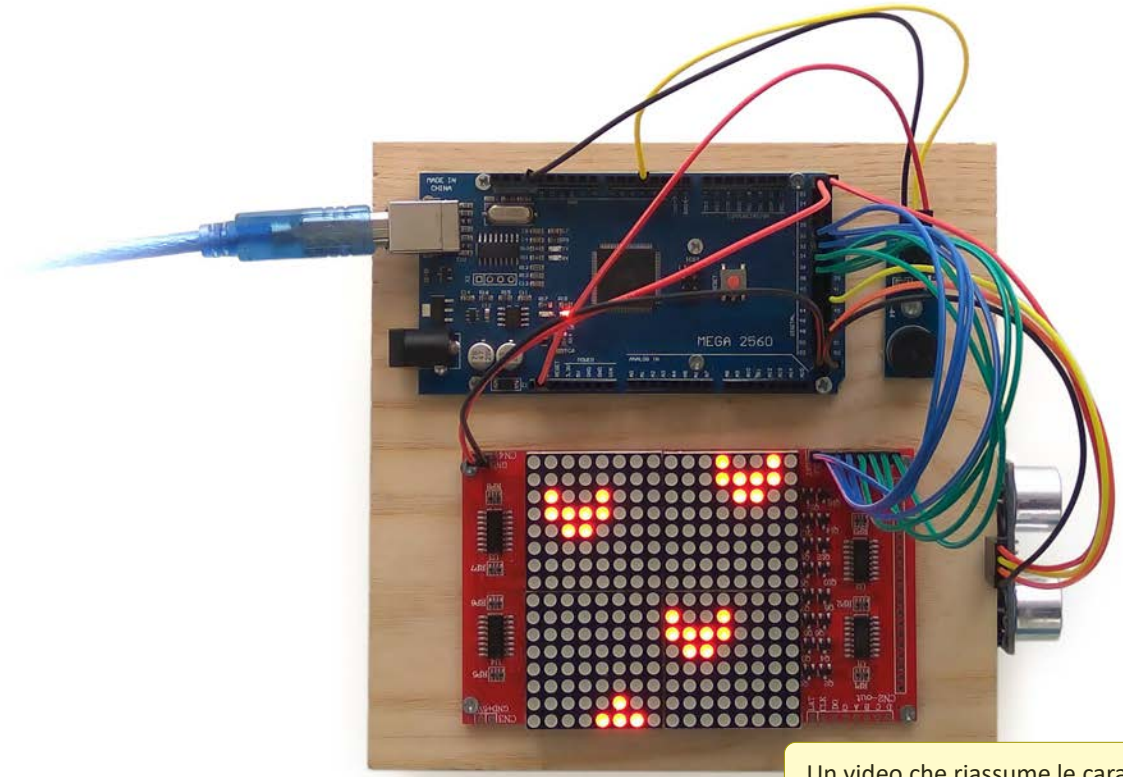
Programmazione in linguaggio assemblativo

Realizzazione di un piccolo videogioco tramite microcontrollore AVR

Progetto di ***architettura degli elaboratori*** *2016/2017 di* ***Francesco Zoccheddu***
Prof. ***Diego Reforgiato***

Introduzione

Come progetto finale per il corso di architettura degli elaboratori, ho deciso di approfondire l'argomento della programmazione a basso livello trattato a lezione, andando a realizzare un piccolo videogioco gestito tramite microcontrollore programmato in linguaggio assemblativo. Il videogioco, come mostrato in figura, è un clone del classico minigioco di corse arcade spesso incluso, assieme ad altri titoli come *Tetris*, *Snake*, *Breakout* o *Tanks*, nelle console portatili economiche intorno agli anni '90. Lo scopo del gioco è quello di evitare le "automobili" che scorrono verticalmente lungo il display, muovendo l'auto del giocatore orizzontalmente tramite gli appositi pulsanti di direzione. A differenza di questo, nella mia implementazione i controlli avvengono tramite sensore di prossimità, probabilmente molto più scomodo e impreciso rispetto ai classici bottoni, ma sicuramente più interessante da gestire dal punto di vista software. Il fine ultimo del progetto non è, infatti, quello di realizzare una console che possa intrattenere, ma piuttosto di apprendere le basi della programmazione a basso livello.



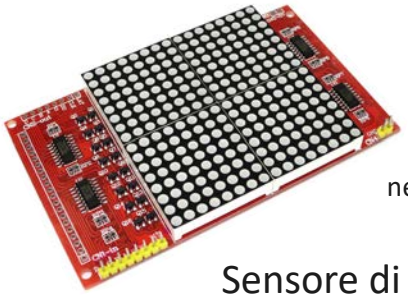
Un video che riassume le caratteristiche del progetto è visualizzabile al seguente indirizzo web:
https://www.youtube.com/watch?v=RqFYr64puMI

Scelta dei componenti

Il progetto prevede che

- le schermate di gioco possano essere visualizzate tramite un display monocromatico da (almeno) 16x16 punti che permetta una frequenza di aggiornamento dell'immagine adeguata ad oggetti in movimento
- il gioco sia controllabile tramite un sensore di prossimità che effettui almeno una decina di misurazioni al secondo per una distanza massima di almeno 20-30cm
- il gioco possa emettere suoni in determinate situazioni
- le schermate di gioco e i vari oggetti a schermo possano essere personalizzate mediante PC o smartphone connessi tramite porta USB
- sia possibile memorizzare permanentemente il punteggio massimo totalizzato

A questo proposito, ho scelto i componenti elencati di seguito.



Display

Una matrice di LED monocromatici 16x16, costituita da 4 unità 8x8 saldate su una scheda elettronica assieme ai relativi decoder, controllabile tramite 8 pin di dati senza la necessità di effettuare alcuna saldatura.

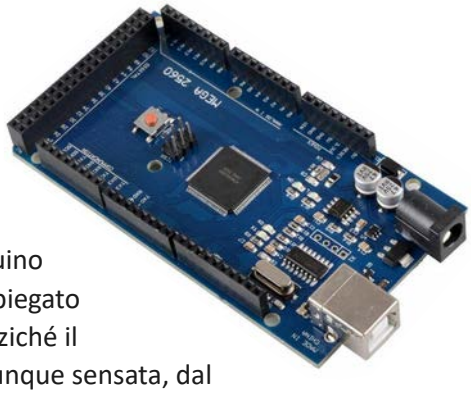
Sensore di prossimità

Un sensore di prossimità *HC-SR04*, capace di misurare distanze da 2cm a 4m impiegando meno di 60ms per misurazione, controllabile tramite 2 pin di dati.

Cicalino

Un buzzer passivo *YL-44*, capace di generare onde sonore quadre di frequenza variabile in funzione del segnale applicato sull'unico pin di controllo.

Microcontrollore



Un MCU *AVR* a 8 bit *Atmel ATmega2560*, saldato in una development board assieme ad un oscillatore di cristallo da 16MHz che funge da segnale di clock, un chip convertitore USB-UART *CH340G* e diversi connettori per i vari pin I/O. La development board nasce come clone economico dell'*Arduino Mega 2560* (presenta, infatti, gli stessi componenti, ad eccezione del chip CH340G), tuttavia in questo progetto **non** verrà utilizzato alcuno strumento o libreria del framework Arduino o della community e il microcontrollore verrà riportato allo stato di fabbrica come spiegato nei paragrafi successivi. La scelta di acquistare una development board completa anziché il singolo MCU Atmel per programmare esclusivamente in assembly AVR rimane comunque sensata, dal momento che, facendo diversamente, avrei dovuto acquistare i componenti elencati sopra separatamente, connetterli in una soluzione sicuramente meno compatta e, soprattutto, rischiare di danneggiarli, vista la mia competenza praticamente nulla in ambito elettronico.

Programmatore

Un programmatore USB-ISP *USBAsp* compatibile con MCU ATmega2560 e software *avrdude*.

Preparazione

Per prima cosa, ho dovuto installare il set di strumenti software per la programmazione AVR , che include l'utility avrdude per la programmazione ISP, le librerie e i compilatori GNU C/C++, l'assemblatore GNU *avr-as* e, su Windows, l'assemblatore Atmel *AVRAsm2*. Successivamente ho connesso il programmatore USBAsp all'interfaccia ISP6 della development board e, tramite avrdude, sono andato a modificare i *fuse bits* del microcontrollore in modo da disabilitare e sovrascrivere il bootloader Arduino preinstallato, riportando l'MCU allo stato di fabbrica. A questo punto ho cominciato a muovere i primi passi nel mondo dell'assembly AVR a partire da semplici programmi che facessero lampeggiare dei LED, fino alla comunicazione con le varie componenti del progetto, passando per la gestione dei timer, dell'I/O, della memoria e degli interrupt. Fin da subito ho sentito la mancanza dei classici meccanismi di debugging (possibili solo tramite On-Chip Debugger dedicato), compresa la possibilità di stampare a video dei messaggi in determinati punti del programma (come avviene spesso nella programmazione desktop) e dei costrutti sintattici che rendono i linguaggi ad alto livello più immediati da scrivere e mantenere. A questo proposito ho deciso di concentrarmi prima di tutto sugli accorgimenti che avrebbero reso il codice più gestibile:

Come documentazione ho utilizzato il datasheet dell'ATmega2560 e l'AVR Instruction Set Manual, scaricabili gratuitamente dal sito Atmel.

- L'implementazione, dove possibile, delle varie funzionalità in *moduli* separati, inizializzati all'avvio e facilmente integrabili o escludibili dalla build in fase di test
- L'utilizzo massivo del preprocessore per motivi di performance, pulizia e chiarezza del codice, parametrizzazione e per la distribuzione delle risorse limitate (ad esempio i timer, come descritto sotto) tra i vari moduli
- Lo sviluppo di un modulo che consentisse la comunicazione con il PC o lo smartphone tramite USB a scopo di debugging (implementata all'estremo opposto tramite un semplice script ricevitore *python*)
- Una serie di regole rigide per i nomi delle etichette che permettessero di differenziare e riconoscere velocemente i riferimenti ai registri, alle subroutine, agli interrupt e ai dati presenti in diverse memorie o segmenti di memoria

Fatto questo, ho cominciato a sviluppare i vari moduli uno per uno, come descritto nella pagina successiva.

```
; @0 (prefix)
; @1 (timer index)
.macro TIM_DEF
	.equ TIM_@1_TAKEN = 1
	.if ((@1 < 0) || (@1 > 5))
		.error "Bad timer index"
	.else
		.equ @0_TCCRA = TCCR@1A
		.equ @0_TCCRB = TCCR@1B
		.equ @0_TIMSK = TIMSK@1
		.equ @0_TIFR = TIFR@1
		.equ @0_OCAaddr = OC@1Aaddr
		.equ @0_OCBaddr = OC@1Baddr
		.equ @0_OVFaddr = OVF@1addr
		.if (@1 == 0) || (@1 == 2)
			.equ @0_TCNT = TCNT@1
			.equ @0_OCRA = OCR@1A
			.equ @0_OCRB = OCR@1B
		.else
			.equ @0_TCCRC = TCCR@1C
			.equ @0_OCCaddr = OC@1Caddr
			.equ @0_ICPaddr = ICP@1addr
			_U_16R_DEF @0_ICR, ICR@1
			_U_16R_DEF @0_TCNT, TCNT@1
			_U_16R_DEF @0_OCRA, OCR@1A
			_U_16R_DEF @0_OCRB, OCR@1B
			_U_16R_DEF @0_OCRC, OCR@1C
		.endif
	.endif
.endmacro
```

Nella porzione di codice riportata a fianco, è implementata una macro che definisce i registri di controllo del timer passato come parametro tramite una serie di costanti. L'utilizzo di questa macro consente di rendere il codice dei moduli indipendente dal timer che andranno effettivamente ad utilizzare e di cambiare timer senza dover modificare alcuna riga di codice. Restituisce, inoltre, un errore nel caso in cui un timer sia già utilizzato da un altro modulo.

Nella porzione di codice riportata sotto, è implementata una macro che consente di gestire un interrupt all'interno di un qualsiasi modulo.

```
; @0 (interrupt vector address)
.macro ISR
	.set ISR_PC = PC
	.org @0
		jmp ISR_PC
	.org ISR_PC
.endmacro
```

Moduli

LED integrato

Definisce macro che consentono di accendere, spegnere o invertire lo stato del LED integrato nella scheda tramite l'utilizzo di un solo registro d'appoggio. Utilizzato per vari test in fase di programmazione e per indicare la presenza di un oggetto rilevato dal sensore di prossimità nel progetto finito.

Gestore dei dati non volatili

Definisce macro che consentono ai moduli di salvare e recuperare dati sulla memoria *EEPROM* non volatile.
Gestisce, inoltre, un piccolo protocollo di comunicazione con una apposita applicazione desktop e mobile che consente di modificare questi dati tramite connessione USB. Le schermate di gioco, le sprite del giocatore e degli ostacoli, lo schema di generazione degli ostacoli, i suoni, i vari parametri dei moduli di gioco, di rendering e di controllo sono salvati in questo modo, e sono quindi completamente personalizzabili.

Misurazione della distanza

Interagisce periodicamente con il sensore di prossimità e, tramite un timer a 16 bit provvisto di *input capture unit*, calcola e salva in memoria *SRAM* la distanza dell'oggetto più vicino nel raggio di misurazione del sensore. Effettua tutte le operazioni in modo asincrono al ciclo di rendering, mediante interrupt, in modo da non occupare il microcontrollore durante i periodi di attesa.

Buzzer

Definisce macro che consentono di caricare dalla memoria e riprodurre sequenze di suoni di durata e frequenza variabile, mediante l'utilizzo di un timer a 8 bit che genera l'onda quadra e uno a 16 bit che scorre la sequenza.

Ciclo di rendering

Definisce il ciclo principale del programma, all'interno del quale, una colonna per volta, viene composta e inviata alla matrice di LED l'immagine generata dalla schermata corrente. Disabilita nelle porzioni di codice critiche gli interrupt e li riattiva subito dopo. Aspetta che i LED siano effettivamente accesi prima di continuare,e, durante l'attesa, consente agli altri moduli di utilizzare la CPU. Gestisce, infine, l'alternarsi delle varie schermate.

Schermata di gioco

Gestisce a intervalli regolari la creazione di nuovi ostacoli e lo scorrimento di quelli preesistenti. Recupera le misurazioni della distanza, fa il possibile per attenuarne il rumore, e muove il cursore del giocatore di conseguenza. Richiede il passaggio alla schermata di pausa in caso di prolungata assenza del giocatore o a quella di game over in caso di scontro del cursore con un ostacolo.

Schermata di pausa

Visualizza un messaggio di pausa e passa alla schermata di gioco quando la presenza del giocatore è rilevata.

Schermata di game over

Visualizza un messaggio di game over per qualche istante, poi stampa a video il punteggio totalizzato nell'ultima partita e quello massimo raggiunto, dopo averli convertiti in cifre decimali.

Come esempio, è riportata una porzione di codice che si occupa di generare una colonna della schermata di gioco.

```
; assegna la colonna 'col' della schermata di gioco al registro 'c'
.macro G_SRC_DRAW
	; carica dalla memoria la colonna della griglia degli ostacoli
	; assegna al registro puntatore 'X' l'indirizzo iniziale della griglia
	ldi XH, HIGH(_g_ram_frame + 1)
	ldi XL, LOW(_g_ram_frame + 1)
	; sposta il puntatore 'X' a seconda della colonna degli ostacoli da caricare
	ldi _g_tmp1, 3
	mul _g_tmp1, _g_col
	clr _g_tmp1
	add XL, mull
	adc XH, _g_tmp1
	; carica la colonna degli ostacoli nel registro 'c'
	ld _g_ch, X+
	ld _g_cl, X
	; carica dalla memoria la colonna del cursore del giocatore
	; calcola la colonna del cursore da caricare
	lds _g_tmp1, _g_ram_col
	sub _g_tmp1, _g_col
	brpl _g_l_draw_abs_done
	neg _g_tmp1
_g_l_draw_abs_done:
	; carica la colonna del cursore nel registro temporaneo 'tmp1'
	ldi XH, HIGH( _g_ram_bm_player_acs )
	ldi XL, LOW( _g_ram_bm_player_acs )
	add XL, _g_tmp1
	clr _g_tmp1
	adc XH, _g_tmp1
	ld _g_tmp1, X
	mov _g_tmp2, _g_cl
	; aggiunge la colonna del cursore al registro 'c'
	or _g_cl, _g_tmp1
	; controlla se il cursore ha punti in comune con gli ostacoli
	and _g_tmp2, _g_tmp1
	; termina se il cursore non ha punti in comune con gli ostacoli
	breq _g_l_draw_done
	; game over
	; riproduce il suono di game over
	BZ_SRC_START _g_ram_snd_over
	; richiede il passaggio alla schermata di game over
	ldi _g_tmp1, ML_SCREEN_SCORE
	sts ml_ram_screen, _g_tmp1
	clr _g_tmp1
	; ferma il timer di scorrimento degli ostacoli
	sts _G_TCCRB, _g_tmp1
_g_l_draw_done:
.endmacro
```