[Agrumino Flash]

[MEMORY MANAGEMENT PER SMART GARDENING]

Simone Balloccu | [ESIT] | A.A.: 2017-2018

Obbiettivo del progetto

Lo scopo del progetto è quello di consentire la memorizzazione di dati e configurazioni sulla memoria flash del device Agrumino, facendo in modo che i dati provenienti dai sensori siano memorizzati dopo ogni ora e ritrasmessi ogni 4 ore. Si devono inoltre consentire il salvataggio sulla flash di eventuali configurazioni ricevute.

I requisti di dettaglio sulla funzionalità implementato e/o sulla architettura HW/SW verranno concordati con i docenti su proposta degli studenti.

Contesto, problematiche risolte e scelte progettuali (HW/SW)

La board di riferimento su cui è stato costruito Agrumino è l'ESP8266 (con l'aggiunta di sensori), vista durante il corso di ESIT. Nel contesto dello sviluppo del progetto AgruminoFlash sono state affrontate le seguenti problematiche con le conseguenti scelte progettuali:

- Utilizzo della memoria FLASH: inizialmente è stata effettuata una ricerca per verificare l'esistenza di progetti simili. Si è scoperto che Arduino dispone di una libreria dedita all'utilizzo della memoria EEPROM, e che l'ESP8266 emula questo tipo di memoria proprio tramite FLASH. Di conseguenza è stato possibile utilizzare la libreria senza necessità di porting. La libreria (nativa di Arduino) è reperibile al seguente indirizzo:
  [https://github.com/esp8266/Arduino/tree/master/libraries/EEPROM]
  Una volta scaricata è sufficiente importarla all'interno della libreria Agrumino, includendo nel file header ***Agrumino.h*** la riga `#include EEPROM.h`

- Strutturazione dei dati: la FLASH dell'ESP8266 è configurata come una sequenza lineare di 4096 registri da 1B completamente general purpose. Non sono quindi fornite metodologie per gestire i diversi tipi di dati. Si è deciso di implementare all'interno della libreria Agrumino un set di metodi atti a lettura e scrittura di quattro tipi di dati fondamentali: int, float, char, bool. Non si è quindi deciso di implementare nessun identificatore per il dato, in quanto avrebbe portato ad un maggiore utilizzo di memoria, né di implementare strutture atte alla memorizzazione di dati "personalizzati" (per esempio strutture) in quanto i dati della sensoristica ricadono sempre nei quattro tipi citati precedentemente, e così si è potuto appurare anche in base ad una ricerca per i vari sensori disponibili per lo smart gardening. Inoltre questa garantisce una gestione semplificata di lettura, scrittura, e gestione della memoria, ed evita all'utente di dover fare controlli a basso livello: la conseguenza è che la fase di coding si focalizza sull'utilizzo della FLASH e non sulla gestione della stessa.

- Tipi di scrittura: si è inoltre deciso di permettere la scrittura sequenziale o a specifici indirizzi indicati dall'utente. Questo permette di utilizzare la libreria in modo più libero e di riorganizzare blocchi omogenei di dati di un certo tipo in blocchi eterogenei; inoltre fornisce all'utente un modo per sfruttare più liberamente la memoria.

- Limiti dei valori dei registri: essendo i registri della memoria da 1B ciascuno, essi sono in grado di contenere solo valori compresi tra 0 e 255. Dopo l'analisi del problema si è concluso che questo limite non è un problema per tre dei quattro tipi di dato: infatti

gli interi della sensoristica sono utilizzati solo per rappresentare percentuali, ed essendo il valore 100 disponibile il problema non si è posto; per char il problema non esiste in partenza, in quanto lo standard ASCII non copre 255 caratteri nemmeno in versione estesa; infine per i booleani bastano due valori.
Il problema si è posto invece per i tipi di dati float, per cui 1B non è sufficiente: si è deciso quindi di rappresentare il dato tramite una struttura d'unione che contiene il valore in 4B, permettendo di recuperare sempre l'intero valore. Questa scelta ha portato anche a diverse politiche di gestione dei dati, che vengono spiegate nel punto successivo.

- RESET dovuto al DeepSleep: una delle caratteristiche che rende la ESP8266 particolarmente adatta all'Internet Of Things è la sua capacità di esercitare un notevole risparmio energetico grazie alla disponibilità di diverse modalità di "sleep". La più efficiente è la Deep Sleep in cui la board viene di fatto disattivata, portando il suo consumo ad un valore di circa 20 microAmpere, come mostrato nella seguente tabella riassuntiva:

Item	Modem-sleep	Light-sleep	Deep-sleep
Wi-Fi	OFF	OFF	OFF
System clock	ON	OFF	OFF
RTC	ON	ON	ON
CPU	ON	Pending	OFF
Substrate current	15 mA	0.4 mA	~ 20 µA

Tabella riassuntiva dei consumi della ESP8266 nelle varie modalità di sleep

Per risvegliare la board è però necessario l'equivalente del reset della board, operazione che porta a far ripartire lo sketch dall'inizio, perdendo quindi eventuali variabili globali. Questo problema è stato risolto mediante l'utilizzo dei registri dedicati DIRTY e START_ADDRESS, il cui utilizzo viene spiegato nel punto successivo.

- Gestione dei registri e registri riservati: una volta inseriti i metodi necessari a scrittura e lettura dei dati si sono resi necessari altri metodi per la gestione dello spazio e delle varie informazioni di memoria. Per questo sono stati inseriti i metodi per ripulire i registri inutilizzati (che discriminano il tipo di dato in base alla loro memory size, ossia 1B o 4B), e per la verifica della disponibilità di un particolare registro. Sono inoltre stati riservati alcuni B per la creazione di alcuni registri "riservati", alcuni dei quali non accessibili all'utente, che conservano informazioni sulla memoria. I registri sono i seguenti:
  - DIRTY: questo registro contiene un booleano, che comunica all'utente se la memoria dell'Agrumino è "sporca", ossia se contiene dati non ancora pushati sul server.
  - LASTFREEADD: "last free address", indica il numero dell'ultimo registro non utilizzato.
  - FREE_MEMORY: indica la memoria libera in byte
  - START_ADDRESS: questo registro si è reso necessario a causa del reset che viene eseguito al risveglio dal Deep Sleep. Esso indica in quale registro inizi il blocco di dati che per l'utente è di interesse leggere, e pertanto va gestito e aggiornato manualmente.

- HOURS: questo registro contiene il numero di ore passate dall'ultimo push dei dati eseguito. Permette quindi di utilizzare una qualsiasi variabile definita dentro lo sketch della board per eseguire il confronto e capire se bisogna pushare i dati, anche a fronte di un RST.

- USERSPACE: contiene un valore fisso che indica il primo registro utilizzabile dall'utente. Viene utilizzato per evitare la scrittura in registri riservati.

- MAX_MEMORY: contiene il valore fisso di 4096MB, ossia la memoria massima del chip, ed è stato introdotto semplicemente per leggibilità del codice.

Va notato che la validità dei dati contenuti negli indirizzi LASTFREEADD e FREE_MEMORY viene compromessa se l'utente effettua scritture non sequenziali: infatti questi registri sono pensati per essere utilizzati solo scrivendo blocchi contigui di dati, e il loro utilizzo nel caso questa condizione mancasse sarebbe impossibile

Verifica funzionale del progetto

Per valutare il progetto sono stati prodotti due sketch distinti:

1) AgruminoFlash.ino: questo sketch è stato concepito per dare una breve demo dei metodi disponibili, e la sua esecuzione porta all'inizializzazione della memoria, seguita dalla scrittura/lettura di diversi blocchi omogenei ed eterogenei di dati. In aggiunta è presente la sovrascrittura (scrittura non sequenziale) di blocchi già scritti per dimostrare l'utilizzo di questa funzionalità. L'intero sketch è corredato da un "logger" che mostra come le statistiche di memoria varino durante le operazioni.

2) AgruminoFlashWithSensors.ino: questo secondo sketch è più orientato all'utilizzo di Agrumino e di fatto esegue la routine di task richiesta dal progetto.
Di seguito la spiegazione passo passo del codice:

   Nel setup viene controllato se la memoria è "dirty", e settato un booleano di conseguenza. Si noti che la memoria viene abilitata all'utilizzo con il metodo enableMemory() che differisce da initializeMemory() in quanto non la cancella:

```
void setup()
{
  //initializing and powering the board, then enabling the memory
  Serial.begin(115200);
  agrumino.setup();
  agrumino.turnBoardOn();
  agrumino.enableMemory();

  //checking if the memory is "dirty" (A.K.A if there's some data to push)
  if(agrumino.getDirty()==0)
    wrote=false; //not dirty
  else
     wrote=true; //dirty

}
```

   Eseguito questo check parte il loop che dipende dal valore del flag dirty (e dal booleano wrote impostato di conseguenza):

a. **Se la memoria è DIRTY e sono passate abbastanza ore dall'ultima lettura dei dati in flash (check ottenuto tramite il controllo del valore del registro HOURS):** allora si prosegue all'estrazione di tutti i dati registrati: ciò è fatto utilizzato START_ADDRESS come registro di shift, semplicemente andando avanti di tanti B quanti sono i dati letti. Quindi se il primo dato letto è un FLOAT si leggerà un float da START_ADDRESS, dopodiché se ne incrementerà il valore di 3 (poiché il primo dei 4B è proprio START_ADDRESS). Questo procedimento viene iterato per tutti i dati, come si può vedere dal seguente screenshot:

```
//getting the address of every sensor data, based on the offset of its data type
attUSBADD = agrumino.getStartAddress();
chargADD = attUSBADD+1 ;
buttonADD = chargADD+1;
tempADD = buttonADD+1;
soilADD = tempADD+4;
illADD = soilADD+4;
voltADD = illADD+4;
battLVLADD = voltADD+4;
```

Terminata questa operazione si richiama initializeMemory() che resetta la memoria (e quindi anche i valori di DIRTY e HOURS), rendendo la board pronta a ricevere nuovi dati.

b. **Se la memoria è DIRTY e non sono passate abbastanza ore:** viene reperito l'ultimo indirizzo disponibile e si scrive un nuovo blocco di dati, nello screenshot qui sotto vengono messe in evidenza le prime due scritture.

```
//reading the different data, and storing them into the flash memory. Also backupping the registers of every data
attUSBADD = agrumino.getLastAvaiableAddress();
agrumino.boolWrite(isAttachedToUSB);
Serial.println("wrote attacchedUSB (bool) in REG_"+String(attUSBADD));

chargADD = agrumino.getLastAvaiableAddress();
agrumino.boolWrite(isBatteryCharging);
Serial.println("wrote chargBatt (bool) in REG_"+String(chargADD));
```

Al termine di questa operazione si incrementa inoltre il valore del registro HOURS, per segnalare la scrittura eseguita:

```
//increasing the hours counter
agrumino.incrHours();
```

c. **Se la memoria non è DIRTY:** stessa routine di b.

Al termine di una delle operazioni elencate sopra la board viene messa in Deep Sleep per il tempo specificato (nell'esempio 1 ora). Si noti che il flag DIRTY permette di evitare la re-inizializzazione della memoria al risveglio (che porterebbe alla cancellazione delle informazioni conservate in flash): semplicemente questa viene ri-abilitata per permettere l'accesso.

Problematiche riscontrate:

Nel corso dello sviluppo del progetto sono state riscontrate le seguenti problematiche HW, che non si è potuto risolvere:

- Difetto delle resistenze: come indicato dai responsabili di Lifely durante la comunicazione inter-progetto le board fornite presentano un difetto inerente alla circuiteria, ed in particolare alle resistenze, che causa spesso l'impossibilità di caricare lo sketch all'interno della board. Ciò è risolvibile imparando meccanicamente la seguente procedura:
    1. Sospendere l'alimentazione alla board (cavo e jumper)
    2. Tenere premuto il tasto di RST
    3. Ricollegare il cavo (senza rilasciare il tasto di RST)
    4. Avviare il caricamento dello sketch e rilasciare il tasto di RST nell'istante che intercorre tra la compilazione e l'inizio del caricamento (ossia quello che intercorre tra la scritta arancione e quella bianca nel logger di compilazione dell'IDE)

  Ciò non è comunque garanzia di funzionamento in quanto il tasto di RST va rilasciato in un lasso di tempo ben preciso all'interno di quell'istante, e ciò ha portato ad un notevole rallentamento del progetto. Si consiglia caldamente ai produttori di Agrumino di risolvere la problematica poiché rende molto complesso lo sviluppo sulla piattaforma.

- Degradazione della board: si è notato come un progressivo utilizzo della board abbia portato ad una difficoltà sempre maggiore da parte della stessa di funzionare, a livello delle funzioni più basilari (compreso l'accensione col jumper e il riconoscimento via cavo). Non è chiaro se il problema sia dovuto all'utilizzo della FLASH ma si esclude questa ipotesi in quanto si è verificato di non essere andati a scrivere in registri riservati; inoltre la memoria FLASH dell'ESP8266 (che è pressochè identica a quella di Agrumino) è garantita per diecimila letture e scrittura, un numero decisamente inferiore a quelle eseguite nell'arco dello svolgimento del processo. Inizialmente il problema si risolve ricorrendo ad un tool presente per Windows e scritto in Python chiama ESPtool, ma con l'andare del tempo le board presentano lo stesso problema sempre più spesso, arrivando alla quasi impossibilità di utilizzo: non è possibile accenderla o farla riconoscere a nessun dispositivo per caricarvi uno sketch.

Istruzioni per riuso e riproduzione:

Il riuso del codice è immediato, in quanto è stata modificata la libreria di Agrumino, senza modifiche da parte dei metodi già presenti in essa (campionamento sensori etc..): questo significa che si può sostituire la libreria con quella modificata e scrivere gli sketch che utilizzano la flash, e al contempo mantenere funzionanti quelli precedentemente scritti. Per quanto riguarda la riproduzione sia gli sketch di DEMO che il codice su GIT è completamente commentato, per cui il codice risulta immediatamente chiaro comprensibile.

Si fa inoltre presente che il codice è perfettamente funzionante su ESP8266: tutti i metodi di scrittura, lettura, gestione memoria etc.. sono utilizzabili immediatamente, ma si consiglia di esportarli dalla libreria di Agrumino in quanto essa contiene molti metodi non utilizzabili su ESP8266, come per esempio quelli relativi alla sensoristica, che non è normalmente presente sulla board vista durante il corso.

Link al repository Github:

Al seguente link sono disponibili la libreria modificata (inclusa la libreria EEPROM) e gli sketch di esempio sviluppati:

https://github.com/uccollab/AgruminoFlash