AT scaricabile dal sito Wavecom *www.wavecom.com.*
[3] "MAN_000016_eng_(AUGUSTO)_ed1.1.pdf" Datasheet modulo Augusto scaricabile dal sito *www.shitek.eu*
[4] "WMP100_PTS_and_CDG_July_2007.pdf" Datasheet Wavecom WMP100 scaricabile dal sito *www.wavecom.com*

Corso Wireless CPU

Connettersi con il DOTA

In questa puntata conclusiva spieghiamo l'applicativo DOTA (Download Over The Air) che ci permette di aprire una connessione dati GPRS e scaricare con protocollo FTP, ad esempio, un aggiornamento FW del nostro programma.

dell'Ing.
NICOLA FRISO

È noto che un programma installato in un hardware diverso da un PC è difficilmente aggiornabile; ad esempio, avete mai aggiornato il firmware di un lettore DVD o di un cellulare? Servono cavi, un computer e appositi software. Eppure non è infrequente dover aggiornare il firmware di un dispositivo gestito da microcontrollore o microprocessore, perché molto spesso, soprattutto nelle prime release, è facile che ci siano errori (bachi) e non è da escludere che il costruttore se ne accorga solo dopo il rilascio del prodotto, magari dopo la segnalazione del cliente. Perciò i dispositivi devono prevedere la possibilità di aggiornare il firmware; questa operazione, che si può compiere mediante porte di comunicazione o, più grossolanamente, sostituendo una EEPROM, nelle Wireless CPU può essere effettuata da remoto, quindi senza collegare alcun cavo. Le Wireless CPU Wavecom-Sierra Wireless serie WMP, ma anche tutte le altre WCPU della famiglia Wavecom-Sierra Wireless, dispongono di una applicazione chiamata DOTA (Download Over The Air) che può essere facilmente integrata all'interno delle nostre applicazioni. Questo

Tabella 1 - *Parametri APN dei principali operatori italiani.*

OPERATORE	APN	USER NAME	PASSWORD
VODAFONE	web.omnitel.it	[vuoto]	[vuoto]
TIM	ibox.tim.it	[vuoto]	[vuoto]
WIND	Internet.wind	[vuoto]	[vuoto]

codice ci permette di aprire una connessione dati GPRS e con protocollo FTP e scaricare un aggiornamento FW del nostro sistema basato su Wireless CPU. Nella scorsa puntata abbiamo spiegato come gestire la memoria Flash della wireless CPU, la quale, come ricorderete, è divisa in tre parti: memoria di programma, memoria dati A&D (application and data storage) e *object flash*, che è una memoria dati ad oggetti già descritta. La memoria A&D è quella che useremo per salvare e installare il firmware aggiornato. Sempre nella puntata precedente, abbiamo descritto le funzioni riguardanti la flash A&D , tralasciando volutamente gli esempi di codice; ciò perché tutte le funzioni cui si è accennato vengono utilizzate dall'applicazione DOTA e saranno oggetto di questa puntata del corso. Utilizziamo quindi questa applicazione come codice di esempio dei servizi A&D. Lo studio dell'applicazione DOTA rappresenta anche un ottimo ripasso di buona parte delle funzionalità illustrate nelle precedenti puntate del corso; in particolare, rivedremo come si sottoscrive un nuovo comando AT, come si gestiscono gli SMS in arrivo e in partenza, come si gestisce la memoria flash A&D ed in ultima analisi vedremo anche un utilizzo pratico delle funzioni di libreria di Wavecom "wip.h" (Wavecom Internet Protocol) per effettuare una connessione GPRS e per scaricare un file da un server FTP. Prima di analizzare il codice sorgente dell'applicazione conviene provarla e vedere come funziona. Al solito lanciamo M2MStudio e creiamo un progetto "Test_DOTA" sulla base dell'esempio DOTA (i vari passi per effettuare tutto ciò sono descritti nella prima puntata del corso). All'avvio dell'applicazione, attivando le TRACE appare la serie di stringhe sotto riportate:

Listato 1 Funzione "appATCmdhdl".

```
void appATCmdhdl(adl_atCmdPreParser_t *param )
{
  switch ( param->Type )
  {
    case ADL_CMD_TYPE_PARA :
      {
        // Get parameter 0
        ascii * P1 = ADL_GET_PARAM ( param, 0 );
        // Get parameter 2
        ascii * P2 = ADL_GET_PARAM ( param, 1 );
        // Get parameter 3
        ascii * P3 = ADL_GET_PARAM ( param, 2 );
        // Configure the GPRS parameters
        DOTA_GprsConfigGprsParams(P1, P2, P3);

        if (appbConnStarted)
        {
          // Clean up the previous connection
          DOTA_SmsDeInit();
          DOTA_GprsStopConnection();
        }

        // Start the GPRS connection
        DOTA_GprsStartConnection();

        // Initialise the SMS service
        DOTA_SmsInit();
       // Set the flag to allow AT+APNPARAMS to restart the
connection procedure
        appbConnStarted = TRUE;
        adl_atSendResponse(ADL_AT_RSP,"\r\nOK\r\n");
         adl_atSendResponse(ADL_AT_RSP,"\r\nWaiting for SMS
for FTP Parameters\r\n");
        break;
      }
  }
}
```

Embedded Application : ADL_INIT_POWER_ON
OAT DOTA Demo Application - 1 - InitType = %d
Hello_World from Open AT
Please Wait for +WIND: 4

l'ultima delle quali indica che l'applicazione è in attesa dell'evento WIND 4 che si scatena quando il modulo GSM si registra nella rete. Una volta registrato, occorre, attraverso un comando AT appositamente creato, definire i parametri di configurazione della connessione GPRS, la cui sintassi è la seguente:

AT+APNPARAMS="apn","user","pass"

I parametri per la configurazione riguardante l'uso con i principali operatori italiani sono riportati nella **Tabella 1**.
Fatto questo, per avviare il download del nuovo firmware occorre ricevere un SMS con la seguente sintassi:

FTP:SERV=80.xxx.xxx.xxx;FILE=arm_Goodbye_World_H.wpb.dwl;PATH=./OATAPP/;USERNAME=Username;PASSWORD=Password

Quando l'applicazione riceve questo SMS, esegue il parser del messaggio per ricavare

i parametri. Fatto ciò, scarica il file indicato nel campo FILE dal server il cui indirizzo IP è indicato nel campo SERV, mentre la sottocartella del server dove si trova il file da scaricare è indicata nel campo PATH. Durante il download appare la scritta "Connection in progress" e, a seguire, compaiono dei punti di avanzamento sullo schermo. Al termine del download viene inviato un SMS contenente il testo "DOWNLOAD SUCCESS" verso il numero di telefono che ha inviato la stringa di configurazione FTP. Per esigenze di spazio, non riportiamo per intero il codice sorgente del DOTA; lo trovate comunque all'interno degli esempi disponibili in M2MStudio, in una delle finestre del wizard di creazione del progetto, selezionando, dal menu a tendina, la voce "DOTA". L'applicativo DOTA è composto da cinque file (.c) e altrettanti file di definizioni associati (.h). Ognuno rappresenta un modulo firmware che gestisce una determinata funzione.

- **appli.c** contiene le funzioni di main, di gestione dei comandi AT sottoscritti **DOTA_appATCmdhdl** e di gestione degli SMS (**appSimHandler**) le quali, a loro volta, in risposta a determinati eventi, richiamano altre funzioni presenti negli altri moduli.
- **DOTA_AnD.c**, che sono funzioni di gestione della memoria flash A&D.
- **DOTA_Bearer_Gprs.c**, che sono funzioni di gestione della connessione GPRS.
- **DOTA_Ftp.c**; gestisce la connessione FTP per il download remoto dell'applicazione.
- **DOTA_Sms.c**; gestisce gli SMS inviati e ricevuti.

Listato 2 - Funzione "DOTA_GprsStartConnection".

```
s8 DOTA_GprsStartConnection()
{
  s8 sRet=0;

  /* Check if the GPRS is configured */
   if ( !DOTA_GprsbConfigured ) return DOTA_GPRS_ERR_CONFIG_
NOT_DONE;

  /* Initialise the WIP library */
  sRet = wip_netInit();
   //sRet = wip_netInitOpts(WIP_NET_OPT_DEBUG_PORT, WIP_NET_
DEBUG_PORT_UART1, WIP_NET_OPT_END);

  /* Open the GPRS bearer */
  sRet |= wip_bearerOpen( &DOTA_GprsBearer, "GPRS", DOTA_Gpr-
sEvHandler, NULL);
  TRACE((1, "wip_bearerOpen: %d", sRet));

  /* Configure the GPRS bearer */
  sRet |= wip_bearerSetOpts ( DOTA_GprsBearer,
                        WIP_BOPT_GPRS_APN, DOTA_GprsApnServ,
                          WIP_BOPT_LOGIN,    DOTA_GprsApnUn,
                           WIP_BOPT_PASSWORD, DOTA_GprsApnPw,
                              WIP_BOPT_END);

  TRACE((1, "wip_bearerSetOpts: %d", sRet));

  if (sRet)
  {
    DOTA_GprsStopConnection();
   return DOTA_GPRS_ERR_CONFIG_FAILED; /* Configuration fai-
led */
  }

  /* Start the GPRS connection */
  sRet = wip_bearerStart ( DOTA_GprsBearer );

  TRACE((1, "wip_bearerStart: %d", sRet));

  if (sRet == WIP_BERR_OK_INPROGRESS)
  {
       adl_atSendResponse (ADL_AT_RSP, "\r\nConnection in
Progress\r\n");
    return OK;
  }
  if (sRet != OK)
  {
    DOTA_GprsStopConnection();
      return DOTA_GPRS_ERR_CONN_START_FAILED; /* Connection
start procedure failed */
  }

  return TRUE;
}
```

L'applicativo per gestire la connessione GPRS ed FTP utilizza le funzioni della libreria Wavecom definite nel file *wip.h*, che deve essere incluso nei suddetti file. Vediamo nel dettaglio il codice di questa applicazione, partendo dalla funzione "main". Si raccomanda di seguire i passi sotto descritti tenendo sempre sott'occhio il relativo codice sorgente, in modo da capire i vari punti in cui il flusso del programma si sposta durante l'esecuzione e non perdere così il filo della descrizione del programma. All'inizio viene controllato il parametro InitType che il sistema operativo passa alla funzione main al suo avvio. I codici predefiniti sono sufficientemente autoesplicativi. L'avvio normale prevede il codice "ADL_INIT_POWER_ON"; l'avvio dopo un download andato a buon fine prevede il codice "ADL_INIT_DOWNLOAD_SUCCESS". Gli altri due sono codici errore qualora il download precedente non fosse andato buon fine. Il parametro *Inittype* viene anche stampato sulla seriale con le righe di codice:

wm_sprintf(RspStr, "\r\nOAT DOTA Demo Application - 1 - InitType = %d\r\n", InitType);
adl_atSendResponse (ADL_AT_RSP, RspStr);
Segue la stampa su seriale del messaggio di

Listato 3 - Funzione "DOTA_FtpStartConnection ".

```
bool DOTA_FtpStartConnection(ascii *FtpServ ,ascii *FileName,
ascii * FilePath,
                             ascii *FtpUn, ascii * FtpPw )
{
   /* GPRS connection is not available, do not start the FTP
connection */
  if (!DOTA_GprsIsConnected())
    return FALSE;

  TRACE((2, "DOTA_FtpStartConnection: "));
  TRACE((2, FtpServ));
  TRACE((2, FileName));
  TRACE((2, FilePath));
  TRACE((2, FtpUn));
  TRACE((2, FtpPw));

  /* Create the FTP channel */
  DOTA_FtpChannel = wip_FTPCreateOpts(   FtpServ,
                        (wip_eventHandler_f) DOTA_FtpSession-
Handler,
                       NULL,
                       WIP_COPT_USER,       FtpUn,
                       WIP_COPT_PASSWORD,   FtpPw,
                       WIP_COPT_PASSIVE,    TRUE,
                       WIP_COPT_TYPE,       'I',
/*                     WIP_COPT_PEER_PORT,  21,*/
                       WIP_COPT_END);

  TRACE((1, "wip_FTPCreateOpts: %x", DOTA_FtpChannel));
  /* If channel not created, return an error */
  if (!DOTA_FtpChannel) return FALSE;

  /* Copy the File path and name */
  wm_strcpy(DOTA_FtpFilename, FilePath);

  if ( FilePath[wm_strlen(FilePath)] != '/')
    wm_strcat(DOTA_FtpFilename, "/");

  wm_strcat(DOTA_FtpFilename, FileName);

  TRACE((1,"FileName="));
  TRACE((1,DOTA_FtpFilename));

  return TRUE;
}
```

benvenuto "Hello World from Open *AT*". A seguire viene sottoscritto il servizio per la gestione della SIM (vedi puntate precedenti) con la funzione **adl_simSubscribe**. Quando la SIM è pronta, il flusso del programma salta alla funzione **appSimHandler**, la quale sottoscrive il nuovo comando AT+APNPARAMS e stampa sulla seriale il messaggio "Please Enter the APN parameters" e la TRACE "ADL_SIM_STATE_FULL_INIT". A questo punto il programma è in attesa dei parametri APN. Con la sintassi vista sopra, dalla console di debug dell'M2MStudio si può inviare il comando AT+APNPARAMS. Nel frattempo la funzione **Dota_AnDInit()** richiamata nel main ha sottoscritto una funzione per la ricezione degli eventi relativi alla A&D e ottenuto l'handle "DOTA_AnDHandle" al servizio; viene quindi formattata la memoria A&D con la funzione **adl_adFormat**, predisponendola a ricevere il nuovo file .dwl. Alla ricezione del comando AT+APNPARAMS, il flusso del programma salta alla funzione precedentemente sottoscritta **appATCmdhdl** riportata anche nel **Listato 1**. Qui i tre parametri APN, user e password vengono passati alla funzione **DOTA_GprsConfigGprsParams**, nella quale il flag **DOTA_GprsbConfigured** viene assegnato TRUE; vengono anche impostate le stringhe DOTA_GprsApnServ , DOTA_GprsApnUn, DOTA_GprsApnPw definite nel modulo DOTA_Bearer_Gprs.c. Si verifica poi se vi e è già una connessione in corso, nel qual caso con le funzioni **DOTA_SmsDeInit**() e **DOTA_GprsStopConnection**(), si cancella la precedente connessione. In seguito, la funzione **DOTA_GprsStartConnection**() dà inizio ad una nuova connessione GPRS. La funzione **DOTA_SmsInit** inizializza il servizio per la ricezione degli SMS attraverso la funzione:

DOTA_SmsHandle = adl_smsSubscribe (DOTA_SmsHandler, DOTA_SmsCtrlHandler, ADL_SMS_MODE_TEXT);

Dove **DOTA_SmsHandler** è la funzione a cui salta quando riceve un SMS, mentre

Listato 4 - Funzione "DOTA_FtpSessionHandler".

```
void DOTA_FtpSessionHandler( wip_event_t *ev, void *ctx)
{
    TRACE((1,"DOTA_FtpSessionHandler:  Event:  %d",  ev-
>kind));
  switch( ev->kind)
  {
    case WIP_CEV_OPEN:
      TRACE((1,"WIP_CEV_OPEN"));

      /* Start the A&D Process */
      DOTA_AnDStart();

      /* Session Established, get the file */
     DOTA_FtpFileHandle = wip_getFile ( DOTA_FtpChannel,
DOTA_FtpFilename,
                                        DOTA_FTPFileDa-
taHandler, NULL );
      break;

    case WIP_CEV_DONE:
      TRACE((1,"WIP_CEV_DONE"));
      break;

    case WIP_CEV_PEER_CLOSE:
      TRACE((1,"WIP_CEV_PEER_CLOSE"));
      break;

    case WIP_CEV_ERROR:
       TRACE((1,"WIP_CEV_ERROR %d", ev->content.error.
errnum));
      break;
  }
}
```

DOTA_SmsCtrlHandler è la funzione di call-back (di ritorno) cui salta il flusso del programma dopo che ha trasmesso un SMS; (entrambe le funzioni sono già state trattate nelle precedenti puntate). Al termine della *appATCmdhdl* viene messo TRUE il flag **appbConnStarted** e stampato il messaggio "Waiting for SMS for FTP Parameters" che indica che l'applicazione è in attesa dell'SMS con la configurazione dei parametri FTP. Torniamo adesso alla funzione **DOTA_GprsStartConnection** riportata anche nel **Listato 2**. Questa funzione richiama direttamente le librerie wip; nell'ordine viene:

- inizializzata la wip library con la funzione **wip_netInit**;
- aperta la connessione GPRS con la funzione **wip_bearerOpen**;
- configurata la connessione con la funzione **wip_bearerSetOpts** , passando i parametri APN,user e password precedentemente ricavati;

Se tutto è andato a buon fine, la funzione **wip_bearerStart** dà inizio alla connessione GPRS; il valore di ritorno di questa funzione (WIP_BERR_OK_INPROGRESS) e la stampa sulla seriale del messaggio "Connection in Progress" confermano la connessione. Con questo abbiamo terminato la descrizione della funzione **appATCmdhdl** e del modulo appli.c. L'applicazione in questo momento è in attesa del messaggio SMS di configurazione FTP. Notate che ci siamo premurati di sottoscrivere la ricezione dei messaggi, perciò quando arriva questo SMS il flusso del programma salta alla funzione **DOTA_SmsHandler**, che si trova nel modulo *DOTA_Sms.c*. Riprendiamo quindi la descrizione del codice sorgente da questo punto. Questa funzione

Listato 5 - Funzione "DOTA_FTPFileDataHandler".

```
void DOTA_FTPFileDataHandler( wip_event_t *ev, void *ctx)
{
  TRACE (( 1, "DOTA_FTPFileDataHandler Event = %d", ev->kind));

  switch( ev->kind)
  {
    case WIP_CEV_OPEN:
      break;

    case WIP_CEV_READ:
      /* Read the data from FTP */
      DOTA_FtpReadFile();
      break;

    case WIP_CEV_WRITE:
      break;

    case WIP_CEV_DONE:
      TRACE (( 1, "File WIP_CEV_DONE"));
      break;

    case WIP_CEV_PEER_CLOSE:
      TRACE (( 1, "FTP WIP_CEV_PEER_CLOSE"));
      /* Install the application */
      DOTA_AnDInstall();

      break;

    case WIP_CEV_ERROR:
      TRACE(( 1, "FTP WIP_CEV_ERROR"));
      break;
  }
}
```

Listato 6 - Funzioni "DOTA_SmsSend" e "DOTA_SmsCtrlHandler".

```
void DOTA_SmsSend(bool bSuccess, DOTA_cbSmsHandler_f app_SmsCb)
{
  ascii SmsDataBuffer[DOTA_SMS_MAX_LENGTH];
  s8 sRet;

  if (bSuccess)
    wm_strcpy(SmsDataBuffer, "DOWNLOAD SUCCESS");
  else
    wm_strcpy(SmsDataBuffer, "INSTALL FAILED");

  TRACE((2, "DOTA_SmsAgentPhoneNum:"));
  TRACE((2, CFG_FILE.DOTA_SmsAgentPhoneNum));

  /* Send SMS */
   sRet = adl_smsSend ( DOTA_SmsHandle, CFG_FILE.DOTA_SmsAgentPhoneNum,
                        SmsDataBuffer, ADL_SMS_MODE_TEXT);

  DOTA_SmsbSent = TRUE;
  DOTA_SmsCb    = app_SmsCb;
}


void DOTA_SmsCtrlHandler( u8 Event, u16 Nb )
{
  TRACE((1,"DOTA_SmsCtrlHandler: %d - %d",Event,Nb ));

  switch(Event)
  {
    case ADL_SMS_EVENT_SENDING_OK:
      if (DOTA_SmsbSent)
      {
       adl_atSendResponse(ADL_AT_RSP,"\r\nSMS Sending success\r\n");

        /* Start the installation */
        //DOTA_AnDInstall();

        DOTA_SmsbSent = FALSE;
      }
      break;

    case ADL_SMS_EVENT_SENDING_ERROR:
      TRACE (( 1, "ADL_SMS_EVENT_SENDING_ERROR"));
      break;

    default:
      break;
  }

  if (DOTA_SmsCb)
    DOTA_SmsCb (Event);
}
```

Listato 7 - Funzione "main" modificata con DOTA.

```
void adl_main( adl_InitType_e InitType )
{

  ascii RspStr[120];
  RspStr[0] = '\0';// empty string;
  TRACE (( 1, "Embedded Application : Main %d", InitType));

  switch (InitType)
  {
    case ADL_INIT_POWER_ON:
      TRACE (( 1, "Embedded Application : ADL_INIT_POWER_ON"
)); break;
    case ADL_INIT_REBOOT_FROM_EXCEPTION:
     TRACE (( 1, "Embedded Application : ADL_INIT_REBOOT_FROM_
EXCEPTION" )); break;
    case ADL_INIT_DOWNLOAD_SUCCESS:
       TRACE (( 1, "Embedded Application : ADL_INIT_DOWNLO-
AD_SUCCESS" )); break;
    case ADL_INIT_DOWNLOAD_ERROR:
       TRACE (( 1, "Embedded Application : ADL_INIT_DOWNLO-
AD_ERROR" )); break;
    default: break;
break;
  }

  wm_sprintf(RspStr, "\r\nOAT  MY DOTA ver.1.00 -  InitType =
%d\r\n", InitType);
  adl_atSendResponse (ADL_AT_RSP, RspStr);

#ifdef APP_HELLO_WORLD_SAMPLE
  adl_atSendResponse (ADL_AT_RSP, version);
#else
   adl_atSendResponse  (ADL_AT_RSP,  "\r\nGoodbye_World  from
SIMPLY \r\n");
#endif // APP_HELLO_WORLD_SAMPLE

 TRACE (( 1, "Embedded : Appli Init" ));
TRACE (( 1, "Embedded Application : Main" ));

 Init_drvMEM();
 ReadDataMem(TYP_STAT);   // read object flash




 if(!STATUS_FILE.f_Boot)    // NORMAL MODE
         {

                  adl_atUnSoSubscribe("+WIND:       4",(adl_
atUnSoHandler_t)Wind4_Handler);

                  // Initialization code
                  //.....
         }
else                                  // BOOT MODE
         {
                  // Subscribe to SIM service
                  adl_simSubscribe  (  (adl_simHdlr_f)  appSi-
mHandler, NULL);
                  // Initialise the A&D service
                  DOTA_AnDInit();
         }

}
```

esegue il parser dell'SMS e vengono stampati sulla seriale anche i vari parametri ricavati durante il parsing. Se tutto è andato a buon fine, viene effettuata una chiamata alla funzione **DOTA_FtpStartConnection** del modulo DOTA_Ftp.c, passandogli come parametri quelli ricavati dal parser. La funzione **DOTA_FtpStartConnection** richiama direttamente la funzione di libreria wip per la creazione di un canale FTP attraverso la funzione **wip_FTPCreateOpts**, cui vengono passati i parametri per la connessione Ftp (vedi **Listato 3**). La **wip_FTPCreateOpts** imposta anche una funzione di call-back ad un evento di apertura sessione FTP chiamato **DOTA_FtpSessionHandler**. Quindi all'apertura della sessione FTP il flusso del programma salta a questa funzione, riportata nel **Listato 4**, all'interno della quale la TRACE di debug stampa il messaggio "WIP_CEV_OPEN". Viene poi chiamata la funzione **DOTA_AnDStart**() che sottoscrive una cella di memoria A&D dove salvare il contenuto del file che viene scaricato via FTP. La cella ha, per il momento, dimensioni sconosciute, infatti viene sottoscritta con il parametro ADL_AD_SIZE_UNDEF. **DOTA_FtpFileHandle** è l'handle al file FTP che viene ottenuto con la funzione di libreria **wip_getFile**. La funzione di call-back **DOTA_FTPFileDataHandler** passata come parametro è quella a cui salta il flusso del programma durante lo scaricamento del contenuto del file .dwl. Il flusso del programma, se tutto sta andando per il meglio, dovrebbe entrare nella **DOTA_FTPFileDataHandler** con attivo l'evento WIP_CEV_READ (vedi **Listato 5**). Viene quindi chiamata la funzione **DOTA_FtpReadFile**(). Questa funzione legge il file a blocchi di massimo 1.500 byte alla volta richiamando la funzione di libreria **wip_read**. Ogni blocco di byte viene scritto in flash con la funzione **DOTA_AnDWriteData**. Se c'è qualche errore la scrittura viene interrotta e quanto è stato scritto fino a quel momento viene perso, ma il vecchio firmware viene comunque mantenuto in memoria poiché non si è effettuata alcuna procedura di installazione del nuovo file .dwl. Dunque, al riavvio dell'applicazione sarà come non fosse successo nulla, il parametro InitType sul main indicherà però un codice di errore. Durante il processo di scrittura in flash A&D del file, se tutto sta andando per il meglio, viene stampata la stringa ".." ad ogni blocco di byte scritto. Al termine si genera l'evento WIP_CEV_DONE nella funzione **DOTA_FTPFileDataHandler** e a seguire l'evento WIP_CEV_PEER_CLOSE. A questo punto il file è stato salvato per intero e senza

errori nella cella di flash A&D sottoscritta. Per renderlo operativo e far partire l'applicazione con il nuovo file, occorre installarlo. La funzione **DOTA_AnDInstall** si occupa di questo. Questa funzione del modulo DOTA_AnD.c richiama la funzione di libreria **ADL adl_adFinalise** (vedi anche la precedente puntata del corso) per finalizzare la cella di memoria e renderla accessibile in sola lettura. Subito dopo tramite la funzione **DOTA_SmsSend** viene inviato l'SMS verso l'utente che ha dato il via alla procedura di aggiornamento remoto, con il messaggio "DOWNLOAD SUCCESS" se tutto è andato a buon fine oppure "DOWNLOAD ERROR" se vi sono stati dei problemi. Se il messaggio viene inoltrato all'operatore telefonico con successo il flusso del programma salta alla funzione **DOTA_SmsCtrlHandler** con l'evento ADL_SMS_EVENT_SENDING_OK. Entrambe la funzioni sono riportate nel **Listato 6**.
Fino ad ora nulla ha ancora chiamato la funzione di installazione del file; questo è forse il punto più delicato di tutto il codice sorgente del DOTA. La riga di codice che richiama la funzione DOTA_SmsSend è la seguente:

DOTA_SmsSend(TRUE, DOTA_AnDSmsEventHandler);

dove **DOTA_AnDSmsEventHandler** è un evento di tipo **DOTA_cbSmsHandler_f** definito come segue nel file *DOTA_Sms.h*: *typedef void (*DOTA_cbSmsHandler_f) (u8 Event)*;, cioè un puntatore a funzione con parametro un byte di nome "Event". La funzione **DOTA_SmsSend** riceve quindi come parametro un puntatore alla funzione di call-back **DOTA_AnDSmsEventHandler**. Quest'ultima funzione però deve essere eseguita solo dopo che si è sicuri di aver inoltrato il messaggio all'operatore telefonico, cosa che avviene solo quando il flusso del programma salta automaticamente alla funzione **DOTA_SmsCtrlHandler** definita in fase di sottoscrizione degli SMS. Per poter saltare quindi alla funzione DOTA_AnDSmsEventHandler "passando" dalla DOTA_SmsSend viene definito e attivato un flag *DOTA_SmsbSent = TRUE* nella DOTA_SmsSend. La riga di codice *DOTA_SmsCb = app_SmsCb*; assegna al puntatore a funzione **DOTA_SmsCb** il parametro **app_SmsCb** che altro non è che la funzione **DOTA_AnDSmsEventHandler**. Quando l'SMS viene inoltrato alla rete, il flusso del programma salta alla funzione **DOTA_SmsCtrlHandler**. Se il flag **DOTA_SmsbSent** è true, significa che era stato inviato un SMS, quindi la funzione stampa "Sending success" e riporta falso il flag. A seguire, le righe di codice:

```
if (DOTA_SmsCb)
   DOTA_SmsCb (Event);
```

verificano se al puntare a funzione **DOTA_SmsCb** è stato assegnato un valore; se la risposta è sì, questo valore è proprio la funzione passata come parametro **DOTA_AnDSmsEventHandler** cui viene passato il parametro Event della **DOTA_SmsCtrlHandler**. In questo caso il parametro Event, se tutto è andato bene, è ADL_SMS_EVENT_SENDING_OK. Quindi è la riga di codice *DOTA_SmsCb (Event)* che sposta il flusso del programma alla funzione **DOTA_AnDSmsEventHandler**. In questa si verifica che l'evento sia proprio ADL_SMS_EVENT_SENDING_OK e quindi finalmente si richiama la funzione **adl_adInstall**, che installa il file e riavvia la WCPU con il nuovo firmware .dwl caricato. Sebbene possa apparire un po' complicato per i neofiti del linguaggio C, quello appena descritto è un metodo corretto ed elegante per eseguire un evento (installazione) a seguito di un altro evento, passando come parametro il risultato (in questo caso ADL_SMS_EVENT_SENDING_OK) dell'evento precedente. Concludiamo con una nota sull'utilizzo pratico di questo esempio DOTA fornito da Wavecom. Il codice dell'esempio affinché sia utilizzabile deve necessariamente entrare a far parte della propria applicazione.
Per far questo suggeriamo alcune modifiche molto semplici da effettuare, che vanno ad intaccare minimamente il codice DOTA e il codice della nostra applicazione (che magari è

Bibliografia:

[1] "ADL User Guide for Open AT OS v6.1.pdf" Documento allegato all'ambiente di sviluppo M2M Studio.
[2] "AT_Command_Interface_Guide_Open_AT_Firmware_7.4.pdf" Guida ai comandi AT scaricabile dal sito Wavecom *www.wavecom.com.*
[3] "MAN_000016_eng_(AUGUSTO)_ed1.1.pdf" Datasheet modulo Augusto scaricabile dal sito *www.shitek.eu*
[4] "WMP100_PTS_and_CDG_July_2007.pdf" Datasheet Wavecom WMP100 scaricabile dal sito *www.wavecom.com*

già stato scritto). In pratica si sfrutta la memoria flash object per salvare un flag chiamato *f_Boot*. All'avvio dell'applicazione, se f_Boot è TRUE si lancia l'applicazione DOTA, mentre se f_Boot è FALSE si avvia normalmente la nostra applicazione. Il **Listato 7** riporta una traccia di codice della funzione main modificata all'uopo. Per avere maggiore flessibilità conviene poi eliminare dal codice del DOTA originale il comando AT+APNPARAMS e permettere anche ai parametri GPRS di essere impostati via sms, come per la configurazione FTP.
Inoltre, poiché i parametri APN e FTP vengono solitamente inviati una sola volta conviene averli salvati da qualche parte in object flash in modo da ricaricarli ogni volta che si alimenta la WCPU e parte l'applicativo. Ovviamente è buona prassi prevedere anche dei parametri di default che vengano scritti in object flash e richiamati qualora non si riceva nessun parametro di configurazione via sms. Fatto questo è possibile aggiungere al parser degli SMS ricevuti il comando:

<password> UPADTEFW

dove <password> è una password che conosce solo chi è abilitato ad eseguire l'update da remoto. Alla ricezione del messaggio "UPDATEFW" l'applicativo non fa altro che settare a TRUE il flag f_Boot e salvarlo in flash e riavviare la WCPU. Al riavvio, trovando il flag f_Boot = TRUE viene avviata l'applicazione in modalità "DOTA", si scarica il nuovo firmware e, se tutto è andato a buon fine, si rimette f_Boot = FALSE e si invia l'SMS "DOWNLOAD SUCCESS". L'applicazione si riavvia quindi aggiornata.
Con questo abbiamo concluso il nostro corso base di programmazione delle Wireless CPU serie WMP con il sistema operativo e le librerie Open AT di Wavecom-Sierra Wireless. Attendiamo fiduciosi le vostre applicazioni Open AT, ovviamente aggiornabili da remoto!