Bergia Stefano 276124 13/01/2020

EXTRAPOINT\_02 APPLICATION NOTE

Nota: ogni variabile e funzione verrà spiegata nella sezione relativa al file in cui essa è definita, a prescindere dalla presenza della clausola extern

* main/sample.c

Variabili:

* + Unsigned char status\_byte: ogni bit di questa variabile viene usata come un flag per segnalare un particolare stato del sistema

0 → 1 per ascensore occupato, 0 per ascensore libero

1 → 1 per ascensore a metà strada tra i piani, 0 per ascensore fermo ad un piano

2 → 1 se l'ascensore sta andando verso l'alto, 0 se sta andando verso il basso (ha significato solo se il bit precedente vale 1)

3 → 1 ascensore al primo piano, 0 ascensore al piano terra (ha significato solo se il bit 1 vale 0)

4 → 1 timer spostamento manuale abilitato, 0 se disabilitato

5 → 1 l'ascensore si sta muovendo in maniera autonoma, 0 ascensore controllato da joystick

6 → 1 stato di allarme

7 → 1 per blinking a 5Hz, 0 per blinking a 2Hz o a 4Hz

* + unsigned int position: può assumere un valore da 0 a 80 e rappresenta la posizione sulla rampa dell’ascensore (ho supposto che l’ascensore si muova a step di 10 cm)
  + unsigned int speedup: coefficiente per velocizzare il debug software
  + unsigned int maintenance: se vale 1 si è in modalità di manutenzione e tutte le azioni al di fuori di quelle legate alla selezione della nota devono essere disattivate
  + unsigned int Note1, unsigned int Note2: possono assumere un valore da 0 a 12 che rappresenta una delle 13 note possibili per la sirena
  + char \* labels[]:vettore contenete 13 etichette da visualizzare sullo schermo in base alla note scelte

Funzioni:

* int main(): esegue l’inizializzazione della scheda, delle variabili, e delle periferiche (inclusa la calibrazione del touch screen) e porta la scheda in stato di risparmio energentico
* void MoveToFloor(char floor): utilizzata per far partire il movimento automatico dell’ascensore in seguito alla pressione dei pulsanti (a patto che sia permesso dallo stato corrente del sistema), imposta correttamente i flag di stato e fa partire il timer0, il quale si occuperà effettivamente della temporizzazione del movimento(vedi dopo)
* void ResetTimerProxy(int n):resetta e disabilita il timer n
* void LED\_On\_Proxy (unsigned int num: accende il led N
* void LED\_Off\_Proxy (unsigned int num: spegne il led N
* void EnableTimerReposition(): abilita il timer 2, usato per movimento manuale
* void EnableTimerWait(): abilita il timer 1 che si occupa dell’attesa di 1 minuto all’arrivo dell’ascensore ad un piano
* void EnableTimerAlarm(int min):abilita il timer 1 che si occupa in questo caso di attendere 1 minuto per far entrare il sistema in stato di allarme (movimento manuale senza input per 1 minuto)
* void DrawPanelMaintenance(): Disegna sullo schermo la schermata iniziale per entrare in modalità Maintenance
* void DrawPanel(int pressed): disegna la schermata di Maintenance per selezionare la nota (pressed ha un valore da 0 a 4 e indica quale elemento della schermata evidenziare,0 per non evidenziare nulla)
* void FocusElement(int pressed,int note): evidenzia l’elemento indicato dal parametro pressed (se assume un valore negativo rimuove l’evidenziazione), note indica l’indice del vettore labels contente il testo da stampare sullo schermo corrispondente alla nota
* button\_EINT/IRQ\_button.h  
  contiene gli handler dei tre pulsanti, tutti presentano debouncing
* RIT/IRQ\_RIT.c

Variabili:

* int down1,down2,down0 : variabili utilizzate per la gestione del bouncing dei pulsanti
* unsigned int count\_2s: variabile utilizzata per attendere 2 secondi prima di confermare lo stato di allarme o per capire se si è rilasciato il pulsante prima dei 2 secondi
* int down0\_supp: variabile utilizzata per resettare count\_2s
* int sel: flag utilizzato per capire quale nota si vuole modificare
* int tmpNote1, tmpNote2: variabili per memorizzare le note selezionate prima che essi vengano salvati

Funzioni:

* void RIT\_IRQHandler (void):handler del timer RIT, eseguito ogni 50 ms, si occupa di gestire tutti gli input del sistema, in particolare vengono verificate in ordine:
  + pressione pulsante del joystick  
    permette, se premuto in stato di ascensore libero, di passare in stato di movimento manuale. Fa partire il timer per la gestione dell’allarme (vedi dopo)
  + direzione joystick in basso

permette se premuto di far partire il timer 2 che si occupa di spostare l’ascensore verso il basso (se il timer è già avviato non viene fatto ripartire). Inoltre riavvia il timer di allarme

* + direzione joystick in alto

come sopra ma verso l’alto

* + nessun input dal joystick

stopppa il blinking del led 7 a 2 Hz e lo accende fisso

* + pulsante KEY1

se l’ascensore è libero richiama MoveToFloor (0), gestendo lo stato e i led

* + pulsante KEY2

se l’ascensore è libero richiama MoveToFloor (1) , gestendo lo stato e i led

* + pulsante INT0

quando viene premuto (se non si è in stato di allarme) viene resettato count\_2s se down0\_supp è settato, altrimenti count\_2s viene incrementato e si entra in uno stato di preallarme (led e sirena attivi ma ascensore non richiamabile) Nel momento in cui count\_2s raggiunge 40 (in realtà 20 a causa del lag dovuto all’elaborazione) sono passati 2s e si passa nello stato di allarme vero e proprio (ascensore marcato come libero e quindi richiamabile).

Quando il pulsante viene rilasciato e count\_2s non ha raggiunto il valore 20 si annulla lo stato di allarme, e in ogni caso si setta la variabile down0\_supp per resettare il contatore alla prossima pressione

Quando il pulsante viene ripremuto si esce dallo stato di allarme.

* + inizio della conversione ADC per recuperare il valore del potenziometro
  + pulsante touch screen per accedere alla modalità maintenance

se l’ascensore è libero si bloccano tutti gli input diversi dal touch screen e si entra in stato di maintenance

* + zone touch screen per selezionare la Nota da modificare

quando si rileva il touch in una delle due zone, sel viene impostato a 0 o a 1 per capire quale delle due note si deve modificare , la zona selezionata viene evidenziata con FocusElement()

* + pulsante touch SAVE per salvare la modifica delle note

copia i valori di tmpNote1 e tmpNote2 dentro Note1 e Note2 (vedi IRQ\_ADC.c),

inoltre il pulsante viene evidenziato finché esso è premuto

* + pulsante touch QUIT per uscire dalla modalità maintenance
* timer/IRQ\_timer.c

Variabili:

* + int frequency[13]:vettore di 13 elementi contenete i coefficienti da assegnare ai timer in modo che emettano le 13 differenti note
  + int SinTable[]: sinusoide campionata da passare al DAC per mettere suoni

Funzioni:

* void TIMER0\_IRQHandler (): handler del timer 0, presenta 2 funzioni, la prima è quella di inviare i valori della sinusoide campionata al DAC per emettere suoni, la seconda è quella di gestire il movimento automatico dell’ascensore. Per quest’ultimo caso il timer deve essere inizializzato in modo che il movimento avvenga a 4Km/h, ovvero a 1,11 m/s, per cui il tempo totale percorrenza è 8m / 1,11 m/s=7,2 s. Riprendendo l’ipotesi di avere un movimento minimo di 10 cm bisognerà attendere 80 step del timer. Per cui il periodo del timer deve essere 7,2s/80=0.09s e il coefficiente 0,09s \* 25 Mhz=0x225510. L’handler del timer ogni volta che viene richiamato incrementa o decrementa la posizione, avvia il timer 3 per la gestione del blinking a 2 Hz e verifica l’arrivo al piano (anche in questo caso aggiorna in modo appropriato lo stato e avvia il blinking a 5Hz per 2 secondi)
* void TIMER1\_IRQHandler (void): handler del timer 1, si occupa di gestire il sistema dopo che l’ascendore è arrrrivato al piano ed è rimasto occupato per un minuto (l’handler libera l’ascensore) e porta il sistema in stato di allarme dopo che per un minuto non si ricevono input dal joystick (e l’ascensore è in modalità di movimento manuale)
* void TIMER2\_IRQHandler (void): handler del timer 2, si occupa di gestire la temporizzazione del movimento manuale, rimane abilitato finché si ricevono input dal joystick o finché non si arriva ad uno dei due piani, si comporta in modo simile all’handler del timer 0, gestisce anche il blinking a 2Hz a 5Hz
* void TIMER3\_IRQHandler (void: handler del timer 3, si occupa di gestire il blinking dei led e l’alternanza delle note della sirena
* void Blink5Hz(),void Blink4Hz(),void Blink5Hz(): inizializzano correttamente il timer3 per effettuare i 3 tipi di blinking
  + void StartBell(int freq): inizializza il timer 0 in modo che emetta un nota con una certa frequenza
* adc/IRQ\_adc.c

Variabili

* + unsigned short AD\_current: valore corrente letto dall’adc
  + unsigned short AD\_last: ultimo valore letto dall’adc

Funzioni

* void ADC\_IRQHandler(void): gestisce la lettura del valore del potenziometro, quest’ultimo viene considerato solo nel caso in cui si è in maintenance mode (variabile maintenance=1).

In particolare il valore viene campionato in un range da 0 a 12, ognuno di questi valori corrisponde ad un indice possibile per il vettore frequency, il quale contiene i contatori da assegnare al timer in modo che la sinusoide campionata venga riprodotta ad una certa frequenza. Il valore letto viene assegnato alla variabile tmpNote1 se nel touch screen abbiamo selezionato la parte superiore dello schermo, oppure tmpNote2 se abbiamo selezionato la seconda. (Nel momento in cui sul touch screen si selezionerà il pulsante SAVE i valori di tmpNote1 e tmpNote2 verranno copiati in Note1 e Note2 che sono le variabili che effettivamente memorizzano l’indice delle note che verranno riprodotte)