Progetto di Sistemi Operativi: **Arduino SmartHouse**

Specifiche di progetto

Il progetto prevede la realizzazione di due programmi Arduino+PC e supporta le seguenti specifiche:

- Comunicazione seriale **Controller**(Arduino Board)**-Client**(PC) interrupt-driven
- Protocollo binario con verifica integrità dei dati (checksum)
- 8 Switch, pin A8-A15
- 8 canali di ingresso ADC (un timer ne gestisce la lettura sequenziale), pin A0-A7
- 8 uscite digitali (pwm), pin 2,3,5,6,7,8,11,12
- 1 uscita digitale (senza pwm), pin 13
- salvataggio, o caricamento di una configurazione precedente, in un file binario

Il protocollo di comunicazione (1): PC -> Arduino

Protocollo binario ad 8 byte. I pacchetti inviati dal client al controller sono così organizzati:

- 1° byte: **comando** impartito alla board (in esadecimale: 0x01, 0x02,...). Sono supportati 6 comandi.
- 2° byte: **checksum**. E' calcolata sommando tutti i byte che compongono il pacchetto, con eventuale somma del riporto se la somma complessiva è maggiore di un byte.
- 3° byte: **size**. Alcuni comandi richiedono l'invio di dati per l'esecuzione. Il campo size specifica quanti dei byte successivi contengono dati.
- 4°-8° byte: **dati**. Utilizzati in base al comando specifico.

Il protocollo di comunicazione (2): Arduino -> PC

I pacchetti inviati dal controller al client sono così suddivisi:

- 1° byte: **segnale di risposta**. Vi solo due tipi di risposta: Ack e Nack.
- 2° byte: **checksum**. E' calcolata sommando tutti i byte che compongono il pacchetto, con eventuale somma del riporto se la somma complessiva è maggiore di un byte.
- 3° byte: **ultimo comando ricevuto**. Ogni pacchetto di risposta specifica a quale comando si riferisce.
- 4° byte **size**. Alcuni comandi, come ad esempio la funzione per leggere il valore di un canale ADC, richiedono l'invio di uno o più dati. Size specifica la dimensione dei dati inviati.
- 5°-8° byte: **dati**. Utilizzati in base al comando specifico.

Il protocollo di comunicazione (3): funzionamento

La board Arduino è in ascolto sulla linea seriale e invia solo pacchetti di risposta. Il client invia pacchetti comando. Ogni pacchetto inviato tra i due interlocutori è sottoposto a due elaborazioni. Al campo size sarebbero sufficienti 3 bit per gestire i dati(5 bit massimo), ma ne utilizza 5 invece, in modo da garantire la possibilità di estendere il protocollo con futuri aggiornamenti. Ai 5 bit della size vengono aggiunti in bit stealing 3 bit di controllo(rappresentano la somma del numero di bit 1). Infine viene calcolata la checksum sull'intero pacchetto.

Quando un pacchetto comando è ricevuto da Arduino controlla che il comando sia valido ed in caso di esito negativo invia un Nack, altrimenti lo elabora. Quando viene ricevuto un pacchetto risposta, PC controlla che il segnale di risposta sia valido e che il pacchetto faccia riferimento all'ultimo comando inviato.

Entrambi i controlli proseguono con la validazione del campo size ed infine della checksum. Il fallimento di uno di questi di questi 3 passaggi comporta per Arduino la mancata elaborazione del pacchetto, per PC il reinvio del pacchetto per 10 volte (dopo 10 fallimenti consecutivi viene mostrato un messaggio di errore).

L’applicazione client: OnPC_client.c

All’avvio dell’applicazione, se è presente un file con una configurazione precedentemente salvata viene chiesto all’utente se desidera caricare tale configurazione. Se la ricerca non va a buon fine o l’utente non acconsente, viene richiesto di inserire un nome per il controller che gestirà i vari device collegati e contestualmente il controller viene resettato. Il comando load non carica nulla sulla scheda Arduino, ma carica solo in locale(sul client) la situazione precedente alla chiusura (associazione pin-dispositivi) in modo da riprendere la configurazione da dove la si era lasciata. Sulla board Arduino continua a girare l’ultima configurazione impostata.

Successivamente è possibile installare dispositivi che utilizzano la pwm o i canali ADC tramite un menù. Per effettuare una scelta è sufficiente inserire da uno a tre caratteri numerici da tastiera. In qualsiasi momento è possibile terminare l’applicazione (verrà chiesto se si desidera salvare la configurazione corrente). Una configurazione è l’associazione tra porte disponibili(8pwm + 8adc) e dispositivi nonché il nome scelto per il controller.

Ad esempio l’utente può effettuare il setup di led collegati ai pin dedicati alle uscite digitali impostandone la luminosità. Le uscite sui pin 7,8,11,12 sono controllabili tramite gli switch (due switch per ogni pin) con i quali si può aumentare o diminuire il livello di luminosità. L’utente può inoltre installare dei fotoresistori sui pin A0-A7 e “collegarli” ad un pin (pwm o non) per un controllo automatizzato della luminosità.

L’applicazione controller: OnBoard_controller.c

La board Arduino all’avvio può solo ricevere ed è in sleep (idle_mode). Appena un pacchetto completo di 8 byte è ricevuto, ne verifica la correttezza ed eventualmente procede ad eseguire il comando richiesto, terminato il quale assembla ed invia un pacchetto di risposta.

Le porte dedicate alle uscite digitali con pwm sono gestite dai Timer 1,3 e 4 sfruttando direttamente l’hardware della scheda (interrupt driven).

Gli switch che pilotano le uscite pwm pin7,8,11 e 12 sono anch’essi interrupt driven utilizzando l’ISR di sistema PCINT2.

I pin di ingresso ADC vengono letti ciclicamente(qualche secondo) dalla scheda utilizzando Timer 2. Essendo selezionabili solo tramite MUX vengono letti sequenzialmente, questa operazione risulta quindi bloccante. I vari timer vengono disattivati quando tutti i dispositivi ad essi collegati sono stati rimossi tramite le apposite funzioni.

Ulteriori informazioni

L’applicazione viene eseguita con il comando:

make -f MakeWriter execute

E’ possibile avviare l’applicazione client in modalità debug tramite la chiamata al suo makefile:

make -f MakeWriter CFLAGS=-DDEBUG

La modalità debug permette di visualizzare il contenuto di tutti i pacchetti scambiati e altre variabili utili nelle operazione di debug.

E’ possibile, sempre attraverso il makefile dedicato, utilizzare il tool di profiling Valgrind per testare eventuali memory leaks:

make -f MakeWriter memtest