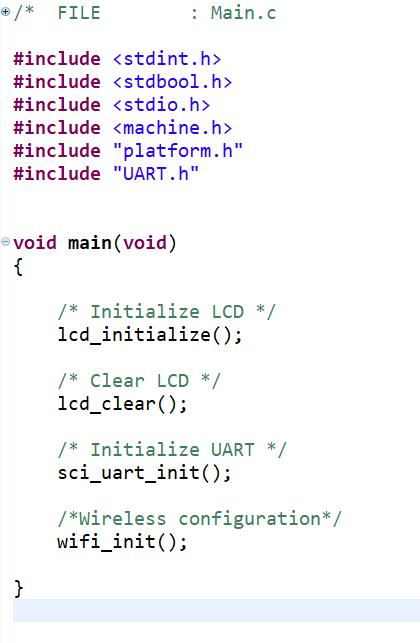
**IL CODICE**

Il codice ha due funzioni fondamentali: impostare il modulo wireless e far interpretare correttamente alla scheda Renesas i messaggi che il modulo invia. Tramite queste operazioni sarà possibile inviare comandi via wireless alla scheda, la quale comunicherà al drone di eseguire le istruzioni associate al comando inviato (l’implementazione di tali istruzioni verrà effettuata da un altro gruppo).

**Wireless.c**

Una volta lanciato, il programma entra ovviamente in questo file contenente il main, il quale è composto unicamente da chiamate alle funzioni necessarie alla configurazione della comunicazione seriale (“void sci\_uart\_init ()”) e del modulo wireless (“void wifi\_init ()”), in seguito il programma resta in attesa di eventuali comandi.

**UART.c**

Questo file è quello che si occupa della configurazione di una comunicazione seriale tra la scheda Renesas e il modulo wireless, e, di conseguenza all’interno di questo sono contenute le definizioni sia per l’interrupt di trasmissione che per quello di ricezione.

Nello specifico è definita la funzione “void sci\_uart\_init ()”, la quale imposta tutti i registri necessari alla corretta comunicazione, inizializza le code di trasmissione e ricezione (strutture dati che vengono definite nel UART.h), infine abilita l’interrupt di ricezione, cosicché, ogni qualvolta il registro di ricezione (SCI12.RDR) riceve dati, questi possono essere elaborati in maniera corretta. Non è invece necessario abilitare l’interrupt di trasmissione in quanto, a differenza della ricezione la quale potrebbe avvenire in qualsiasi momento, avviene solo quando si inviano i comandi di inizializzazione al modulo wireless, ed essendo questi inviabili solo tramite la funzione “void AT\_settings (unsigned char\* buffer)” l’interrupt viene abilitato e disabilitato contestualmente alla chiamata di questa.

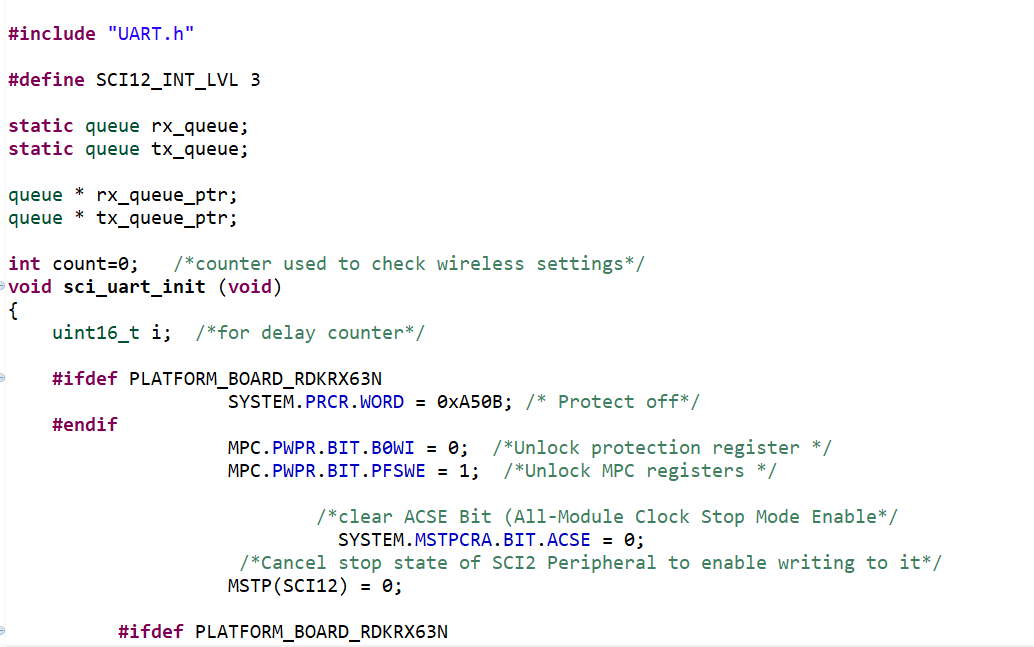
Dopodiché si definisce la funzione “void wifi\_init ()”, una funzione il cui unico compito è chiamare la funzione “void AT\_settings (unsigned char\* buffer)” per tre volte al fine di inviare al modulo wireless i comandi AT contenuti nelle variabili definite a inizio funzione.

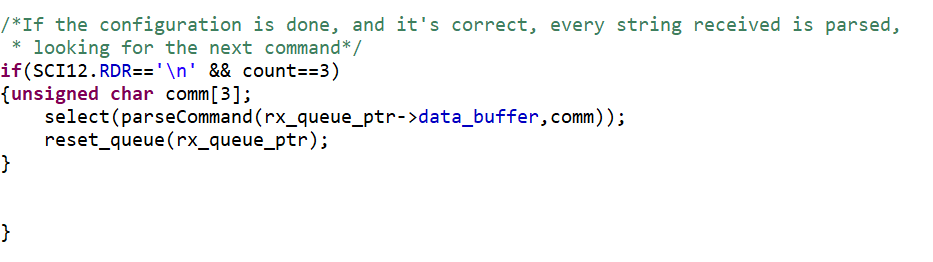
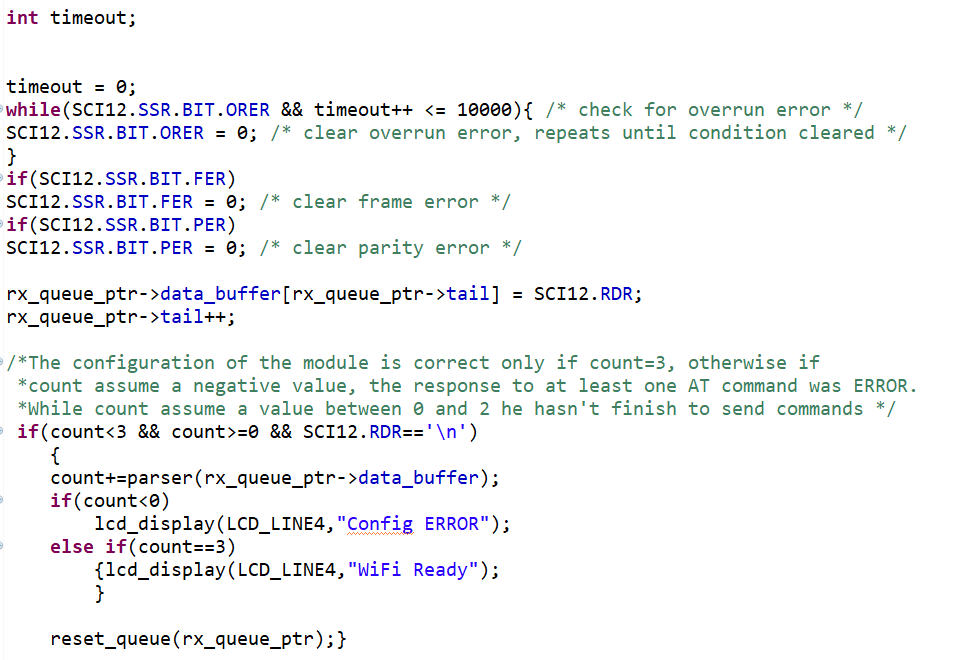
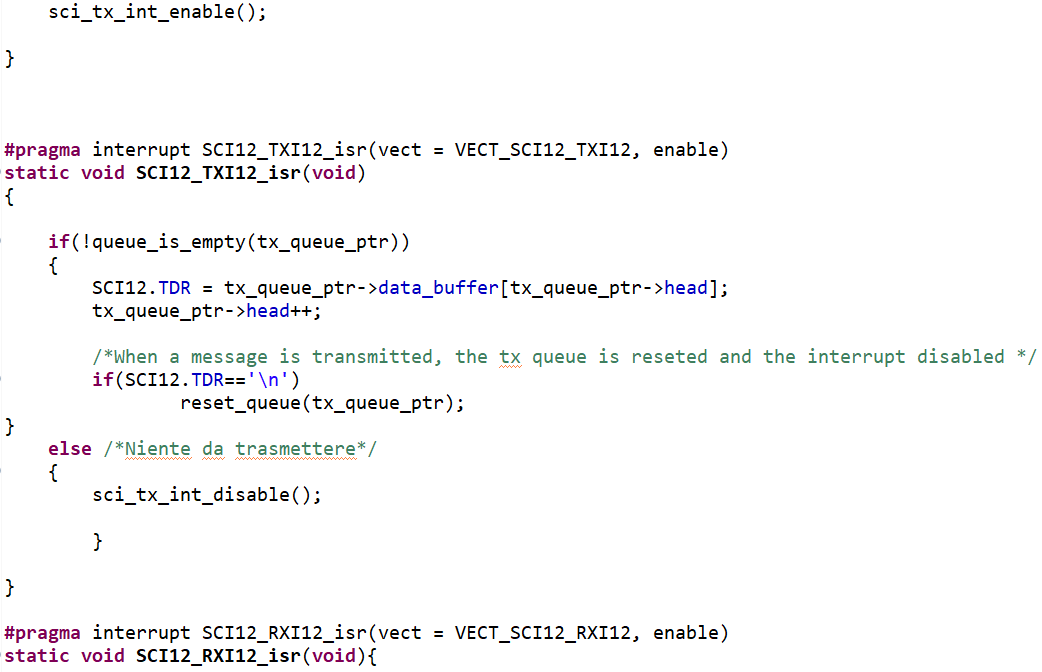
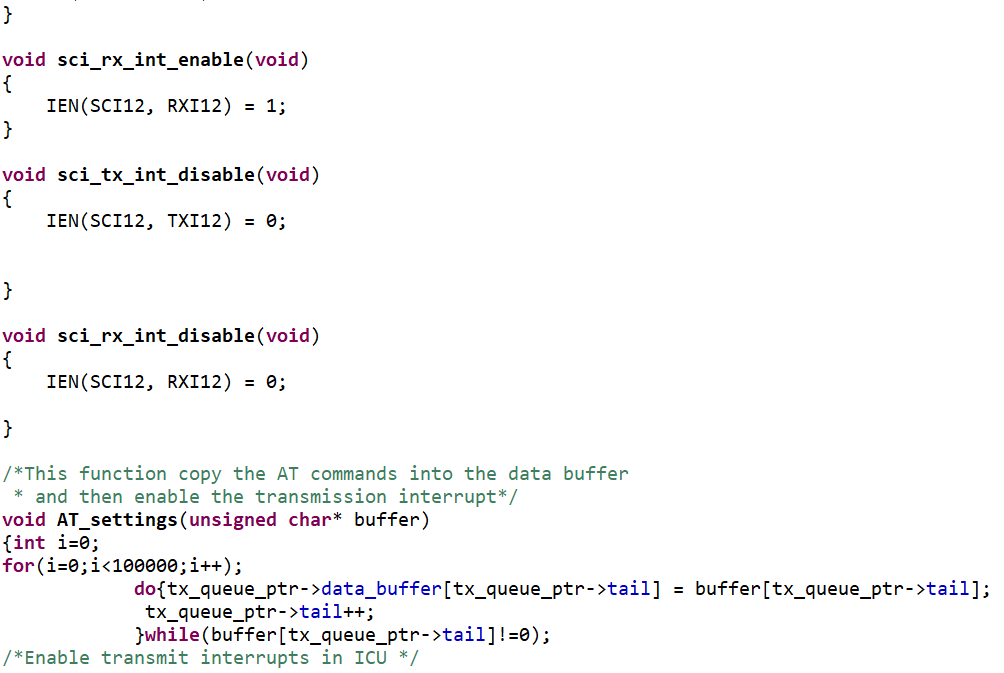
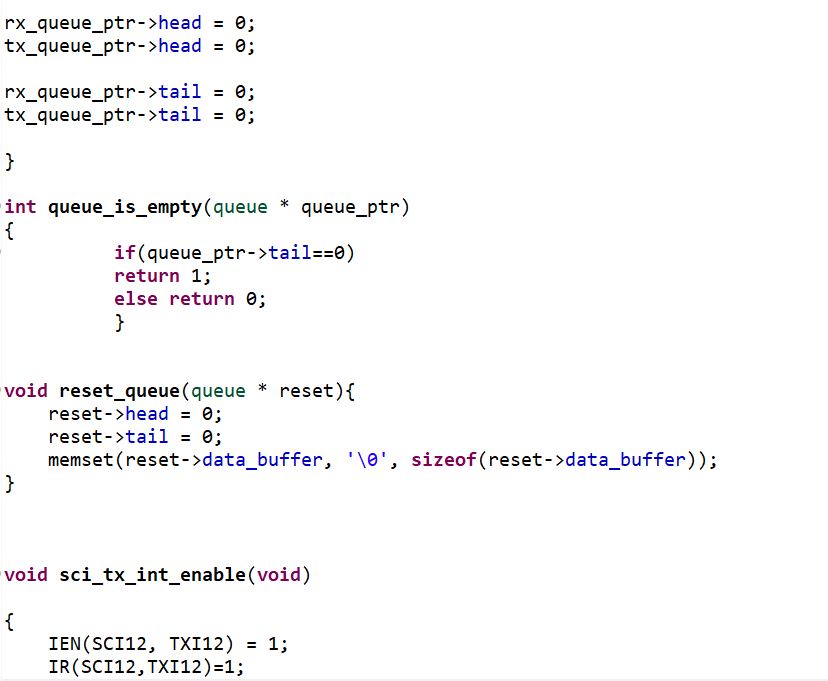
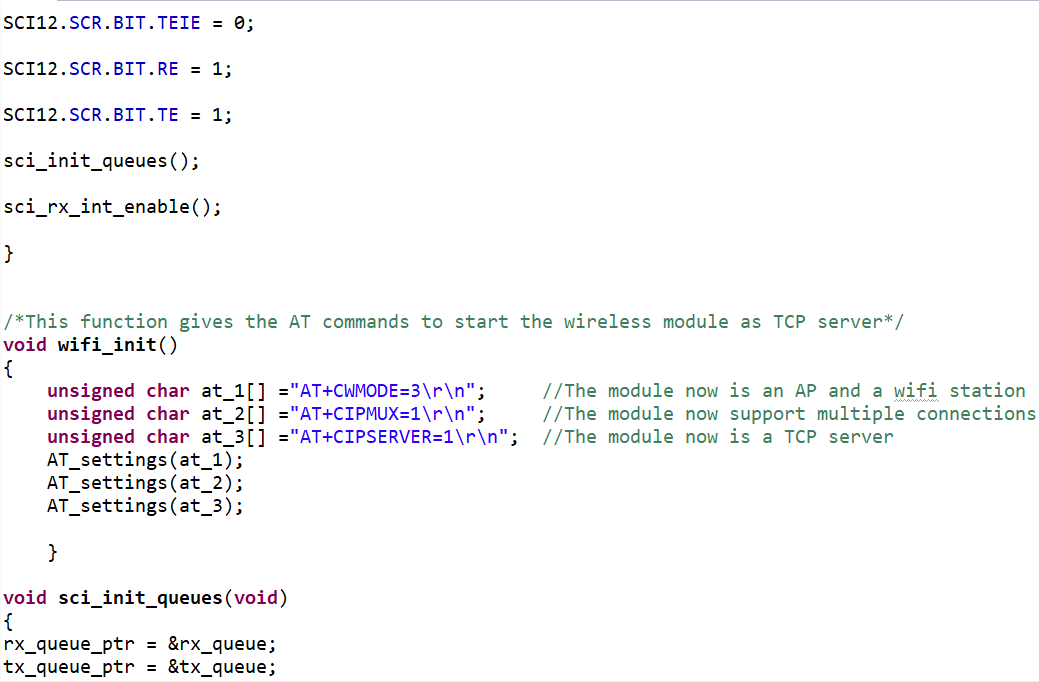
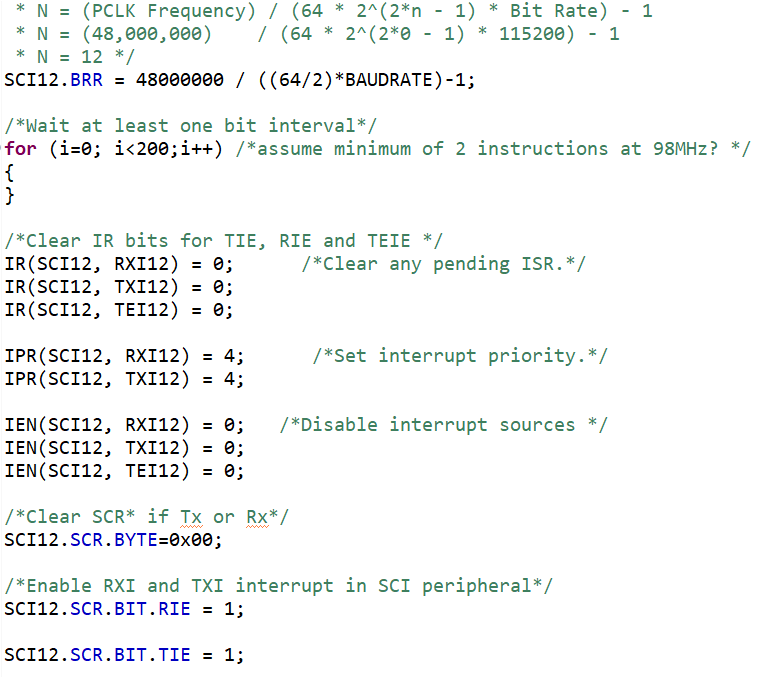
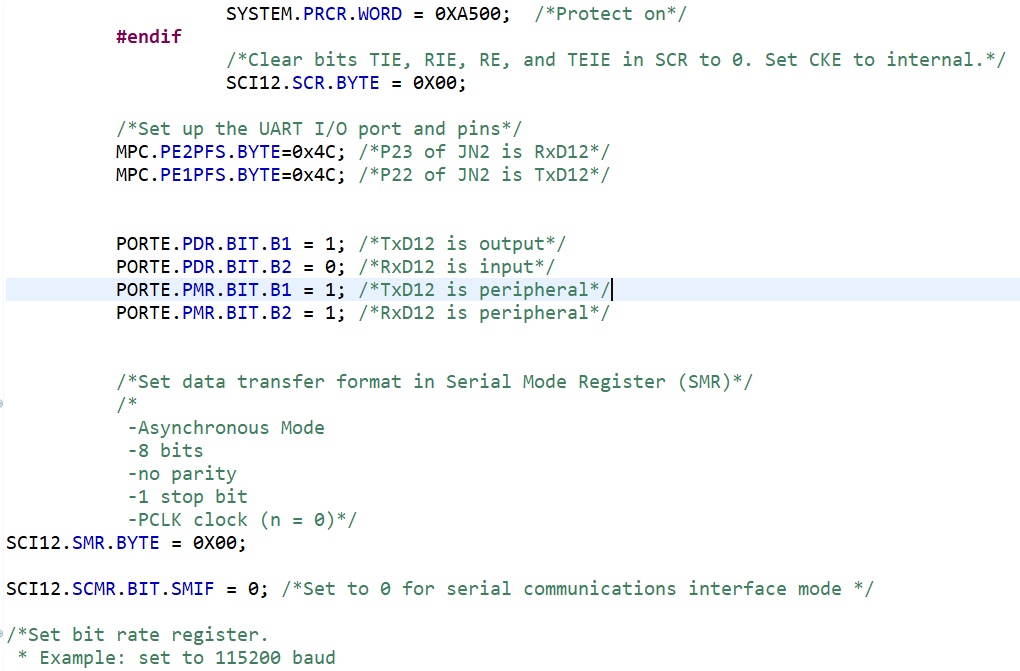
In seguito, vengono definite funzioni ausiliarie “void sci\_init\_queue (queue\* queue)”, “void reset\_queue (queue\* reset)”, “int queue\_is\_empty (queue\* queue)” le quali rispettivamente: inizializzano, svuotano e verificano se sono vuote le code di trasmissione e ricezione.

Viene poi definita la funzione “void AT\_settings (unsigned char\* buffer)”, la quale ricevuto come parametro un una stringa composta di caratteri unsigned (ovvero che nella codifica ASCII assumono valori compresi tra 0 e 255), li copia nella coda di trasmissione e attiva l’interrupt di trasmissione, il quale si occuperà concretamente dell’invio di tale stringa. È presente un ciclo for vuoto all’inizio della funzione, ciò è necessario per rallentare l’inizio dell’invio di una nuova stringa permettendo il completo invio della precedente.

Si arriva dunque alla definizione del primo interrupt, nella fattispecie quello di trasmissione, il cui funzionamento consiste nella copiatura, carattere per carattere, della stringa contenuta nella coda di trasmissione, all’interno del registro di trasmissione (SCI12.TDR) il quale essendo connesso al registro di ricezione del modulo wireless gli comunicherà il suo contenuto. Quando non ci saranno più caratteri da inviare l’interrupt di trasmissione verrà disabilitato e la coda svuotata.

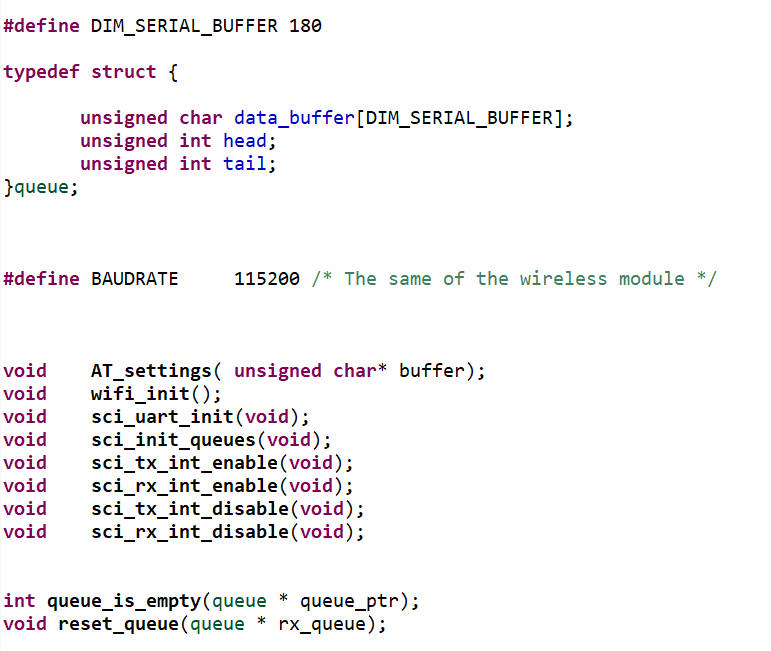
Si giunge infine all’interrupt di ricezione, il quale, messo in funzione ogni qualvolta il registro di ricezione (SCI12.RDR) riceva un carattere dal modulo wireless, esegue la funzione opposta dell’interrupt precedente, ovvero inserisce ogni carattere all’interno della coda di ricezione. Il contenuto della coda di ricezione va interpretato a questo punto al fine di stabilire il comportamento che il programma dovrà seguire a seconda del messaggio inviato, ciò è possibile innanzitutto tramite la funzione “int parser (unsigned char\* buffer)”, la quale si occupa di verificare che il modulino abbia risposto in maniera positiva ai 3 comandi AT inviati, se ciò avviene si incrementa un contatore fino a farlo giungere al valore 3 o ad un valore ampiamente negativo, nel primo caso si stamperà sullo schermo della Renesas il messaggio “Wifi Ready” a testimonianza del fatto che la comunicazione è ora possibile, nel secondo caso invece comparirà “Config Error” ad indicare che uno dei comandi inviati ha prodotto un messaggio d’errore. Tuttavia, oltre a saper verificare le risposte del modulo ai comandi di configurazione, il programma deve interpretare un comando inviato via wireless, per svolgere tale mansione viene chiamata “unsigned char\* parseCommand (unsigned char\* buf, unsigned char\* comm )”, una funzione che permette di ricavare dalla coda di ricezione solo i 3 caratteri che comporranno la nuova stringa comm, che identifica in maniera univoca il comando ricevuto dalla scheda, a questo punto la stringa ottenuta viene passata alla funzione “void select (unsigned char\* comm)”, la quale mostrerà sullo schermo della scheda Renesas, il comando inviato. La coda di ricezione viene svuotata ogni qualvolta si riceve il carattere ‘\n’, il quale identifica la chiusura del messaggio inviato dal modulo.





**UART.h**

Questo file contiene le intestazioni delle funzioni definite in UART.c, contiene inoltre la definizione del baudrate col quale la scheda comunicherà (impostato in accordo a quello utilizzato dalla scheda) e della struttura dati queue, precedentemente menzionata con la traduzione italiana coda. Un tipo di dato queue è in realtà una struct composta da un vettore di unsigned char di dimensione 180 chiamato data\_buffer, una head che indica la posizione del vettore in cui è presente il primo carattere che compone il messaggio da considerare e una tail che indica la posizione del vettore in cui è presente l’ultimo carattere.



**parsing.c**

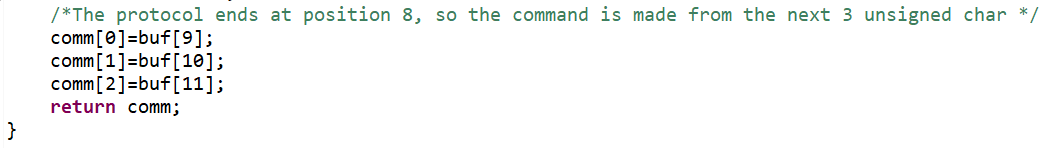
Questo file contiene le funzioni necessarie a controllare le risposte del modulo wireless che vengono chiamate in UART.c.

La funzione “int parser (unsigned char\* buffer)” si occupa di controllare le risposte del modulo ai comandi AT, questo compito viene eseguito sapendo che il modulo invierà la stringa “OK\r\n” o la stringa “ERROR\r\n” a seconda del fatto che la configurazione sia corretta o meno. Partendo da questo presupposto si passa la stringa ricevuta come parametro (il contenuto della coda di ricezione) alla funzione “int isEqual (unsigned char\* buf, unsigned char\* control, int i, unsigned char check)”, la quale verificherà se la stringa equivale a una delle due scritte in precedenza. Quindi, se la coda di ricezione contiene “OK\r\n” la funzione ritornerà 1, se invece contiene ”ERROR\r\n” essa ritornerà -100, valori utili ad aggiornare il contatore, contenuto nell’UART.c, incaricato di determinare l’esito della configurazione del modulo. Nel caso la coda di ricezione non contenga nessuna delle due stringhe il risultato della funzione è 0, risultato che non implica nulla ai fini del programma, infatti, una tale situazione, è provocata dalla ricezione di stringhe ininfluenti inviate dal modulo (queste possono essere viste nel test in cui si collega il computer al modulo wireless).

Viene poi definita la funzione, precedentemente chiamata, “int isEqual (unsigned char\* buf, unsigned char\* control, int i, unsigned char check)”, il cui scopo è determinare se due stringhe sono uguali. Questa riceve in ingresso le stringhe da confrontare, la posizione da confrontare e il carattere che chiude la stringa. La posizione è necessaria perché la funzione è implementata ricorsivamente in modo che, ad ogni chiamata venga confrontato il carattere successivo, se le due stringhe si equivalgono la funzione ritornerà 1, altrimenti 0.

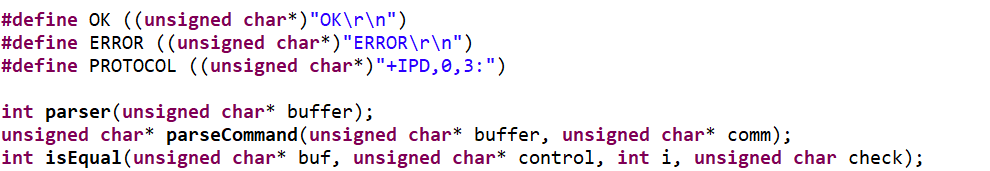
L’ultima funzione definita è “unsigned char\* parseCommand(unsigned char\* buf, unsigned char\* comm)”, che, in modo equivalente alla funzione “int parser (unsigned char\* buffer)” chiama “int isEqual(unsigned char\* buf, unsigned char\* control, int i, unsigned char check)” al fine di controllare se la coda di ricezione (passata come unsigned char\* buf) contenga la stringa “+IPD,0,3:”, ovvero l’header corretto che precede un comando restituito (il perché sia questo l’header corretto è spiegato nelle considerazioni effettuate dopo i test). Una volta appurato che l’header sia corretto possiamo salvare il comando nella variabile comm che sarà il valore restituito dalla funzione, se l’header dovesse essere errato a comm sarà assegnato il valore “NNN” ad indicare che il comando vada ignorato, tale ipotesi si verificherà successivamente ad ogni comando in quanto il modulo invia caratteri ininfluenti alla determinazione del comando oppure con comandi inviati di lunghezza diversa da 3, entrambi i casi generano header differenti come è possibile vedere dal test.





**parsing.h**

Questo file contiene unicamente le intestazioni delle funzioni definite in parsing.c e la definizione di 3 costanti corrispondenti alle stringhe da confrontare enunciate in precedenza.

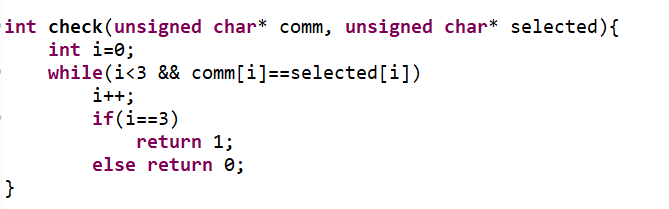
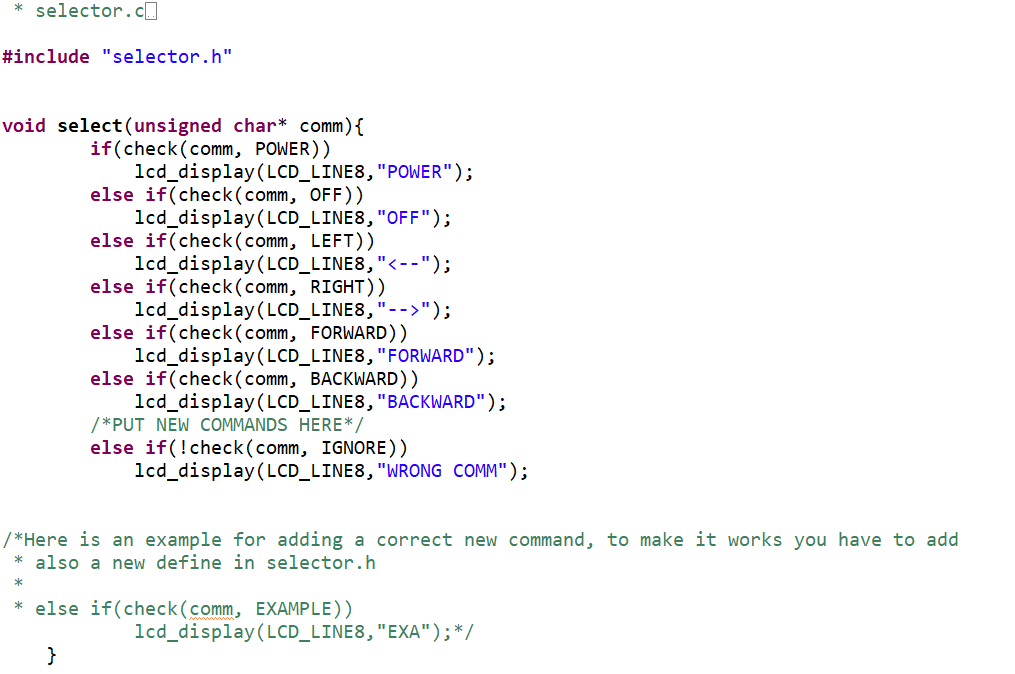


**selector.c**

Questo file contiene le funzioni necessarie a stampare a schermo il comando che è stato inviato dall’utente via wireless.

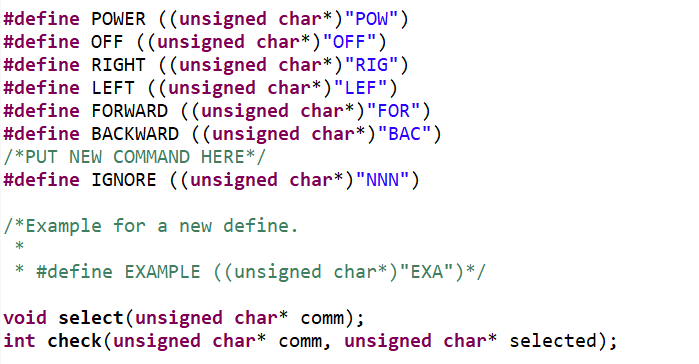
La prima funzione qui definita è “void select (unsigned char\* comm)”, questa si occupa di confrontare il comando ricavato dalla funzione parseCommand con ognuno di quelli definiti in selector.h per poi stampare a schermo quello corretto, nel caso in cui il comando sia “NNN” non verrà stampato nulla, rimarrà quindi il precedente, se invece il comando non è tra i previsti ed è diverso da “NNN” verrà stampato “WRONG COMM”.

La seconda funzione “check (unsigned char\* comm, unsigned char\* selected)” è quella che si occupa del confronto tra il comando inviato dall’utente e ognuno di quelli definiti in selector.h, ritorna 1 se sono uguali 0 altrimenti.



**selector.h**

Questo file contiene le intestazioni delle funzioni definite in selector.c e le definizioni di tutti i comandi che il programma riconosce



**CONCLUSIONI**

Lo scopo del progetto era quello di implementare un codice che permettesse di comunicare via wireless dei comandi alla scheda Renesas, essendo questo programma in grado di trasmettere dei comandi e farli comparire a schermo si può dire che l’obiettivo sia raggiunto.

È da sottolineare comunque che l’obiettivo finale dev’essere quello di far compiere determinate operazioni ad un drone contestualmente all’invio di un comando. Il codice che supporta tali operazioni è delegato ad un altro gruppo il quale potrà avvalersi della struttura fornita da questo progetto andando a sostituire la stampa a schermo, che è stata utilizzata come test del codice, con il codice da essi sviluppato.

I comandi definiti in questo codice sono 6: accensione, spegnimento, sinistra, destra, avanti, indietro, i quali dovrebbero coprire le funzioni basilari del drone. Nel caso in cui si ritenesse necessario aggiungere nuovi comandi basta aggiungere un nuovo if tra quelli presenti in selector.c e una nuova define in selector.h come mostrato nei commenti a quelle righe di codice. Si tengano presenti queste linee guida per l’introduzione di un nuovo comando: dev’essere obbligatoriamente di 3 caratteri, non possono essere utilizzate combinazioni di caratteri già presenti, non si può utilizzare la stringa “NNN” in quanto ne è riservato l’utilizzo per indicare i comandi da ignorare.

È infine di interesse per l’utente finale constatare che verrà dato un messaggio d’errore solo per comandi errati che siano di lunghezza 3, mentre negli altri casi rimarrà a schermo il comando precedente, è inoltre importante sottolineare che, nonostante il modulo sia configurato in modo da gestire connessioni simultanee di più dispositivi (condizione necessaria al fine di poterlo usare come tcp server), dal punto di vista pratico non sarà possibile inviare comandi da un secondo dispositivo, in quanto la scelta di “+IPD,0,3:” come unico header accettato implica che, un comando, inviato da un secondo dispositivo, venga ignorato, infatti tale comando anche se di lunghezza corretta, avrebbe un 1 al posto dello 0, implicando così che venga ignorato il comando.