Scuola Universitaria Professionale
della Svizzera Italiana
SUPSI
Dipartimento
Tecnologie
Innovative

Progetto di Diploma

I3-O3-06/07-DIP- C00382

Beacon CW per TISat-1

Studente: **Andrea De Maria**

Docente: **Paolo Ceppi**
per Prog. di diploma:
Relatore: **Paolo Ceppi**
Co-relatore: **Allen Weston**

Data: **14 settembre 2007**

Indice

Elenco delle figure

Ringraziamenti

Colgo l'occasione per ringraziare l'ing. Ivano Bonesana per i drivers dell'interfaccia SPI e assieme all'ing. Andrea Spiga per le informazioni sul processore MSP430.
Ringrazio anche l'ing. Stjepan Puseljic per la collaborazione e la velocità nel fornire i vari componenti necessari per il funzionamento.
Un grazie inoltre al prof. Paolo Ceppi e al prof. Allen Weston per l'aiuto e la collaborazione durante tutto il progetto.

Beacon CW per TISat-1

Riassunto / abstract

Il beacon per il progetto TISat-1 è un componente fondamentale, al quale è affidato il compito di rendere reperibile e identificabile il satellite e di dare informazioni base sullo stato degli apparati di bordo.
Il contatto con il beacon confermerà che le operazioni previste durante la fase di entrata in orbita sono state eseguite con successo.
Questo progetto offre delle ottime soluzioni per quanto riguarda la stabilità e il risparmio energetico, due punti su cui mi sono soffermato a lungo durante l'analisi.
Il programma da me realizzato è una versione che permette di sperimentare 3 metodi di trasmissione intercambiabili durante l'esecuzione. Inoltre tramite semplici impostazioni del software sarà possibile modificare il funzionamento del beacon.
Il team SSL potrà eseguire dei test di funzionalità al fine di trovare le impostazioni che danno i migliori risultati.

The beacon for the project TISat-1 is a fundamental component which is submitted the assignment of make available and identifiable the satellite and to give information base on the state of the apparatuses on board.
The contact with the beacon will confirm that the previewed operations during the phase of entrance in orbit have been performed successfully.
This project offers some good solutions as it regards the stability and the energetic saving, two points on which I have stopped for a long time during the analysis.
The program realized by me is a version that allows to experiment 3 methods of transmission interchanging during the execution. Moreover through simple settings of the software it will be possible to modify the functionalities of the beacon.
The SSL team will be able to perform some tests of functionality with the purpose of finding the formulations that give the best results.

1. Progetto assegnato

1.1. Caratteristiche generali del progetto

Corso di laurea:	Informatica TP
Opzione del progetto:	O3 Informatica tecnica e microelettronica
Tipo di progetto:	diploma
Semestre:	Semestre Estivo

1.2. Elementi organizzativi del progetto

Codice di progetto:	C00382
Titolo del progetto:	Beacon CW per TISat-1
Stato del progetto:	scelto da studente
Confidenzialità:	Confidenziale

1.3. Elementi descrittivi del progetto

1.3.1. Descrizione

Il primo livello di connessione fra TISat-1 e la terra è garantito da un sistema semplice: un trasmettitore che emette periodicamente l'identificativo del satellite e alcune (poche) informazioni sullo stato di salute degli apparati di bordo.

1.3.2. Compiti

Con un trasmettitore (o transceiver) ad alta integrazione, realizzare il sistema richiesto.
Sono richieste valutazioni di affidabilità (punti deboli) del sistema e, se del caso proposte per realizzazioni ridondanti.
Si lavorerà con kit di sviluppo pronti.

1.3.3. Obiettivi

Beacon CW collaudato funzionante.

1.3.4. Tecnologie

Microcontrollori, software - firmware (90%)
Sensori.
Trasmissione.

1.4. Contatti del progetto

Relatore:	Ceppi Paolo
Co-relatore:	Weston Allen
Partner esterno:	SUPSI-SpaceLab ( )

2. Introduzione

SUPSI SpaceLab sta sviluppando, con studenti e collaboratori, il satellite artificiale TISat-1 del tipo CubeSat, il quale sarà messo in orbita polare attorno alla Terra ad una quota di circa 700 km.

Il progetto beacon consiste nello sviluppare un componente che invii un messaggio e delle informazioni base riguardanti lo stato del satellite (SOH, State Of Health).
Il termine beacon sta per radiofaro o radio beacon, il quale è un trasmettitore radio unidirezionale che trasmette continuamente un segnale su una specifica frequenza.
Il beacon era indispensabile soprattutto prima dell'introduzione del GPS, del LORAN e del VOR, per determinare, per mezzo di radioricevitori direzionali, la posizione rispetto ad un punto di riferimento noto, il radiofaro stesso.
Esistono differenti tipi di radiofari. In aviazione ad esempio è utilizzato il Non-directional Beacon (NDB) come guida per raggiungere gli aeroporti.
Nell'ambito dei satelliti invece questo componente ha una funzione inversa, per cui dalla terra noi sappiamo, grazie al beacon, quando il satellite è nella portata della stazione d'ascolto.

L'invio avverrà tramite OOK[1] (On – Off – Keying), che è un tipo di modulazione ASK che rappresenta i dati digitali come presenza o assenza di una portante. Nella sua forma più semplice, la presenza di una portante per un tempo specifico indica un 1 binario, mentre la sua assenza per la stessa durata temporale uno 0.

Per l'invio del messaggio identificativo verrà utilizzata la codifica Morse [2], vale a dire un sistema per trasmettere lettere, numeri e segni di punteggiatura per mezzo di un segnale in codice ad intermittenza.

Cattere	Codice	Numeri e punteggiatura	Codice
A	·—	**0**	—————
B	—···	**1**	·————
C	—·—·	**2**	··———
D	—··	**3**	···——
E	·	**4**	····—
F	··—·	**5**	·····
G	——·	**6**	—····
H	····	**7**	——···
I	··	**8**	———··
J	·———	**9**	————·
K	—·—	**.**	·—·—·—
L	·—··	**,**	—··——
M	——	**:**	———···
N	—·	**?**	··——··
O	———	**=**	—···—
P	·——·	**-**	—····—
Q	——·—	**(**	—·——·
R	·—·	**)**	—·——·—
S	···	**"**	·—··—·
T	—	**'**	·————·
U	··—	**/**	—··—·
V	···—	**@**	·——·—·
W	·——		
X	—··—		
Y	—·——		
Z	——··		

Tabella 1: Codifica Morse

Come si può vedere dalla tabella 1, vengono utilizzati solo punti, spazi e linee, le quali sono definite nel seguente modo:

1. Il punto è l'unità di tempo base del Morse.
2. La linea è 3 volte un punto.
3. Gli spazi
   3.1. tra i punti e le linee di un carattere sono della lunghezza di un punto
   3.2. tra le lettere sono ampi come una linea (3 punti)
   3.3. tra parole sono lunghi 7 punti.

La scelta di questo sistema di comunicazione (OOK) è derivato dal fatto che permette di ridurre il consumo energetico; in quanto se noi usassimo una comunicazione basata su portante e modulante avremmo di conseguenza due frequenze che dobbiamo emettere, con un consumo maggiore di energia. Questa modulazione permette inoltre di lasciare spenta la portante (consumo nullo) quando si vuole inviare uno 0.

3. Requisiti

Questi requisiti sono stati discussi durante le riunioni del progetto.

3.1. Codice identificativo tramite Morse

La legge prevede che se si vuole inviare delle informazioni tramite radio frequenza, si deve entro un certo lasso di tempo inviare un codice identificativo. Questo codice sarà assegnato a TISat-1 dagli appositi organismi di coordinamento.
Per le prove sono stato autorizzato ad usare l'identificativo personale del prof. Ceppi.
Prima di inviare qualsiasi informazione il CubeSat trasmetterà il codice identificativo.

3.2. Invio di un messaggio tramite Morse

In aggiunta al codice di identificazione si vuole inviare altre informazioni, come ad esempio il nome del satellite, della scuola, del paese d'origine, ecc. Questa parte del messaggio verrà inviata tramite Morse, per far si che ogni persona con le apparecchiature necessarie sia in grado di comprenderle.

3.3. Invio dello stato del satellite

Un altro scopo del progetto è quello di inviare lo stato delle apparecchiature di bordo, come ad esempio le temperature, le tensioni e le correnti. Questa parte può prevedere anche una codifica diversa da quella Morse.

3.4. Ridurre il consumo di energia

Si dovrà cercare di ridurre al minimo il consumo di energia, in quanto il CubeSat possiede delle risorse limitate.

3.5. Sviluppare il progetto su due kits diversi

Il team SSL ha identificato due possibili soluzioni hardware su cui sviluppare il progetto: uno basato sul processore MSP430 [3] che pilota un transceiver esterno e l'altro su un intel 8051 che si trova integrato in componenti come ad esempio, il transceiver CC1010 (Chipcon/TI) [3].

3.6. Linguaggio di programmazione

Il software deve essere sviluppato in C e per eventuali parti critiche in Assembler.

3.7. Trovare l'ambiente di sviluppo

Eseguire una ricerca per identificare il tool o i tools necessari per sviluppare il prototipo.

4. Studio delle soluzioni

4.1. Morse

Una parte del messaggio che il cubesat deve inviare verrà appunto spedito tramite codice Morse.
Questa codifica, com'è stata descritta nell'introduzione, necessita di tre caratteri: lo spazio, il punto e la linea. Per eseguire la conversione dal carattere ASCII verrà sviluppata una funzione che restituisce un vettore contenete i fattori di moltiplicazione del tempo di base. Per esempio se si richiede la codifica della lettera 'a' (in morse . -) la funzione restituisce il vettore contenente i seguenti dati:

actualCode [0] = 1; fattore per il punto
actualCode [1] = 1; fattore per lo spazio
actualCode [2] = 3; fattore per la linea

L'invio verrà coordinato tramite un interrupt dato da un timer. Il tempo è dato dalla moltiplicazione fra il fattore e il tempo di base. L'idea di utilizzare un interrupt è pensata per far in modo che il processore sia in grado di svolgere altre attività.

Visto il funzionamento del sistema OOK e del codice Morse (intermittenza) l'accensione o lo spegnimento avverranno tramite l'inversione dello stato attuale, per cui se la portante è accesa verrà spenta e viceversa.

4.1.1. Metodi d'invio

Per l'invio dei dati sono stati identificati due metodi diversi:

1. Preparazione ed invio

Questo metodo si basa sull'idea di eseguire una conversione da ASCII a Morse dell'intero messaggio e quando questa è terminata si procede con l'invio.

Vantaggi
Questo metodo permette di ridurre al minimo la funzione legata all'interrupt e quindi renderla più veloce.

Svantaggi
Dal momento che questa codifica viene eseguita a priori, si ha una mole maggiore di dati in memoria.

2. Codifica ed invio

Questa soluzione invece si basa sul concetto di creare il messaggio solo durante la procedura d'invio. In pratica la conversione di un simbolo ASCII viene codificato in Morse solo durante l'invio (on the fly).

Vantaggi
Non vengono salvate grandi quantità di dati se non quelle del simbolo che si sta' inviando.

Svantaggi
La funzione di interrupt risulta più pesante dato che dovrà occuparsi della codifica da ASCII a Morse.

Metodo scelto

Entrambi i metodi hanno caratteristiche positive e negative.
Da una stima approssimativa, si può supporre che la funzione di interrupt necessiti al massimo di 10 condizioni if e 5 chiamate a funzioni, più alcune operazioni di somma. Di fatto il secondo metodo anche se ha come lato negativo la funzione di interrupt, questo non risulterà un problema, dato che il tempo di base per la durata di un punto del codice Morse sarà piuttosto elevato (100ms).

Formula di verifica del tempo necessario per la funzione di interrupt :

$$100ms \geq 10 \cdot tIF + 5 \cdot tCall \cdot 8 \cdot tSum$$

Formula 1: Funzione di interrupt

In cui:
tIF: tempo in ms per eseguire un'operazione di if
tCall: tempo in ms per eseguire un'operazione di call
tSum: tempo in ms per eseguire un'operazione di if

Per questo motivo verrà scelto il secondo metodo, che utilizza meno memoria.

4.2. Sensori

Questa parte è suddivisa in due, una parte che si occupa di leggere i valori dei sensori e l'altra dell'invio. La lettura dei sensori non è parte del mio lavoro, ma si può supporre che il periodo di lettura dei sensori potrebbero essere diversi; ad esempio la temperatura potrebbe essere letta ogni tempo t_0 e invece lo stato delle batterie ad ogni tempo t_1 e via di seguito.
Per questa ragione al momento che si vogliono inviare i dati tramite beacon è possibile che alcuni valori siano nuovi e altri invece uguali ai valori spediti nella trasmissione precedente.
Una situazione potrebbe essere la seguente:

t_0: tutti i sensori vengono letti
t_1: il beacon invia tutti i valori letti
t_2: solo un sensore viene riletto
t_3: il beacon deve inviare i dati vecchi tranne che per il sensore letto al tempo t_2 per il quale viene inserito il valore dell'ultima misura.

Data la situazione appena esposta, il programma dovrà prevedere un salvataggio dei valori, dando in aggiunta anche la possibilità di modificarli.
Per poter dare questa funzionalità si prevede di usare un vettore contenente i valori dei sensori. Ogni sensore avrà un proprio identificativo. Utilizzando questa struttura, verrà modificato solo il sensore indicato tramite il proprio ID, senza alterare gli altri.
La lettura dei sensori non è ancora stata sviluppata, pertanto ne verrà simulata una, grazie alla quale sarà possibile testare il funzionamento del programma.

4.3. Consumo energetico

Per ridurre il consumo la CPU viene spenta quando non è necessaria. Questo risparmio è utile anche se minimo dal momento che il rapporto fra il consumo della CPU e del sistema di trasmissione è elevato (rapporto 1:10'000).
Grazie al fatto che viene usato un interrupt per l'invio dei dati, è possibile accendere il processore solo durante l'esecuzione della funzione ad esso associato.

Oltre a questa tecnica è stato utilizzato un timer che si occupa di attivare dopo un tempo *t* la lettura dei sensori oppure l'invio del messaggio. Il processore viene messo in standby fino a quando il timer non scade o fino a quando non è richiesto per eseguire altre operazioni.
Per l'invio dei dati sui sensori, ulteriormente a quanto è stato appena detto, ci si è concentrati su come sviluppare una codifica a risparmio energetico.
Il miglior modo per ridurre il consumo è quello di mantenere spenta la portante, si è pensato pertanto di sfruttare lo spegnimento legato al tempo trascorso. Il tempo per cui la portante rimane spenta è legata al valore che si vuole inviare. Grazie a questo sistema sarà necessario accendere la portante solo per indicare l'inizio e la fine del valore (definiti in seguito come start bit e stop bit).

Schema illustrativo del metodo appena spiegato:

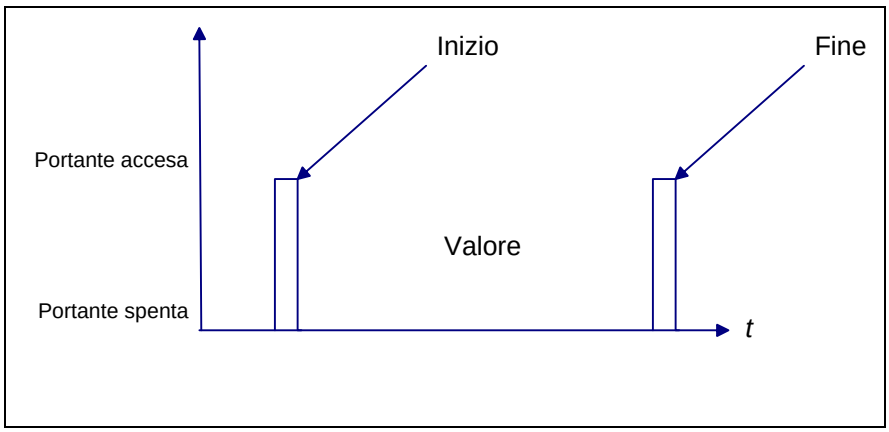

Figura 1: Invio dati dei sensori

Il valore letto dai sensori viene moltiplicato per un fattore di base uguale per tutti. Per esempio se il valore letto è di 15 e il fattore di base è di 10ms, la portante verrà spenta per 150ms. Questo valore (150ms) verrà poi inserito nel timer che gestisce l'interrupt per l'invio del codice Morse permettendo così di sfruttare la stessa funzione di IRQ per l'intera trasmissione.
Il valore che si dovrà inviare potrà essere sia positivo che negativo, per cui la distinzione avverrà cambiando la lunghezza dello start bit. Lo start bit del valore negativo sarà leggermente allungato.
Per ridurre ulteriormente il consumo di energia è possibile accorciare il tempo necessario per inviare il messaggio. Per questo motivo il tempo di base del codice Morse è parametrizzabile come pure quello per l'invio ad impulsi. Appena sarà possibile fare dei test, prendendo in considerazione la distanza e gli eventuali disturbi, verrà stabilito il valore minimo per cui la lettura risulti corretta.
Un altro metodo per diminuire il tempo d'invio e quindi del consumo, è quello di eseguire una riduzione tramite un calcolo matematico sui valori letti dai sensori.

4.4. Diagramma di sviluppo del progetto

Per fare in modo di riuscire a sviluppare il progetto su due processori diversi, il programma verrà creato su più livelli.

Il seguente schema rappresenta i livelli e il loro posizionamento:

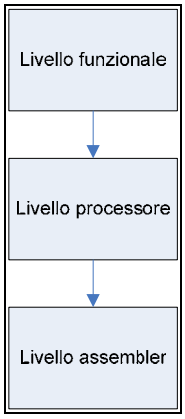

Figura 2: Schema a livelli

Livello assembler

Questo livello conterrà del codice, per svariati motivi (velocità, facilità, ecc.), scritto in assembler e quindi specifico per il processore utilizzato.

Livello processore

Contiene tutte le funzioni necessarie al funzionamento strettamente legate al processore. Un esempio di questo livello potrebbe essere l'accensione e lo spegnimento del trasmettitore.

Livello funzionale

Comprende la logica del programma e questa parte si appoggia sugli altri due.

4.5. Ambiente di sviluppo

Dopo una ricerca su internet sono stati identificati i seguenti strumenti di sviluppo:

Tool di sviluppo	Commento
Eclipse [4] con l'aggiunta del modulo Embedded e del compilatore specifico per il processore	La soluzione con Eclipse permette di rimanere con lo stesso tool di sviluppo per entrambi i processori e i compilatori sono OpenSource [5][6].
IAR System [7]	IAR System è forse il più completo e anche se a pagamento offre una versione gratuita denominata KickStart che permette di creare applicazioni limitate a 4 kb.
CrossWorks for MSP430 [8]	Tool che è utilizzato dai progettisti SSL.
µVision [3]	Tool di sviluppo per la scheda CC1010.

Tabella 2: Tools di sviluppo

Un vantaggio delle prime due possibilità è quello che supportano entrambi i processori, ma visto che i tools CrossWorks e µVision sono già conosciuti dal gruppo SSL opterò per queste soluzioni, in modo da poter approfittare dell'esperienza già raccolta e della consulenza degli assistenti, così da ridurre al minimo i ritardi di sviluppo.

4.6. Flessibilità del programma

Il programma dovrà prevedere una certa flessibilità, alcuni cambiamenti necessiteranno una nuova compilazione del codice ed altri invece si potranno eseguire tramite seriale.

5. Design e concezione

5.1. Schema UML

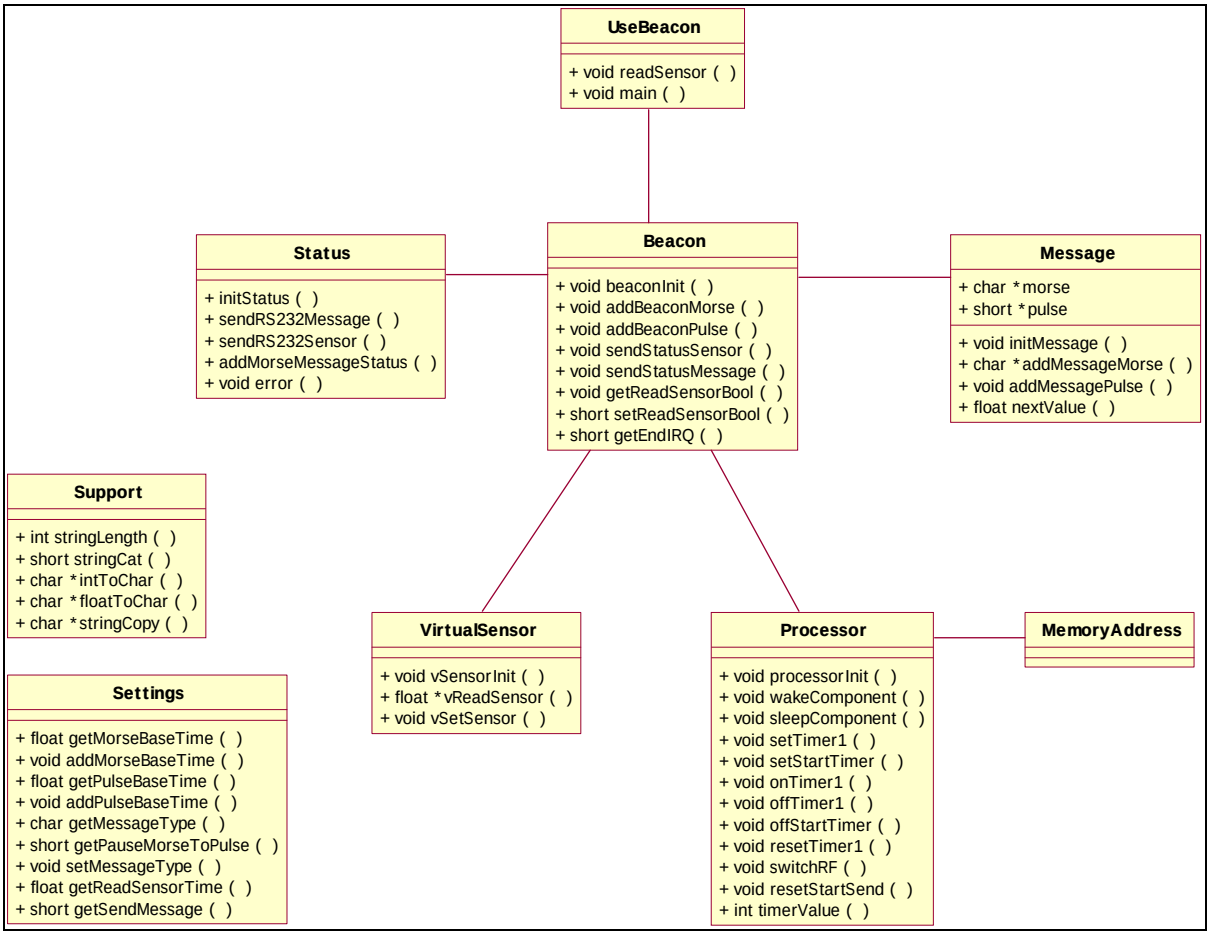

Figura 3: Schema UML

Lo schema racchiude le principali funzioni del programma inserite nel rispettivo file di intestazione.
Spiegazione di ogni singolo file:

UseBeacon
È un esempio di utilizzo del programma beacon.

Beacon
Il file beacon.c è il punto di accesso del progetto. Le funzioni contenute in questo file si appoggiano, quando necessario, alle altre. In questo modo se si vuole manipolare il funzionamento del beacon è possibile eseguire tutte le operazioni senza dover accedere agli altri files. Possiamo vedere questo programma come un package e il beacon.c è il suo punto di accesso.

Message
Contiene le procedure per la creazione del messaggio da inviare sia per la codifica Morse e sia per quella ad impulsi.

Status
Sono state raggruppate le funzioni necessarie per la comunicazione tramite la seriale.

Support
Al suo interno si possono trovare delle funzioni di conversione ed elaborazione delle stringhe.

Settings
La maggior parte dei parametri del programma sono in questo file c.

VirtualSensor
Simula l'esistenza dei sensori.

Processor
Contiene i drivers per le parti necessarie del processore.

MemoryAddress
Sono definite alcune costanti legate agli indirizzi del processore e del trasmettitore.

5.2. Inizializzazione del beacon

Il beacon inizializza e prepara oltre a se stesso anche tutte le parti necessarie al funzionamento e più precisamente:

1. Inizializza il message.
2. Inizializza la seriale.
3. Inizializza la VirtualSensor.
4. Aggiunge la stringa, passata come argomento, nel message come messaggio Morse e nel file di Status che lo invierà al terminale.
5. Inizializza il processore.
6. Inizializza il trasmettitore con la frequenza specificata come argomento (MSP430).
7. Inizializza il trasmettitore (CC1010).

5.3. Lettura dei sensori

La lettura dei sensori, com'è già stato spiegato, è simulata. All'interno del programma esiste un file .c denominato con VirtualSensor, il quale contiene un vettore di grandezza fissa impostata tramite la costante SENSORS_NUM situato nel file settings.h e in più dispone delle seguenti funzioni:

```
// inizializza il vettore contenente i valori
void vSensorInit (void);
// ritorna il vettore dei sensori
float *vReadSensor(void);
// cambia il valore di un sensore
void vSetSensor(short id, float value);
```

Nel file useBeacon è stata creata una funzione (readSensor())che si occupa di interagire con la VirtualSensor e di inviare i valori al beacon.
Dopo aver letto i valori viene impostata la variabile tramite la funzione setReadSensorBool(short boolean) a false, grazie alla quale non verrà ripetuta la lettura dei sensori fino a quando questa variabile non verrà settata a true.
In riferimento a quanto descritto nell'analisi verrà utilizzato un vettore dove verranno salvati i valori dei sensori. Questa struttura si trova nel file message.c ed è stata

chiamata pulse. La sua grandezza viene anch'essa definita tramite la costante SENSORS_NUM.
Come nell'invio del messaggio Morse, anche in questo caso viene creato un vettore (pulseCode) contenente i fattori di moltiplicazione del tempo di base.

Nell'analisi si è pensato di utilizzare un calcolo che si occupa di ridurre il valore. La formula è la seguente:

$$valoreTrasmesso = \frac{valoreMisurato}{10} \pm 10$$

Formula 2: Riduzione del valore

La divisione per 10 riduce la grandezza del numero. La somma con il numero 10 permette di inviare anche il valore 0 e inoltre non è possibile confonderlo con il codice Morse dal momento che il suo fattore di moltiplicazione più grande è 7.
Come si può vedere dalla formula 2 il dieci può essere sommato oppure sottratto, questo varia in base al segno del numero che si vuole ridurre. Cambiando lo start bit per i numeri negativi, la portante rimarrà spenta per lo stesso periodo come se fosse un valore positivo. La funzione che svolge questa operazione è stata chiamata adapted() e la si può trovare nel file message.c.

5.4. Varianti della codifica ad impulsi

L'invio dello stato del satellite avviene, come esposto precedentemente, con una codifica diversa.
Di questa codifica ne sono state identificate tre variazioni.

5.4.1. Variante A

Ogni informazione che si vuole inviare ha una finestra di tempo; qualora il valore fosse più piccolo, la portante rimarrebbe spenta fino allo scadere del tempo definito dalla finestra. Come è già stato introdotto nell'analisi verrà accesa la portante per indicare l'inizio e la fine del valore, con questo metodo verrà attivata la portante anche per identificare il termine della finestra, quest'ultima indicherà inoltre l'inizio del prossimo valore da inviare.
La Figura 4 mostra uno schema di funzionamento:

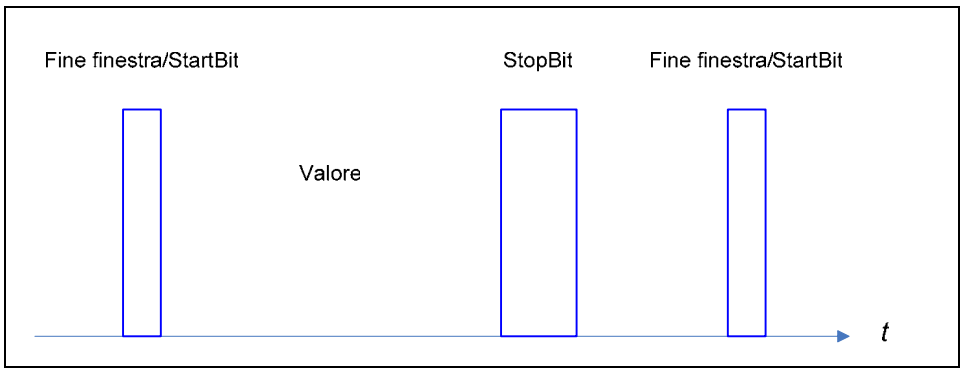

Figura 4: Variante A

Vantaggi
Il tempo necessario per inviare ogni singolo valore è prefissato dalla finestra; nel caso in cui durante una comunicazione avvenisse un disturbo sarebbe possibile stabilire di quale

sensore si sta leggendo il valore, tenendo in considerazione il tempo in cui la ricezione non è avvenuta.

Svantaggi
Per valori molto più piccoli della finestra, sarà necessario attendere comunque il termine di quest'ultima, rendendo così la trasmissione più lenta.

Esempio
Con questa codifica il vettore pulseCode conterrà le seguenti informazioni:

```
pulseCode[0]=START_BIT; //portante accesa
pulseCode[1]=value; // portante spenta
pulseCode[2]=STOP_BIT; //portante accesa
pulseCode[3]=WINDOW-value-STOP_BIT; //portante spenta
```

Il valore value viene preso dal vettore pulse.

5.4.2. Variante B

Questa variante (Figura 5) è simile alla precedente, ma con la differenza che la lo start bit, varierà in base al sensore inviato.

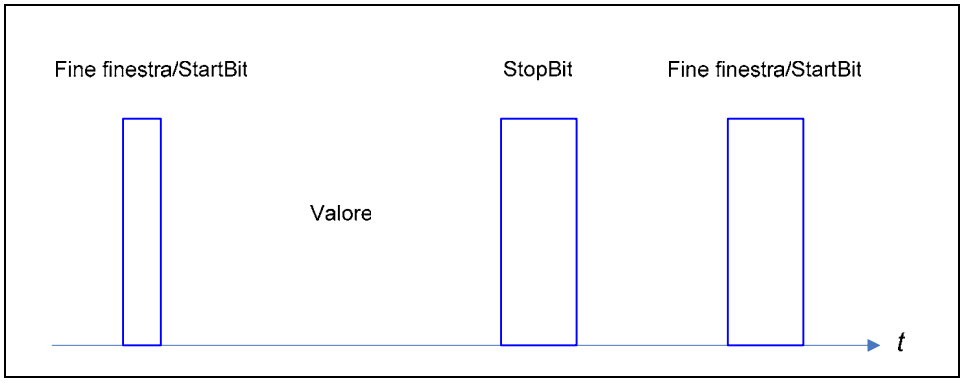

Figura 5: Variante B

La fine della finestra risulta essere anche in questo caso lo start bit del valore successivo.

Vantaggi
Oltre ai vantaggi della variante A, si potrà distinguere in qualsiasi momento durante la lettura dello start bit, di quale sensore si sta ricevendo il valore.

Svantaggi
La trasmissione risulta ancora più lenta della variante precedente, dal momento che lo start bit cresce in modo lineare con il numero del sensore inviato.

Esempio
Con questa codifica il vettore pulseCode conterrà le seguenti informazioni:

```
pulseCode[0]=START_BIT+idSensor; //portante accesa
pulseCode[1]=value; // portante spenta
pulseCode[2]=STOP_BIT; //portante accesa
pulseCode[3]=WINDOW-value-STOP_BIT; //portante spenta
```

Il valore value viene preso dal vettore pulse.

5.4.3. Variante C

Questa variante (Figura 6) non necessita di una finestra e il segnale di termine di un valore indica l'inizio del valore successivo.

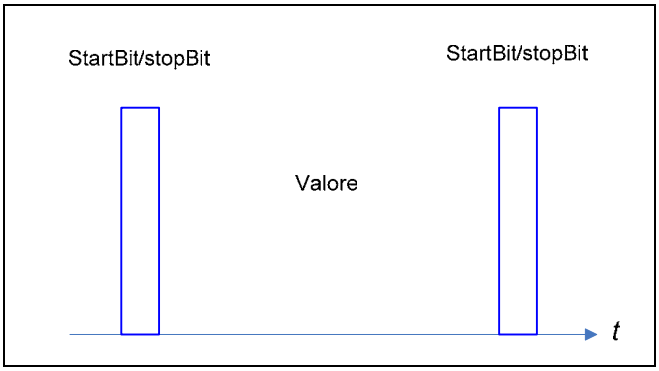

Figura 6: Variante C

Vantaggio
Con il fatto che non si ha più bisogno della finestra la trasmissione diventa più veloce, se paragonata con gli altri due metodi.

Svantaggi
Non sarà possibile stabilire di quale sensore si stanno ricevendo i dati, se la lettura non avviene all'inizio della trasmissione dei valori.

Esempio
Con questa codifica il vettore pulseCode conterrà le seguenti informazioni:

```
pulseCode[0]=value; //portante spenta
pulseCode[1]=STOP_BIT; // portante accesa
```

Quest'ultima codifica ha il primo valore diverso da tutti gli altri, infatti non ha lo stop bit dell'invio precedente, per cui ne è stato aggiunto uno davanti al valore; la pulseCode del primo invio è perciò così formata:

```
pulseCode[0]=STOP_BIT; //portante accesa
pulseCode[1]=value; //portante spenta
pulseCode[2]=STOP_BIT; // portante accesa
```

5.4.4. Gestione

Per identificare quale variante è stata impostata, è stata programmata nel file settings.c la funzione getMessageType() che restituisce il metodo selezionato in quel momento.
Per la creazione del messaggio in queste tre varianti sono state sviluppate le seguenti funzioni:

```
void addMessagePulseA(float value, short *endPulse, float *pulseCode);
void addMessagePulseB(float value, short *endPulse, float *pulseCode);
void addMessagePulseC(float value, short *endPulse, float *pulseCode);
value: valore che si vuole inviare
endPulse: numero di valori contenuti nel vettore
pulseCode: vettore con i fattori di moltiplicazione
```

La procedura che richiama la funzione corretta è la nextPulseCode ed è situata, assieme alle precedenti 3, all'interno del file message.c .

Nello stesso tempo, i valori start bit, stop bit, la lunghezza della finestra e l'allungamento per i numeri negativi, sono contenuti nel file message.c e sono delle costanti così nominate:

```
#define NEGATIVE_STEEP 1;
// Constanti per l'invio dei dati ad impulsi per il metodo A
#define START_A 1;
#define STOP_A 3;
#define WINDOW_A 50;

// Constanti e variabili per l'invio dei dati ad impulsi per il
metodo B
short idSensor=0;
#define IDSENSOR_STEEP 1;
#define START_B 3;
#define STOP_B 1;
#define WINDOW_B 50;
#define PAUSE_B 1;

// Constanti e variabili per l'invio dei dati ad impulsi per il
metodo C
#define STOP_C 1;
```

5.5. Punti di sincronismo

Il sincronismo fra il CubeSat e la stazione di ascolto è stato posto fra il termine del messaggio Morse e l'inizio di quello ad impulsi. Viene eseguito uno spegnimento della portante con un fattore definito nella variabile pauseMorseToPulse nel file settings.c, moltiplicato con il tempo base del codice ad impulsi. Questa variabile viene letta tramite la funzione getPauseMorseToPulse contenuta anch'essa nel file settings.c .
Come già introdotto, la manipolazione della portante avviene cambiando il suo stato attuale (da portante spenta a portante accesa e viceversa). Per far si che questa operazione sia sempre corretta e che quindi non avvenga un'accensione piuttosto che uno spegnimento all'interno del messaggio, vengono inseriti dei punti di sincronismo, con lo scopo di forzare il prossimo stato del trasmettitore.
I punti di sincronismo avvengono:

1. All'interno del codice Morse dopo ogni spazio fra parole.
2. Al termine del messaggio Morse e prima di quello ad impulsi.
3. Alla fine della trasmissione.

Per eseguire questa operazione è sufficiente richiamare le funzioni nextSwitchRFOff(), nextSwitchRFOn() e offRF(), situate nel driver del processore.

5.6. Gestione del messaggio

5.6.1. Morse

All'interno del file message.c, è stata creata una stringa di lunghezza fissa con il nome morse.
Per aggiungere un messaggio si deve chiamare il comando addBeaconMorse(char *str) che a sua volta richiama la funzione addMessageMorse(char *str), la quale accoda alla variabile morse il valore contenuto in "str". Eseguita questa operazione la funzione

ritorna il puntatore all'inizio della variabile morse e la passa alla funzione addMorseMessageStatus(char *str), la quale invia l'intero messaggio al terminale. In questo modo si aggiorna automaticamente il messaggio da inviare tramite il trasmettitore e quello da spedire al terminale.

La Figura 7 riproduce il diagramma di sequenza dell'intera operazione:

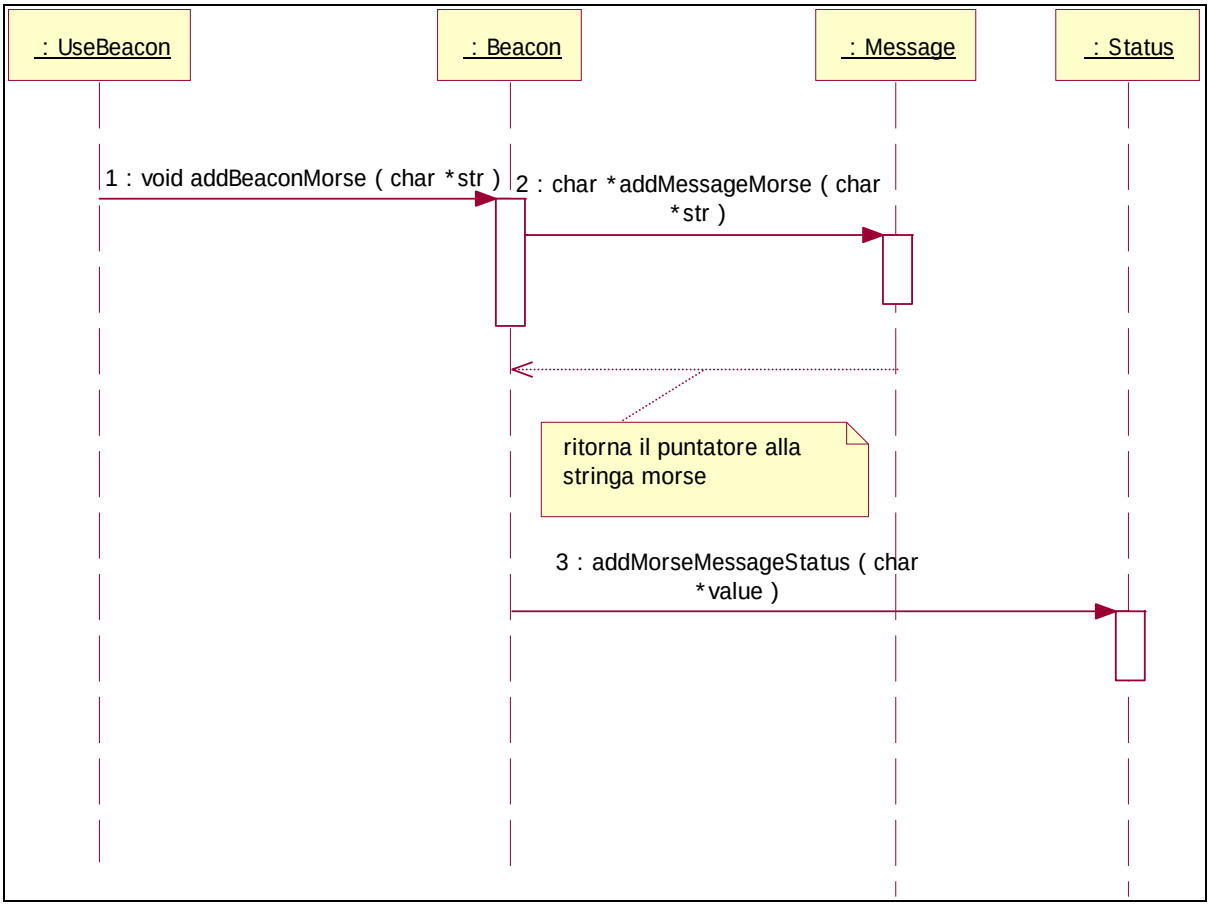

Figura 7: Diagramma di sequenza per stringa Morse

Questa struttura permette d'inviare anche i valori letti dai sensori. È sufficiente richiamare la funzione addBeaconMorse con all'interno il valore del sensore convertito in stringa.

5.6.2. Modulazione ad impulsi

Per poter collocare un valore all'interno di questa struttura deve essere chiamata la funzione addBeaconPulse, la quale a sua volta chiamerà la addMessagePulse.
La addBeaconPulse richiede come parametro un id di tipo short e un valore di tipo float. Nel caso in cui non venisse mai specificato il valore per un determinato id, quest'ultimo varrà 0.

La Figura 8 illustra il procedimento descritto:

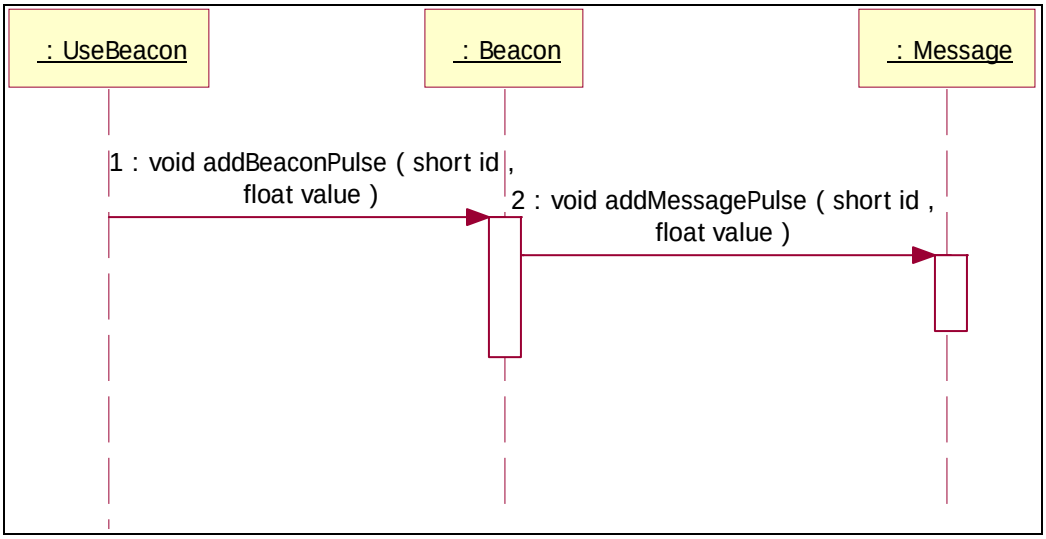

Figura 8: Diagramma di sequenza per invio tramite transceiver

5.7. Avvio della trasmissione

Per stabilire l'inizio della trasmissione è stato utilizzato un secondo timer (definito come start timer), legato alla funzione di interrupt startSend (Figura 9) posta nel file beacon.c. Per impostare i tempi di attesa del timer e quante letture si devono svolgere dai sensori, sono state realizzate le seguenti variabili nel file settings.c:

```
volatile float readSensorTime=1; // tempo di base in secondi per la lettura
dei sensori (0<valore<=15 )
volatile short sendMessage=1; // ogni n letture dei sensori invia i dati
```

Dopo aver eseguito sendMessage letture, si procede con l'invio del messaggio.
La funzione che verifica questa condizione è la getBoolSendMessage ed è stata messa nel file beacon.c. Attualmente questa funzione verifica solo le condizioni appena menzionate; in futuro sarà possibile inserire ulteriori controlli prima di procedere con l'invio dei dati, come ad esempio la verifica dello stato delle batterie.

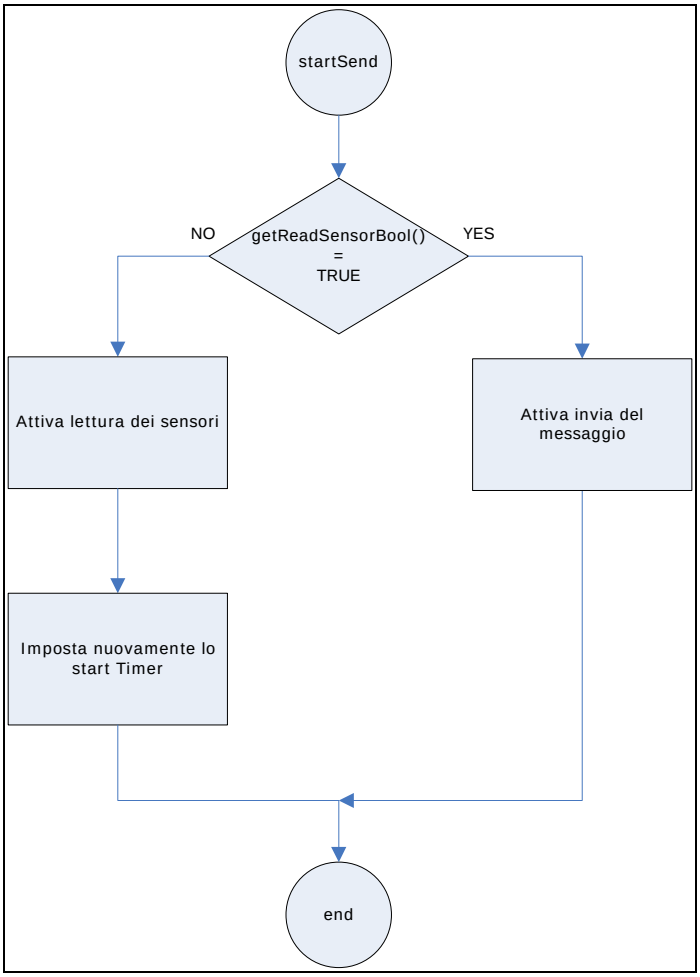

Figura 9: Diagramma di flusso startSend

5.7.1. Invio del messaggio

Il codice identificativo verrà immesso davanti alla variabile, nella quale è salvato il messaggio da inviare come codice Morse.
La prima versione del software prevedeva che la variabile morse fosse una stringa dinamica, ma creava diversi problemi durante la reallocazione; per dare comunque una flessibilità si è pensato di definire una costante che stabilisce la grandezza massima della stringa. La costante è stata inclusa nel file settings.h con il nome di STRING_LENGTH ed impostata a 80.
Com'è stato spiegato nello studio, quando viene richiesto un codice Morse per un determinato carattere, vengono ritornati i fattori di moltiplicazione. La funzione che si occupa di eseguire questa operazione è la:

```
void getMorseFrom(char ch, short *finish, short *code);
ch : carattere che si vuole convertire
finish: numeri di fattori inseriti nell'array
code: l'array contenente i fattori.
```

Riprendendo l'esempio precedente, la funzione getMorseFrom restituirebbe il seguente vettore:

```
actualCode [0] = 1; fattore per il punto
actualCode [1] = 1; fattore per lo spazio
actualCode [2] = 3; fattore per la linea
```

Il vettore actualCode ha grandezza di 11, questa è la quantità massima necessaria per il funzionamento ed è anche una variabile globale così da poter utilizzare sempre la stessa per tutte le conversioni.
Come già illustrato nello studio, per gestire lo spegnimento e l'accensione della portante, è stato utilizzato un timer. Dopo aver attuato la conversione basterà chiamare la funzione che cambia lo stato del trasmettitore e di seguito impostare il timer cambiando l'indice dell'array.
Illustrazione di uno pseudocodice che descrive questo funzionamento:

```
switchRF();
setTimer(actualCode[i]*base);
i++;
```

La Figura 10 mostra il diagramma di flusso della funzione di interrupt legata al timer:

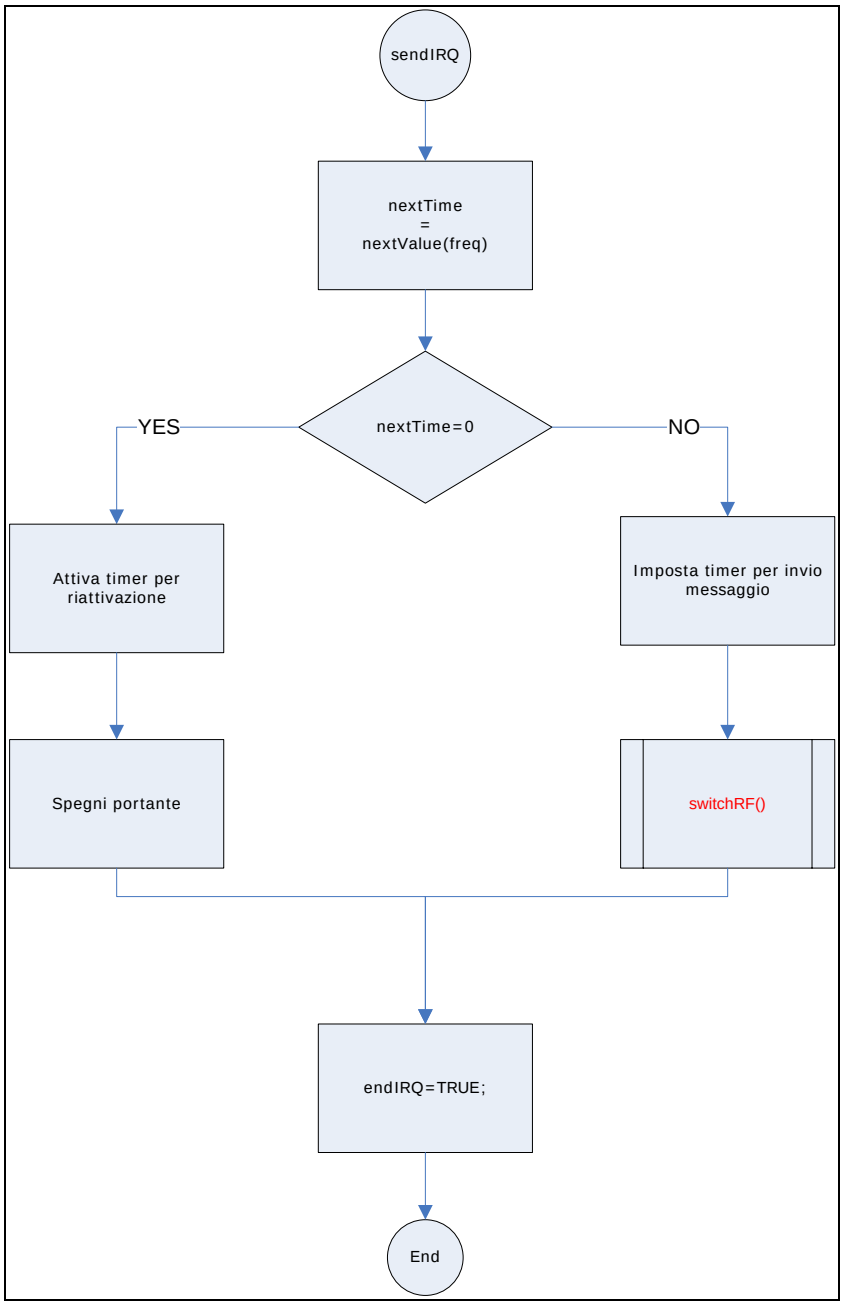

Figura 10: Diagramma di flusso sendIRQ

La funzione che ritorna il prossimo periodo è la nextValue ed è contenuta nel file message.c. Come si può vedere dalla Figura 10, se la funzione nextValue restituisce 0, significa che è stata terminata la trasmissione.

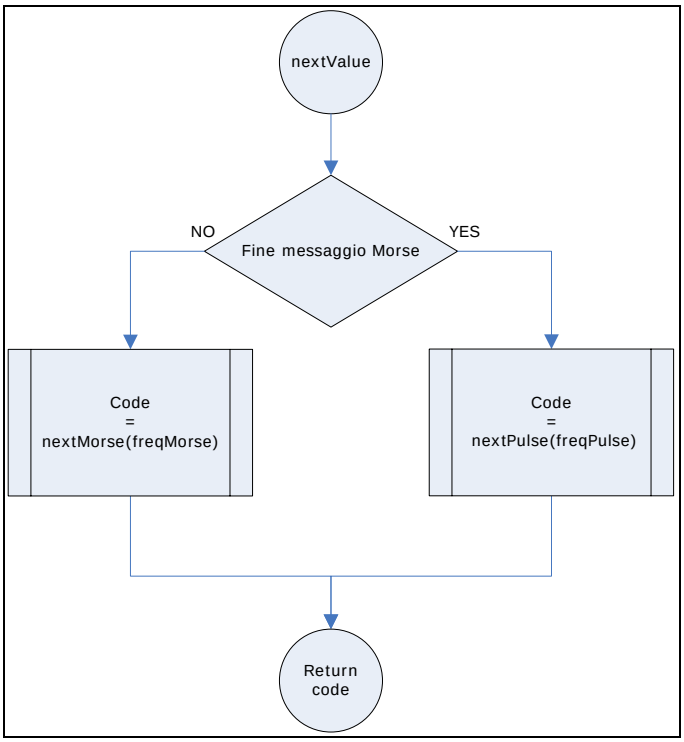

Figura 11: Diagramma di flusso nextValue

La funzione che gestisce l'indice del vettore è la stata chiamata nextMorse (Figura 12) e viene richiamata dalla nextValue nella fase d'invio di questa codifica. La nextMorse serve anche per gestire se il carattere attuale è stato inviato completamente e quindi se è necessario passare al prossimo carattere oppure se ci sono ancora valori non trasmessi nel vettore attuale.

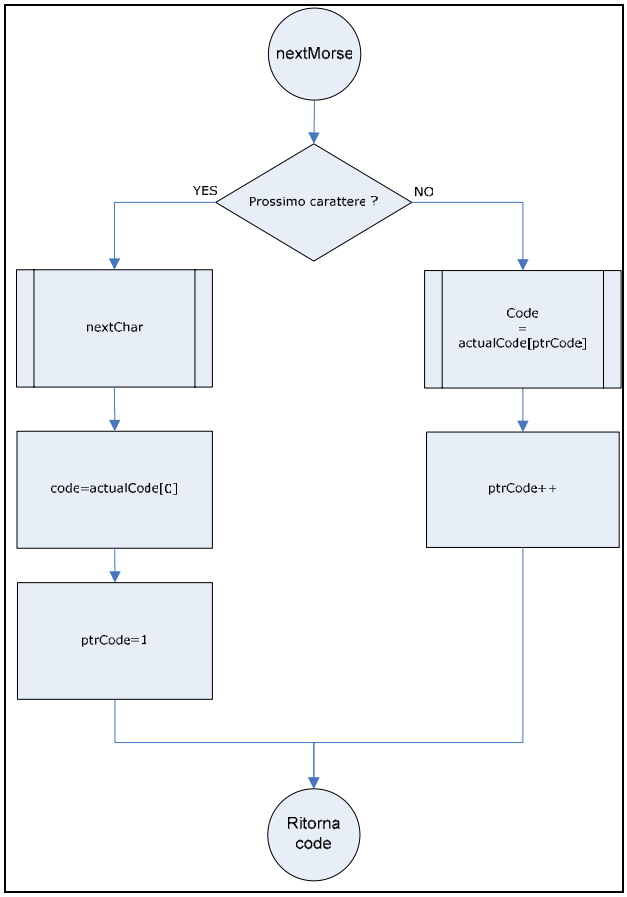

Figura 12: Diagramma di flusso nextMorse

Come si può vedere dal diagramma di flusso è stata progettata un'ulteriore funzione (nextChar), la quale è in grado di ritornare il prossimo carattere in Morse. È stata programmata per poter gestire meglio il carattere di spazio, infatti quest'ultimo è diverso dagli altri. La nextChar (Figura 13) restituisce nel caso di spazio la moltiplicazione fra 7, che è l'identificato in codice Morse dello spazio e il tempo di base.

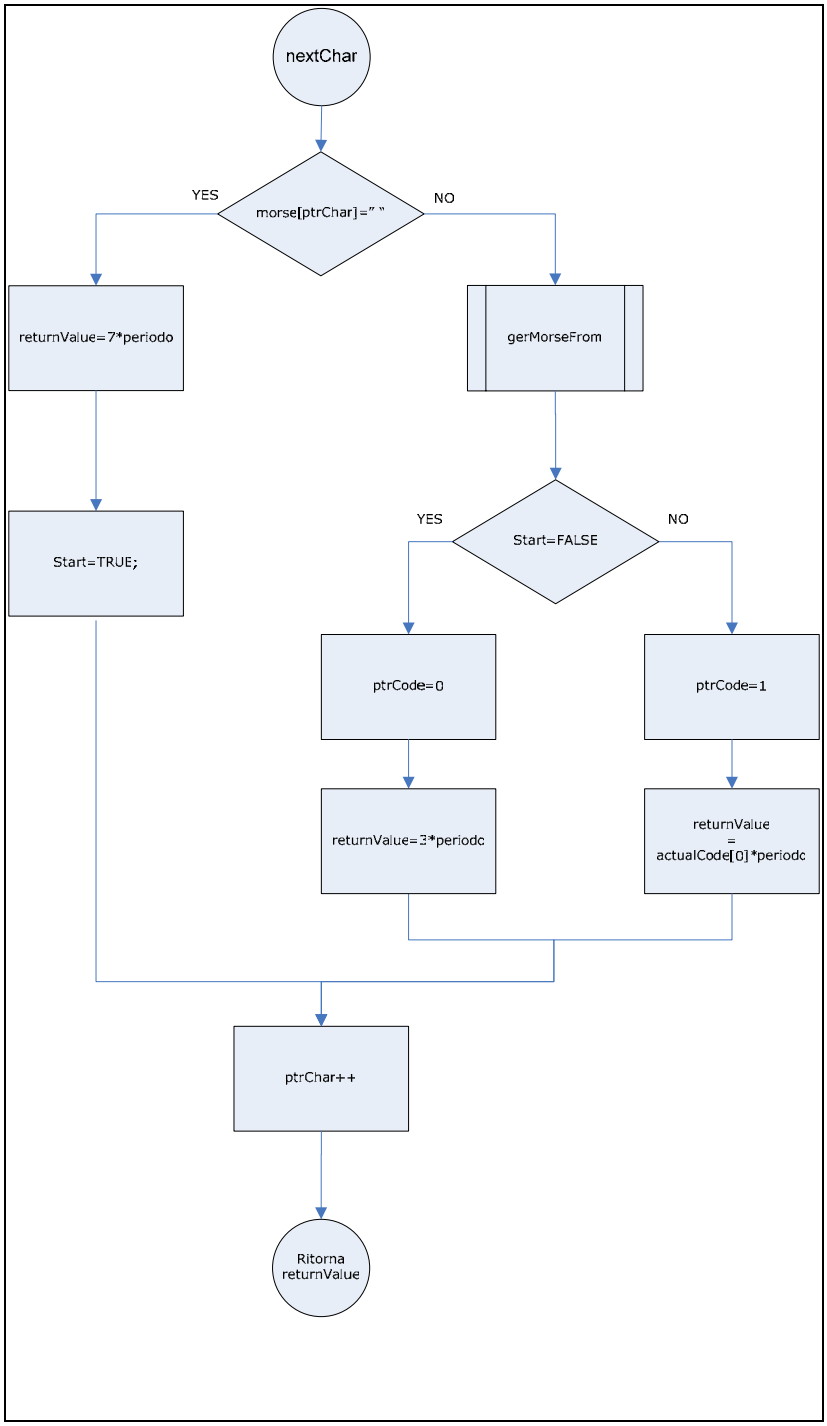

Figura 13: Diagramma di flusso nextChar

La seconda condizione di if permette di ritornare il primo valore del prossimo carattere nel caso in cui nella chiamata precedente si ha elaborato uno spazio; mentre se il carattere non è stato uno spazio verrà impostato il timer con 3*periodo, vale a dire il valore di base del Morse, per uno spazio fra lettere.

Terminata questa fase d'invio viene attivata quella ad impulsi. La funzione nextValue richiama la funzione pulseCode:

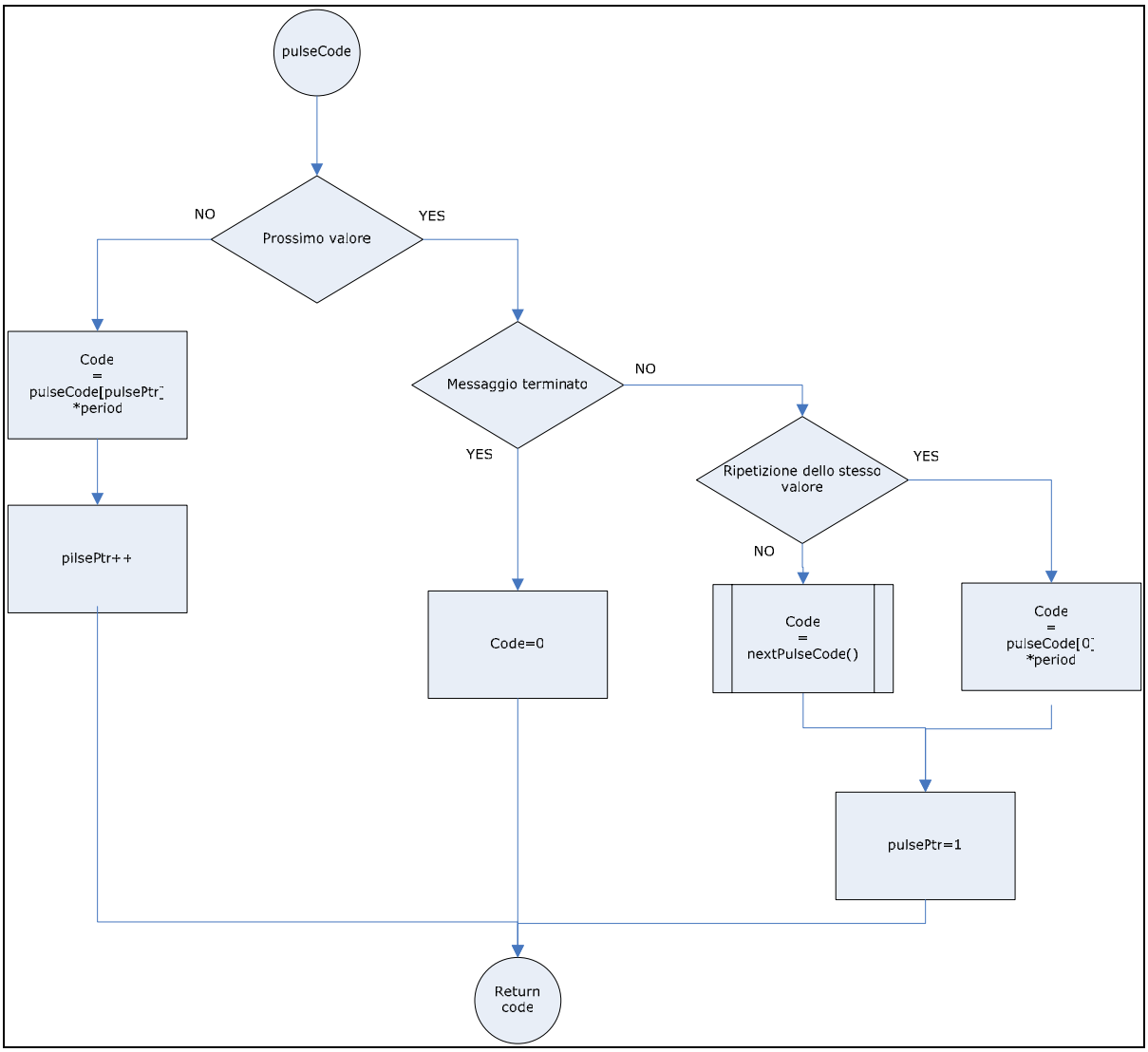

Figura 14: Diagramma di flusso pulseCode

Come si può vedere anche nel digramma di flusso la funzione pulseCode esegue un controllo sulla ripetizione dello stesso valore. Nel file settings.h esiste infatti una costante REPEAT_SENSOR_VALUE che permette la ripetizione dei valori; se questa viene impostata ad esempio a 2, ogni valore verrà ripetuto due volte consecutivamente. Da ultimo per elaborare il prossimo valore viene richiamata la funzione nextPulseCode:

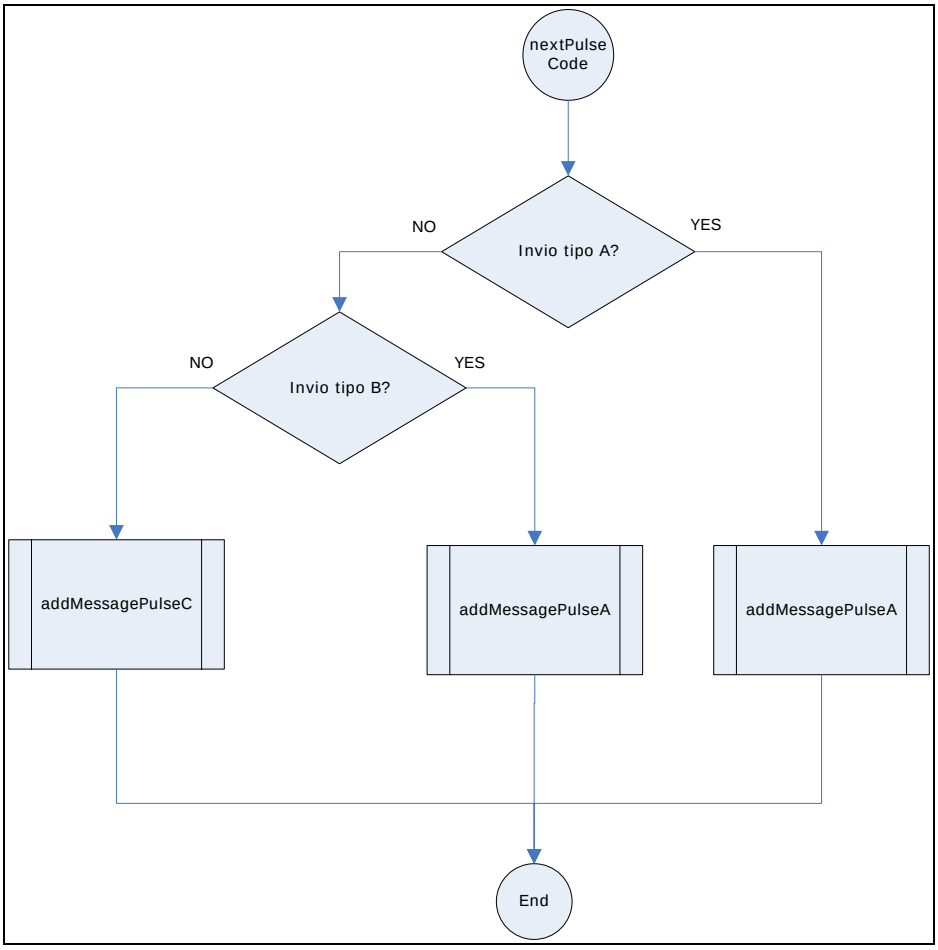

Figura 15: Diagramma di flusso nextPulseCode

La nextPulseCode a sua volta richiama la funzione che codifica il valore in base alla variante (a,b o c) scelta.

5.8. Comunicazione della seriale

Per l'invio del messaggio di testo se ne è già parlato nel capitolo 5.6.1. L'invio dei valori al terminale invece è separato dall'aggiunta nel beacon e quindi per poterli inviare al PC, si deve utilizzare la funzione sendStatusSensor inserita nel file beacon.c, la quale successivamente richiama la sendRS232Sensor sviluppata nel file status.c.

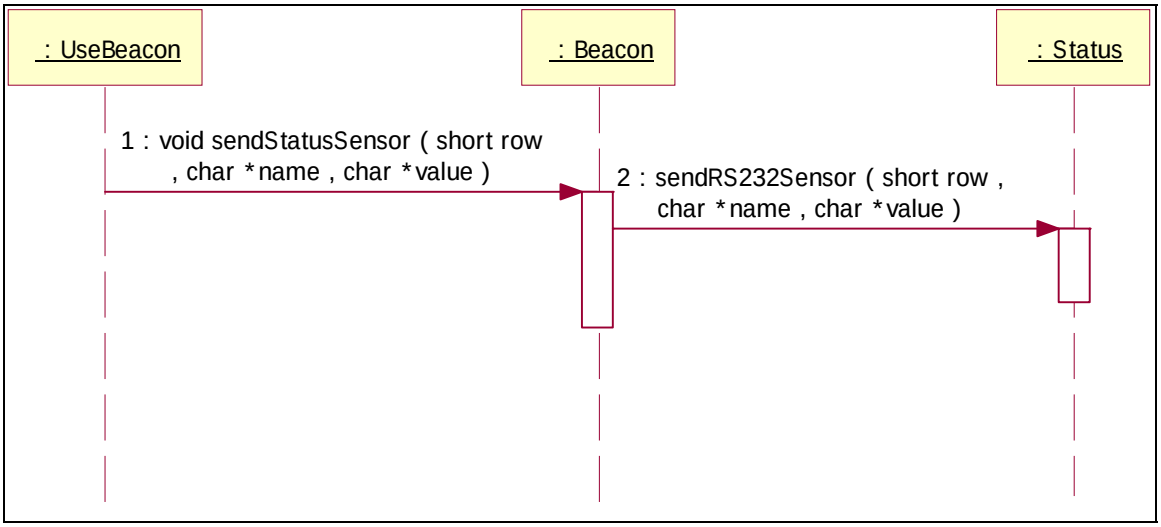

Figura 16: Diagramma di sequenza per invio tramite seriale

La seriale è interamente gestita dal file status.c .
La seguente funzione:

```
sendRS232Message(short row, char *name, char *value)
short row: linea su cui si vuole scrivere l'informazioni
name: nome del valore
value: valore
```

ha una variabile row che identifica la riga in cui deve essere stampato il messaggio.

Le prime 4 righe del terminale sono così suddivise:

1. Messaggio.
2. Tipo di impulso selezionato.
3. Tempo di base del Morse.
4. Tempo di base ad impulsi.

Per questo motivo quando il programma riceve una richiesta alla riga "row" quest'ultima viene aumentata di 4.
La funzione sendRS232Sensor aggiunge davanti al "name" il numero del sensore e poi richiama la sendRS232Message, la quale invia i dati alla seriale.

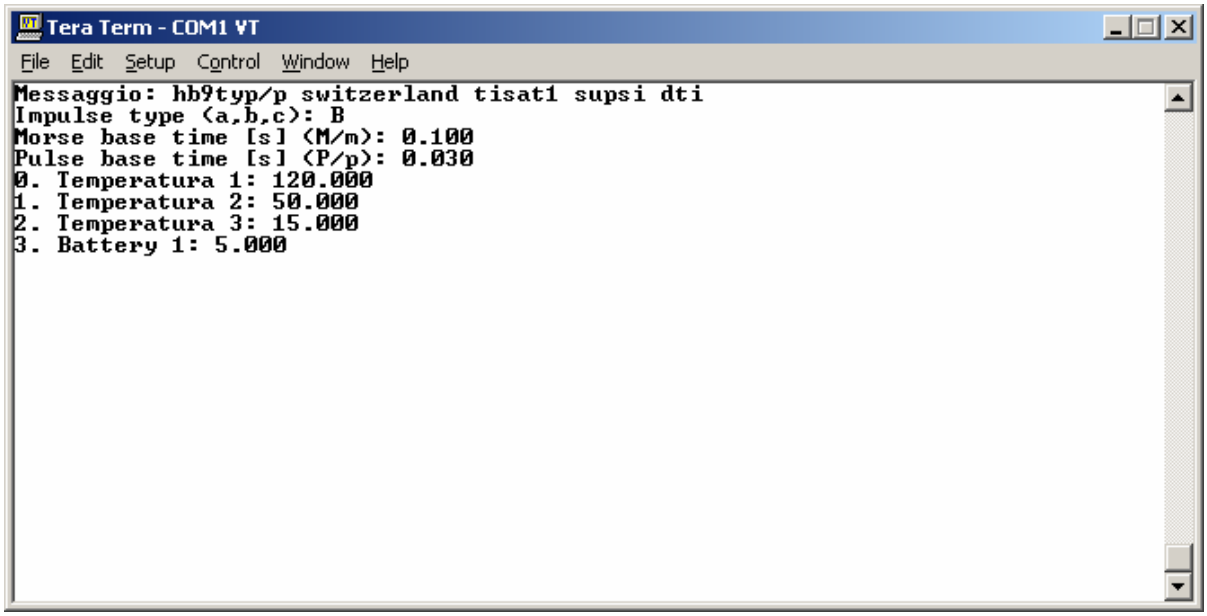

Figura 17: Comunicazione tramite seriale

Per la gestione dei dati da inviare è stato utilizzato un buffer circolare. Questo è situato nel file status.c e la sua grandezza è definita dalla costante BUFFER_SIZE nel file settings.h . Nel buffer vengono inserite le seguenti informazioni:

1. Codice escape per posizionarsi sulla riga “row”.
2. Codice escape per la cancellazione della riga.
3. Inserimento della stringa “name”.
4. Inserimento della stringa “value”.

Come si può vedere anche nella schermata precedente, è presente una sezione dedicata agli errori riscontrati dal programma. La funzione che gestisce gli errori è stata chiamata error e si trova nel file status.c .
Attualmente viene visualizzato un errore nei seguenti casi:

1. Il valore del sensore supera la finestra.
2. È stata superata la lunghezza massima della stringa morse.
3. Si cerca di salvare o modificare i valori dei sensori dando un identificatore errato.
4. Si inserisce tramite seriale un’opzione non esistente.
5. Quando viene richiesto una diminuzione del tempo di base inferiore al valore 0.01.

Le operazioni di scrittura e di ricezione tramite la seriale vengono gestite da un interrupt.

Tramite la seriale è possibile modificare il funzionamento del programma; nella Tabella 3 sono riportati i comandi per modificare le impostazioni:

Comandi	Operazione
a, b o c	Modifica il tipo di impulso
e	Cancella la lista degli errori
M	Aumenta il tempo di base del Morse
m	Diminuisce il tempo di base del Morse
P	Aumenta il tempo di base degli impulsi
p	Diminuisce il tempo di base degli impulsi

i	Cambia il valore di un sensore

Tabella 3: Comandi per la seriale

Dopo aver premuto l'opzione *i* verrà richiesto il numero del sensore (situato davanti al nome) e infine il valore che si vuole assegnare.

Questa parte di codice viene compilata solo nella versione di debug.

5.9. Esempio di utilizzo

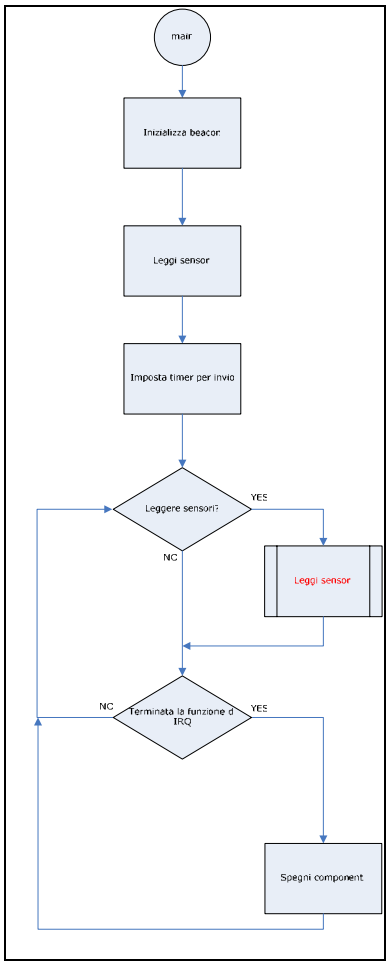

Figura 18: Diagramma di flusso del main

Nella Figura 18 è riportato il main del file useBeacon.c, questo è un esempio di utilizzo del programma. All'interno si può trovare:

1. Inizializzazione del beacon.
2. Inizializzazione del trasmettitore.
3. Invio della potenza di uscita dell'antenna RF.
4. La lettura dei sensori tramite la virtualSensor.
5. Impostazione del timer start Send.
6. Controllo sul termine della funzione sendIRQ.
7. Aggiunta dei valori dei sensori al beacon.
8. Invio al terminale dei valori dei sensori.
9. Impostazione a false della lettura dei sensori.
10. Messa in standby del processore.

6. Sviluppo su MSP430

Il modello della scheda di sviluppo è la MSP430F169 prodotta dalla softBaugh [9]

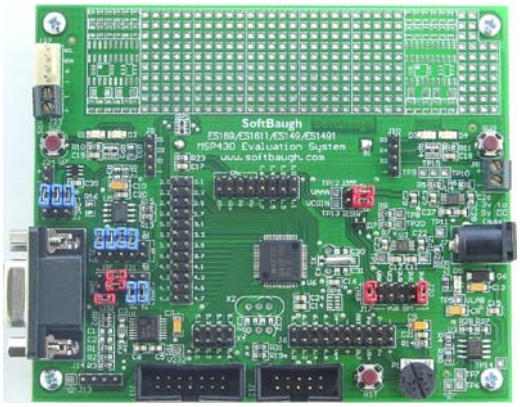

Figura 19: Scheda MSP430F169

I principali dati tecnici della scheda sono:

- processore programmabile MSP430F169
- interfaccia RS232 della Texas Instruments (MAX3221)
- alimentazione tramite 3.3v
- collegamento con il tool di sviluppo tramite JTAG
- 4 led verdi
- 2 bottoni senza interrupt
- quarzo a 32kHz.

Per lo sviluppo tramite MSP430 si è dovuto aggiungere la comunicazione con il trasmettitore, il modello scelto è il CC1100[3] e le sue caratteristiche principali sono descritte nella tabella sottostante:

Frequenza (Min)(MHz)	300, 400, 800
Frequenza (Max)(MHz)	348, 464, 928
Tensione di funzionamento (Min)(V)	1.8
Tensione di funzionamento (Max)(V)	3.6
Tecniche di modulazione	FSK, OOK, MSK, GFSK
Temperature per il funzionamento (°C)	-40 to 85
Consumo di corrente (RX)(mA)	14
Consumo di corrente (TX)(mA)	16.2
Velocità FSK (Max)(kbps)	500
Sensibilità della ricezione (FSK)(dBm)	-110
Range di potenza (dBM)	-30 to 10
RSSI in uscita	Digital
Collegamento dell'antenna	Differential

Tabella 4: Dati tecnici CC1100

Per la comunicazione fra MSP430 e il CC1100 si è utilizzato SPI. Questo modello prevede il seguente protocollo di comunicazione:

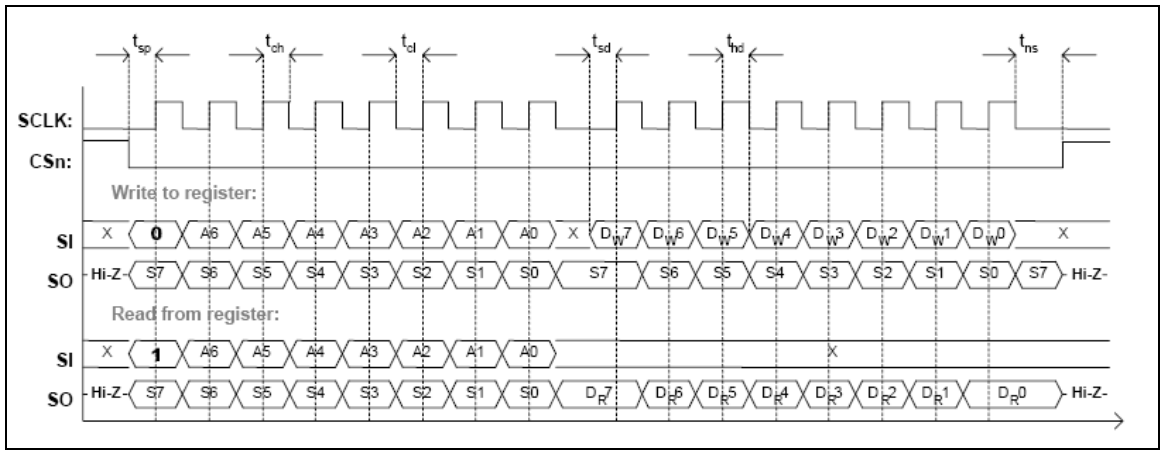

Figura 20: Protocollo di comunicazione SPI per CC1100

Il driver è stato programmato da Ivano Bonesana e prevede la possibilità di inviare le impostazioni per il funzionamento. Le funzioni sviluppate sono le seguenti:

```
void SPI0_init(void);
void SPI0_slow(void);
void SPI0_speedup(void);
char SPI0_sendString(unsigned char *str, int strlen);
unsigned char SPI0_putByte(unsigned char data);
```

Sulla base di queste funzioni ne sono state create altre più specifiche per l'invio dei dati ai registri interni del CC1100:

```
unsigned char SPI0_readReg(unsigned char address);
unsigned char SPI0_writeReg(unsigned char address, unsigned char data);
void SPI0_WriteBurstReg(char addr, char *buffer, char count);
void SPI0_writePatable(char *buffer, char count);
unsigned char SPI0_commandStrobe(char *command);
```

Le prime due permettono di interagire con i registri. La 3a e la 4a eseguono un'operazione di Burst che nel programma viene utilizzato per impostare la potenza del trasmettitore. L'ultima invece serve ad impostare i comandi di strobe descritti nella Tabella 5.

Per queste funzioni non sono state seguite le regole di scrittura del codice inserito in questo documento, dal momento che il driver non è stato da me prodotto.

Address	Strobe Name	Description
0x30	SRES	Reset chip.
0x31	SFSTXON	Enable and calibrate frequency synthesizer (if MCSM0.FS_AUTOCAL=1). If in RX (with CCA): Go to a wait state where only the synthesizer is running (for quick RX / TX turnaround).
0x32	SXOFF	Turn off crystal oscillator.
0x33	SCAL	Calibrate frequency synthesizer and turn it off (enables quick start). SCAL can be strobed from IDLE mode without setting manual calibration mode (MCSM0.FS_AUTOCAL=0)
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and MCSM0.FS_AUTOCAL=1.
0x35	STX	In IDLE state: Enable TX. Perform calibration first if MCSM0.FS_AUTOCAL=1. If in RX state and CCA is enabled: Only go to TX if channel is clear.
0x36	SIDLE	Exit RX / TX, turn off frequency synthesizer and exit Wake-On-Radio mode if applicable.
0x38	SWOR	Start automatic RX polling sequence (Wake-on-Radio) as described in section 19.5.
0x39	SPWD	Enter power down mode when CSn goes high.
0x3A	SFRX	Flush the RX FIFO buffer. Only issue SFRX in the IDLE, TXFIFO_UNDERFLOW or RXFIFO_OVERFLOW states.
0x3B	SFTX	Flush the TX FIFO buffer. Only issue SFTX in the IDLE, TXFIFO_UNDERFLOW or RXFIFO_OVERFLOW states.
0x3C	SWORRST	Reset real time clock.
0x3D	SNOP	No operation. May be used to pad strobe commands to two bytes for simpler software.

Tabella 5: Comandi di strobe

I registri per la configurazione invece sono contenuti nel file MemoryAddress.

Per il processore MSP430 non è stato possibile inserire la richiesta di attivazione dei componenti all'interno del file CPUMSP430.c (wakeComponent), in quando quest'ultima deve essere inserita in una funzione di interrupt.

6.1. Risultati

Tempo base	Tempo letto [ms]	Tempo previsto [ms]	Valore spedito	Valore letto	Errore su valore letto
0.03	0.655	0.6600	120	118.33	1.41%
	0.446	0.4500	50	48.67	2.74%
	0.343	0.3450	15	14.33	4.65%
	0.310	0.3105	3.5	3.33	5.00%
0.05	1.096	1.1000	120	119.20	0.67%
	0.748	0.7500	50	49.60	0.81%
	0.574	0.5750	15	14.80	1.35%
	0.517	0.5175	3.5	3.40	2.94%
0.1	2.200	2.2000	120	120.00	0.00%
	1.496	1.5000	50	49.60	0.81%
	1.150	1.1500	15	15.00	0.00%
	1.034	1.0350	3.5	3.40	2.94%

Tabella 6: Risultati ottenuti con MSP430

Questi risultati sono stati ottenuti analizzando, tramite l'oscilloscopio, il pin collegato all'accensione del led. Questa accensione avviene dopo l'attivazione del trasmettitore.

I risultati si discostano dal valore spedito, perché la lettura tramite oscilloscopio non è abbastanza precisa, infatti paragonando il tempo letto [ms] e il tempo previsto [ms] si può osservare che la precisione varia da 0 ms a 5ms.

6.2. Memoria utilizzata

La versione di debug occupa circa 16.02KB ed è così suddivisa:

Sezione	Utilizzo in [KB]
CODE	14.33
CONST	0.69
IDATA0	0.27
INTVEC	0.03
UDATA0	0.71

Tabella 7: Memoria utilizzata nella versione di debug

Per contro la versione non di debug occupa circa 6.31KB:

Sezione	Utilizzo in [KB]
CODE	6.03
CONST	0.05
IDATA0	0.05
INTVEC	0.02
UDATA0	0.16

Tabella 8: Memoria utilizzata nella versione senza debug

7. Sviluppo su CC1010

Il modello della scheda di sviluppo utilizzata è la CC1010eb.

Figura 21: Scheda CC1010

I principali dati tecnici della scheda sono:

- processore programmabile Intel 8051
- 2 interfacce RS232
- alimentazione tramite 6.0v
- collegamento con il tool di sviluppo tramite parallela
- 4 led (rosso, verde, giallo e blu)
- 4 bottoni.

L'implementazione su questa scheda è stata completata. Purtroppo però c'è un difetto al quale non si è riusciti a porre rimedio ed è l'invio tramite seriale. Il problema è

riconducibile al controllo dello stato del buffer (della seriale) durante l'invio; quest'ultimo avviene tramite una variabile booleana, in quanto non mi è stato possibile capire su quale registro si deve verificare lo stato. Capita dunque che la variabile venga controllata un attimo prima del suo cambiamento e per questo non invia i dati in modo corretto.
Le funzioni di interrupt legate alla seriale, all'interno del processore i8051, sono state sviluppate in modo diverso, infatti non esiste la distinzione fra funzione per la ricezione e quella per l'invio, ma si deve eseguire un controllo per stabilire chi ha effettuato l'IRQ.
Il timer 0 di questo modello si può impostare al massimo fino a 50ms e il timer 2 fino a circa 1 secondo, per questa ragione le funzioni di interrupt impostano il tempo più volte prima di svolgere le funzioni legate al beacon. Questa modifica ha portato un errore durante la fase d'invio dei dati, visto che la re-impostazione del timer impiega circa 1ms di utilizzo della CPU. Per bilanciare questa carenza, al posto di impostare il timer a 50ms, viene impostato solo a 49ms. Grazie a questa soluzione i tempi risultano corretti.
Tranne il problema della seriale, il beacon è funzionante completamente.

7.1. Risultati

Tempo base	Tempo letto [ms]	Tempo previsto [ms]	Valore spedito	Valore letto	Errore su valore letto
0.03	0.661	0.6600	120	120.33	-0.28%
	0.452	0.4500	50	50.67	-1.32%
	0.347	0.3450	15	15.67	-4.26%
	0.311	0.3105	3.5	3.67	-4.55%
0.05	1.104	1.1000	120	120.80	-0.66%
	0.752	0.7500	50	50.40	-0.79%
	0.576	0.5750	15	15.20	-1.32%
	0.518	0.5175	3.5	3.60	-2.78%
0.1	2.200	2.2000	120	120.00	0.00%
	1.500	1.5000	50	50.00	0.00%
	1.150	1.1500	15	15.00	0.00%
	1.035	1.0350	3.5	3.50	0.00%

Tabella 9: risultati con CC1010

Le osservazioni sui risultati sono identiche a quelle del processore MSP430. Su questo processore sono già stati svolti dei miglioramenti sui tempi descritti nel capitolo precedente.

7.2. Memoria utilizzata

La versione di debug occupa circa 20.79KB ed è così suddivisa:

Sezione	Utilizzo in [KB]
CODE	19.43
CONST	0.02
XDATA	1.33
DATA	0.01

Tabella 10: Memory utilizzata nella versione di debug

Per contro la versione non di debug occupa circa 13.64KB:

Sezione	Utilizzo in [KB]
CODE	13.24
CONST	0.02
XDATA	0.37
DATA	0.01

Tabella 11: Memory utilizzata nella versione senza debug

La differenza di utilizzo fra il processore MSP430 e questo è dovuta all'uso di funzioni in C per la gestione del processore, quest'ultime sono fornite dal produttore Chipcon.

8. Porting su altre piattaforme

Nel capitolo dell'analisi si è specificato che il programma è stato suddiviso a livelli. Il livello assembler non è stato utilizzato e perciò tolto dal programma.
Il livello processore invece è formato dal suo file header "Processor.h", il quale contiene le funzioni necessarie al funzionamento.

Lista delle funzioni del processor.h:

```
// inizializza il processore
void initProcessor(void);
// attiva i componenti
void wakeComponent(void);
// disattiva i componenti
void sleepComponent(void);
// imposta tempo su timer 1
void setTimer1(short temp);
// imposta tempo su start timer
void setStartTimer(short temp);
// attiva timer 1
void onTimer1(void);
// attiva interrupt per start timer
void onStartTimer(void);
// spegne timer 1
void offTimer1(void);
// spegne start timer
void offStartTimer(void);
// spegne flag per interrupt timer 1
void resetTimer1(void);
// spegne flag per interrupt start timer
void resetStartTimer(void);
// on->off off->on scheda RF
void switchRF(void);
// forza lo spegnimento dell'antenna RF
void offRF(void);
// dato un tempo in secondi ritorna il valore da inserire nel
timer
int timerValue(float seconds);
// inizializza la seriale
void serialInit(void);
```

```
// attiva interrupt di invio su seriale
void onSerialTX(void);
// attiva interrupt di ricezione su seriale
void onSerialRX(void);
// disattiva interrupt di invio su seriale
void offSerialTX(void);
// disattiva interrupt di ricezione su seriale
void offSerialRX(void);
// disattiva interrupt globale
void globalISROff(void);
// attiva interrupt globale
void globalISROn(void);
void RFinit(void);

// impostazione del clock, utilizzato per SPI (forniti da
Ivano Bonesana)
void selMCLK(char clk, char divmx);
void selSMCLK(char clk, char divsx);
void selACLK(char divax);
void setDCO(char dcox,char rselx);
```

Le implementazioni delle funzioni appena elencate invece vanno inserite in un file .c; il suo nome sarà legato al processore in utilizzo. Ad esempio per il processore MSP430 l'implementazione del file processor.h è stato chiamato CPUMSP430.c .

Dopo aver scritto i drivers sarà anche necessario modificare le funzioni di interrupt (Tabella 12).

Nome funzione	File in cui è contenuta
sendIRQ	Beacon.c
startSend	Beacon.c
usart1Rx	Status.c
usart1Tx	Status.c

Tabella 12: Funzione di interrupt

9. Problemi riscontrati

- In più occasioni ho notato delle differenze fra le implementazioni delle librerie C in CrossStudio e quelle ANSI-C. Questa anomalia mi è stata confermata anche dall'ing. Ivano Bonesana.
- Il programma prevedeva la reallocazione della memoria. Durante le fasi di test risultava che questa operazione veniva svolta in modo corretto, ma in verità il programma entrava in un ciclo infinito. Dopo diversi tentativi si è scoperto che il problema era dovuto al continuo ingrandimento dello spazio, per cui si è potuto risolvere il difetto eseguendo una reallocazione più grande (circa 50 valori alla volta). È stata dunque sostituita con dei vettori di grandezza fissa; anche questo malfunzionamento può essere ricollegato al problema descritto al punto precedente.
- In alcuni casi dopo la compilazione del programma sul processore MSP430 e l'invio sulla scheda, il software non veniva eseguito. Questa lacuna mi è stata confermata dall''ing. Ivano Bonesana e dall'ing Andrea Spiga ed è imputabile ad un'anomalia di CrossStudio.
- Avendo sviluppato il programma prevalentemente sul processore MSP430 durante lo sviluppo per il processore i8051, ho riscontrato delle anomalie dovute ad operazioni che MSP eseguiva automaticamente, come ad esempio lo spegnimento delle richieste di interrupt.

10. Test eseguiti

- Durante i test svolti con la versione che permetteva la reallocazione della memoria, anche se quest'ultima entrava in un ciclo infinito, la trasmissione procedeva comunque in modo corretto grazie all'utilizzo dell'interrupt.

- I primi test sono stati svolti accendendo un led piuttosto che il trasmettitore.

- Per verificare il funzionamento del codice Morse si è utilizzato un microfono collegato al computer ed ad un cicalino (collegato all'accensione del led). L'audio in entrata veniva verificato con un programma, in grado di interpretare il codice Morse. Ne sono stati trovati diversi, ma quello che ottiene il risultato migliore e che permette il funzionamento senza dover impostare troppi parametri, è il CwGet e si può trovare all'indirizzo [10].
  Il valore del tempo di base per cui il software riesce a riconoscere il codice Morse è di 0.03 s.

- Come test per l'invio dei dati tramite impulsi è stato usato un programma SoundForge, il quale è un software per la registrazione dell'audio. Oltre a registrare l'audio è possibile stabilire il tempo trascorso fra due punti all'interno della trama, in pratica con questa soluzione si è simulato un oscilloscopio.

- Terminata la fase di test tramite SoundForge sui dati inviati ad impulsi, si è passato a testare il funzionamento tramite un oscilloscopio.

- Come verifica di accensione e spegnimento del trasmettitore è stato utilizzato un componente fornitomi (Figura 22) dal team SSL; esso funge da galvanometro grazie ad una modifica proposta dal prof. Allen Weston.

Figura 22: Amperometro

11. Sviluppi futuri

- Per l'impostazione del timer sul modello CC1010 sono state utilizzate delle librerie in C fornite dal produttore. Per ottenere dei risultati migliori sarebbe il caso di sostituire queste funzioni con del codice scritto in assembler.

- Eseguire un controllo sullo spazio disponibile del buffer circolare utilizzato per l'invio dei dati tramite seriale.

- Dare la possibilità di cambiare completamente il messaggio Morse durante l'esecuzione del programma.

- Attualmente sulla versione tramite MSP430 si può impostare il timer per la lettura dei sensori solo con dei valori compresi fra 0 e 15 (0 escluso) secondi. Negli sviluppi futuri sarebbe ottimale poter inserire valori maggiori. Questo può essere eseguito introducendo più chiamate della funzione di interrupt prima di svolgere i compiti assegnatogli. Ad esempio se si volesse impostare un tempo di 45 secondi, la funzione di interrupt verrebbe richiamata per 3 volte (impostando il timer a 15s) prima di eseguire le operazioni legate al beacon.

- Inserire un comando che permetta di ripulire la schermata del terminale.

- Attualmente tramite il processore MSP430 è possibile inserire la frequenza di funzionamento nella chiamata della funzione di inizializzazione del beacon. Questo non è stato eseguito sul processore CC1010 per mancanza di tempo. All'interno del cdrom si può trovare un file excel (CC1010-CalcoloFrequenza.xls) che permette di risalire alla formula necessaria per le impostazioni dei registri.

12. Bibliografia

[1] Documento: Modulazioni_digitali.pdf
Modulazione OOK

[2] Internet: http://www.wikipedia.org
Raccolta informazioni

[3] Internet: http://www.ti.com
Componenti utilizzati

[4] Internet: http://www.eclipse.org
Tool di sviluppo

[5] Internet: http://mspgcc.sourceforge.net/
Compilatore per MSP430

[6] Internet: http://sdcc.sourceforge.net/
Compilatore per Interl8051

[7] Internet: http://www.iar.com/
Tool di sviluppo

[8] Internet: http://www.rowley.co.uk/
Tool di sviluppo

[9] Internet: http://www.softbaugh.com
Scheda di sviluppo

[10] Internet: http://www.dxsoft.com/en/products/cwget/
Codice Morse

[11] Documento: msp.pdf
Documentazione su processore MSP430

Conclusioni

Il programma funziona in modo corretto ed è stato implementato completamente su entrambi i kits di sviluppo.

Questo progetto mi ha permesso, per la prima volta, di lavorare con due processori dei quali non possedevo nessuna conoscenza. Per poter sviluppare sui micro-controllori si deve procedere con molta più attenzione che con la programmazione su PC, dal momento che un piccolo errore può provocare risultati imprevedibili e difficili da individuare.

Piano di lavoro

Lavoro previsto	Settimana								Giorni effettivi	Lavoro effettivo
	1	2	3	4	5	6	7	8		
	Giorni previsti									
Consegna del materiale	0.5								0.5	-
Installazione server CVS	0.5								0.5	-
Ricerca dei software necessari	1								1	-
Lettura dei datasheet su MSP430	1.5								1.5	-
Lettura dei datasheet su CC1010	1								1	-
Redigere documentazione	0.5								0.5	-
Lettura dei datasheet su CC1010		1.5							1.5	-
Riunione di progetto		0.5							0.5	-
Ricerca sul funzionamento del beacon		0.5							0.5	-
Ricerca sul funzionamento del Morse		0.5							0.5	-
Soluzione al risparmio energetico		1.5							1.5	-
Scrittura documento su regole di programmazione		0.5							0.5	-
Soluzione al risparmio energetico			0.5						0.5	-
Ricerca informazioni su CC1010			0.5						0.5	-
Installazione dei software necessari			0.5						0.5	-
Brainstorming			0.5						0.5	-
Redigere documentazione			0.5						0.5	-
Diagramma di flusso 1° versione			1						1	-
Redigere documentazione			0.5						0.5	-
Eventuali ritardi			1						1	Sviluppo su MSP430
Prove per verificare funzionamento del MSP430				1.5					1.5	-
Riunione di progetto				0.5					0.5	-
Prove per verificare funzionamento del CC1010				2					2	Sviluppo su MSP430
Eventuali ritardi				1					1	Problema con la reallocazione
Verifica del diagramma di flusso					0.5				0.5	Problema con la reallocazione
Sviluppo su MSP430 in base al diagramma					2				2	Problema con la reallocazione
Riunione di progetto					0.5				0.5	-
Sviluppo su MSP430 in base al diagramma					1				1	-
Implementazione su CC1010					1				1	Test su MSP430
Implementazione su CC1010						0.5			0.5	Test su MSP430
Test di funzionamento						0.5			2	Sviluppo su MSP430
Riunione di progetto						0.5			0.5	-
Verifica del diagramma di flusso						1			1	-
Sviluppo su MSP430 in base al diagramma						2.5			2.5	-
Sviluppo su MSP430 in base al diagramma							0.5		1	Implementazione su cc1010
Test e correzioni funzionalità di debug							1.5		1.5	-
Redigere presentazione							1		1	-
Redigere documentazione							2		1	-
Redigere documentazione								4	3	-
Test di funzionamento								1	1	-

Giorni previsti	40
Giorni effettivi	40

Tabella 13: Piano di lavoro

Allegati

Un cd rom contenente:

1. Il presente documento.
2. La presentazione.
3. Il poster.
4. Codice sorgente per la versione basato sul processore MSP430.
5. Codice sorgente per la scheda CCC1010.
6. Protocolli delle riunioni.
7. Raccolta di documentazione trovata sui vari componenti.
8. Esempi per il processore MSP430 e la scheda CC1010.
9. Libreria per la programmazione tramite Keil µVision.
10. File per l'installazione del programma CWGet
11. File per l'installazione del programma CrossStudio for MSP430
12. File per l'installazione del programma SoundForge
13. File per l'installazione del programma Tera Term pro

Software utilizzati

CWGet versione 1.60
SoundForge versione 6.0
CrossStudio for MSP430 versione 1.4 Build 1
Keil µVision 2 (versione del compilatore 7.08)
TeraTerm Pro versione 2.3

Scrittura del codice

Intestazione dei files
Dovrà contenere queste informazioni:
Nome del progetto
Nome del file
Autore
Indirizzo email
Data di creazione
Tipo di lavoro
Descrizione

Commenti
Vengono scritti in italiano.

Nomi Header file
In inglese con la prima lettera in maiuscolo di tutti i nomi.

Nomi funzioni
In inglese con la prima lettera in minuscolo e gli altri nomi in maiuscolo.

Nomi costanti
In inglese tutto maiuscolo separato da "_" (underscore).

Esempi:

noerror => NO_ERROR