

Sistemi Embedded



Progettazione gioco del Tris su DE10 Lite con display touch LT24
in linguaggio Verilog e C

Studente:

Antonio Di Vito

Professore:

Federico Baronti

Obiettivi di progetto

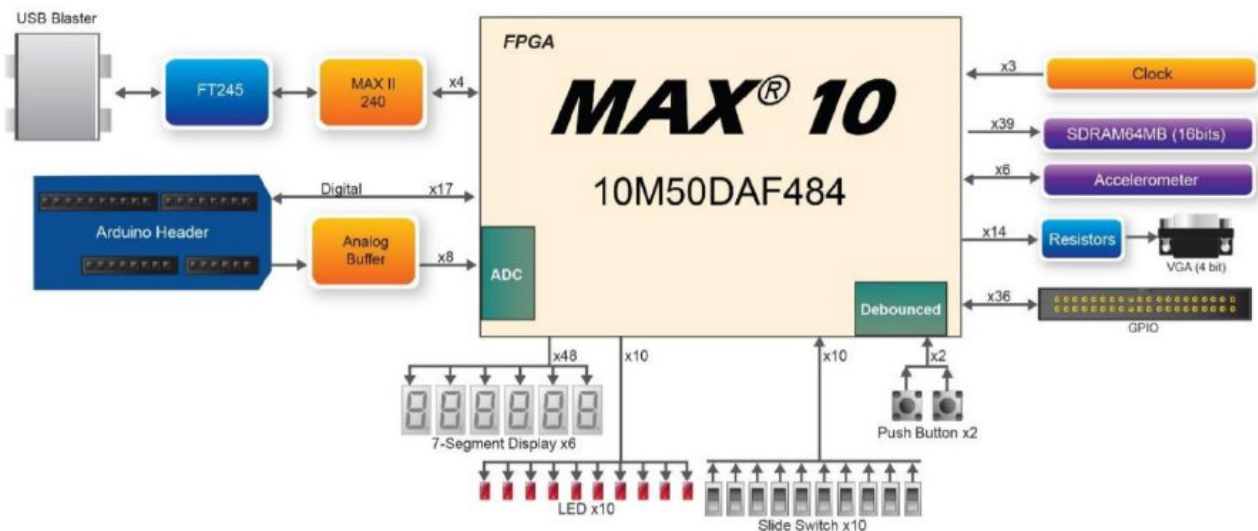
Il progetto consiste nello sviluppare un sistema hardware e software interfacciato con la periferica esterna di visualizzazione e interazione per poter giocare al gioco del Tris. A tale scopo vengono usati per la realizzazione della parte hardware i tool di Intel Quartus e Platform Designer (ex Qsys). Questi software operano di base attraverso un linguaggio di descrizione hardware, in particolare il Verilog.

Successivamente è stata usata l'IDE Eclipse per la programmazione software della logica di gioco nel linguaggio C.

Si analizzerà prima la parte hardware, descrivendo i blocchi IP di Qsys usati per lo scopo, e poi la parte software riguardante il gioco vero e proprio.

La parte HW può essere riutilizzata per altri progetti che fanno uso del display touch LT24, indipendentemente dalla parte software che può essere riscritta per altri scopi.

Breve descrizione della scheda programmabile Altera DE10 Lite



La DE10 Lite presenta una piattaforma hardware dotata di molte periferiche costruite attorno l'FPGA MAX 10. Questo dispositivo è dotato di:

- FPGA MAX 10 10M50DAF484C7G;
- Due ADC integrati;
- 50K elementi programmabili;

- 1638 Kbits di memoria M9k;
- 5888 Kbits di memoria flash;
- 4 PLL.

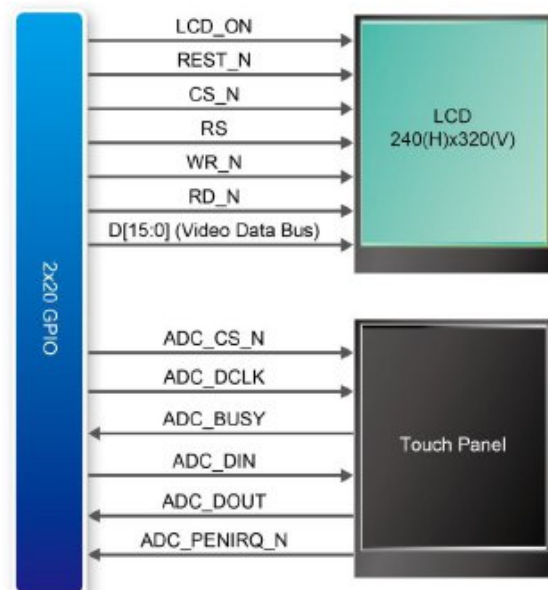
Altro hardware presente su scheda:

- 64 MB di memoria SDRAM;
- USB blaster;
- GPIO headers;
- Arduino headers;
- Porta VGA;
- Accelerometro;
- 10 led;
- 10 switch;
- 2 pushbuttons;
- 6 display a 7 segmenti.

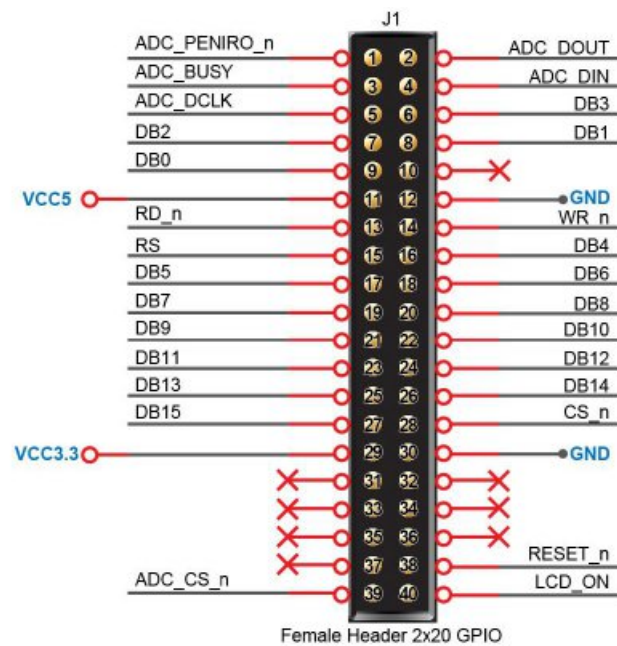
Prima di iniziare con la descrizione hardware e software, si descrive sommariamente il funzionamento del display touch LT24.

LT24 e varie funzionalità

Questo dispositivo è fondamentalmente basato su due parti hardware distinte: il pannello touch resistivo a tocco singolo e il sottostante pannello LCD da 240*320 pixel a 65 mila colori. Presenta una interfaccia GPIO 2*20 per il collegamento con la scheda programmabile. Globalmente, la scheda è fatta così e presenta questi collegamenti:



Il collegamento con l'interfaccia GPIO è il seguente:



In fase di progetto, attraverso il datasheet viene quindi effettuato l'assegnamento dei PIN usati attraverso il software Quartus. Lo standard IO per i pin è di 3.3-V LVTTTL. A parte i segnali di input ADC_PENIRQ_N, ADC_DOUT e ADC_BUSY, gli altri sono tutti degli output.

Ora si descriverà brevemente il dispositivo da un punto di vista fisico, più tardi si parlerà di come lo si è interfacciato con Qsys in HW e come viene pilotato via software Eclipse in C. Saranno descritte le parte del display e la parte del touch.

Parte del display LCD

Come già accennato, il display presenta una risoluzione di 240*320 pixel e ha 65 mila colori. Per usare il dispositivo viene usato il SoC driver ILI9341 della ILITEK. In particolare, viene usata la configurazione RGB 5-6-5 65k colori basata sull'interfaccia 8080 a 16 bit.



L'FPGA si occupa di pilotare i collegamenti con cui si interfaccia il dispositivo:

- CSX è un pin di attivazione attivo basso, sostanzialmente serve ad accendere il display;
- D/CX è il pin di selezione fase Data o Command: quando è 1, è selezionata la Data mode, quando 0 è selezionata la Command mode;
- WRX è il segnale di scrittura e scrive dati al fronte in salita del clock;
- RDX è il segnale di lettura e legge dati al fronte di salita del clock;
- Da D0 a D7 sono i pin che si occupano di tradurre la Command mode quando selezionata;
- Da D0 a D15 sono i pin che si occupano di tradurre i dati RGB quando la Data mode è selezionata.

Per dare una idea veloce del funzionamento, viene mostrato il pixel data transfer format:

Count	0	1	2	3	...	238	239	240
D/CX	0	1	1	1	...	1	1	1
D15		0R4	1R4	2R4	...	237R4	238R4	239R4
D14		0R3	1R3	2R3	...	237R3	238R3	239R3
D13		0R2	1R2	2R2	...	237R2	238R2	239R2
D12		0R1	1R1	2R1	...	237R1	238R1	239R1
D11		0R0	1R0	2R0	...	237R0	238R0	239R0
D10		0G5	1G5	2G5	...	237G5	238G5	239G5
D9		0G4	1G4	2G4	...	237G4	238G4	239G4
D8		0G3	1G3	2G3	...	237G3	238G3	239G3
D7	C7	0G2	1G2	2G2	...	237G2	238G2	239G2
D6	C6	0G1	1G1	2G1	...	237G1	238G1	239G1
D5	C5	0G0	1G0	2G0	...	237G0	238G0	239G0
D4	C4	0B4	1B4	2B4	...	237B4	238B4	239B4
D3	C3	0B3	1B3	2B3	...	237B3	238B3	239B3
D2	C2	0B2	1B2	2B2	...	237B2	238B2	239B2
D1	C1	0B1	1B1	2B1	...	237B1	238B1	239B1
D0	C0	0B0	1B0	2B0	...	237B0	238B0	239B0

Sostanzialmente, quando viene selezionato il D/CX a 0, si potrà decodificare l'ingresso dei pin da D0 a D7 per la fase di Command, mentre selezionando D/CX a 1, si potrà decodificare i pin da D0 a D15 ottenendo la gamma di colori in fase di Data.

Nella parte software, verrà descritto nel dettaglio il pilotaggio.

Parte del touchscreen

Il dispositivo per il tocco è essenzialmente guidato dal funzionamento di un convertitore analogico digitale: in particolare viene utilizzato il driver AD7843 della Analog Devices. E' un ADC a 12 bit usato per convertire il tocco nelle corrispondenti coordinate su un piano cartesiano X-Y.

Per ottenere le coordinate dall'ADC, l'utente deve monitorare il segnale di interrupt della periferica proveniente dal pin ADC_PENIRQ_N. Tale uscita resta alta quando non viene toccato il dispositivo, grazie al collegamento verso l'alimentazione positiva attraverso un resistore di pull-up. Nel momento del tocco, il segnale di interrupt va basso verso massa, e di conseguenza una control word viene mandata all'ADC con la comunicazione seriale SPI.

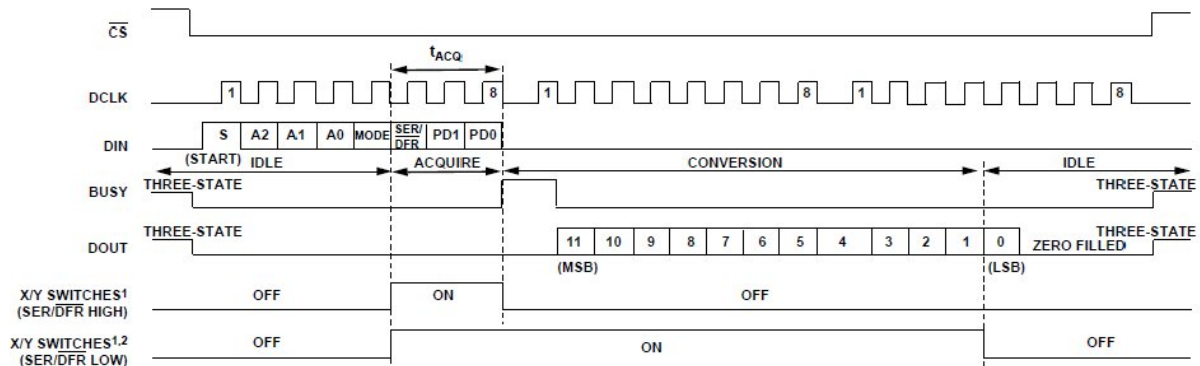
Il registro di controllo è a 8 bit, nello specifico:

MSB							LSB
S	A2	A1	A0	MODE	SER/DEF	PD1	PD0

I caratteri in figura sono degli mnemonici che raffigurano i bit da 0 a 8. Descrivendoli:

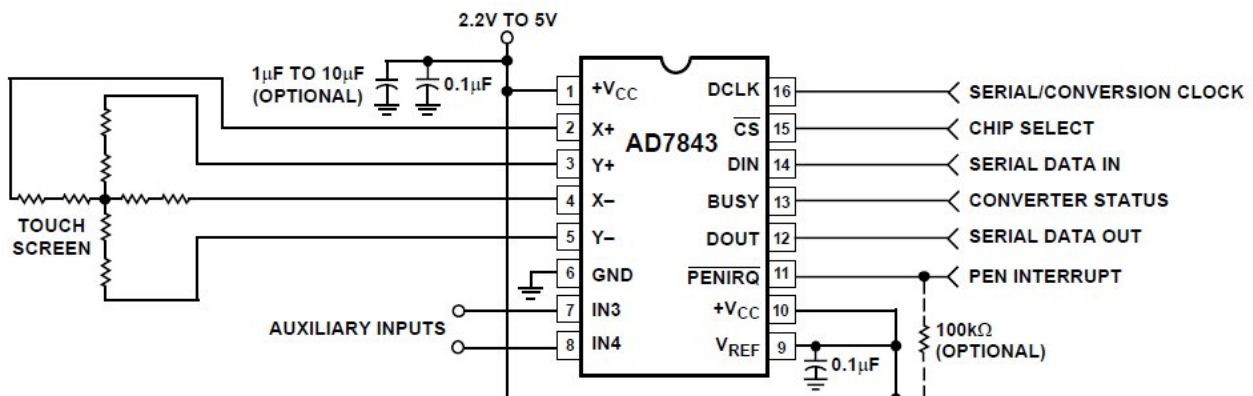
- S (bit 7) è il bit di start della control word corrispondente a DIN. Si può far partire una nuova control word ogni 15 cicli di clock quando si seleziona la conversione a 12 bit (quella usata in questo progetto);
- A2-A0 (bit da 6 a 4) sono i bit di selezione del canale, e assieme al SER/~DEF controllano le impostazioni del multiplexer, degli switch e dei riferimenti;
- MODE (bit 3) è il selettore della risoluzione del convertitore, nello specifico viene posto a 0 per la profondità a 12 bit;
- SER/~DFR (bit 2) è il bit di selezione per single-ended/differential reference. Verrà posto nel nostro progetto sulla differential reference (posto a 0) e come già detto seleziona la modalità di funzionamento del dispositivo. Il funzionamento sarà brevemente spiegato fra poco;
- PD1 e PD0 (bit 1 e bit 0) riguardano i bit del power management del AD7843.

Il diagramma di timing della SPI è questo:



Descrivendolo brevemente: anzitutto, per avviare un processo di conversione è necessario che il segnale di chip select $\sim CS$ sia basso (quindi dispositivo attivato), quindi si riceve in ingresso la sequenza della control word. L'acquisizione della coordinata avviene nel momento in cui viene passato il bit di selezione del riferimento SER/ $\sim DFR$ di tensione, quindi successivamente viene codificato (ovvero digitalizza il tocco) in uscita su 12 bit, rispettando le tempistiche necessarie del dispositivo.

Per avere una migliore visione di insieme, viene rappresentato il circuito analogico equivalente: in questo schema viene illustrato intuitivamente il funzionamento del tocco, che individua la coordinata attraverso i resistori e le capacità, e la parte di controllo. Nella parte di descrizione del software, verrà discusso meglio il pilotaggio.



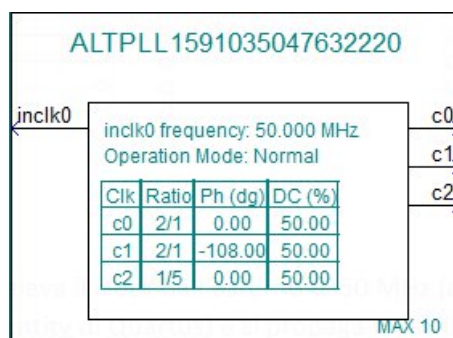
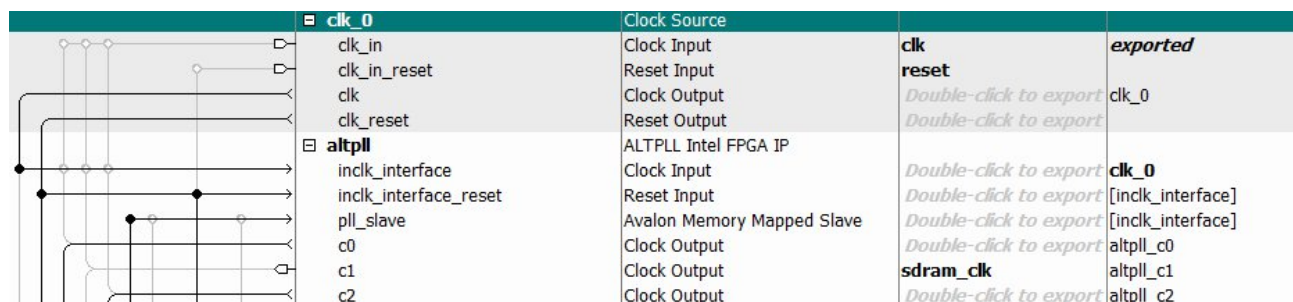
Parte Hardware - SOPc System Builder (Qsys)

Ora passiamo alla descrizione dell'hardware istanziato nel tool Qsys (che genera il file .sopcinfo del computer progettato, utilizzabile in Eclipse Nios) e poi compilato da Quartus (che genera il file .sof per l'FPGA). Il .sopcinfo file permette la compilazione del progetto software e altro non è che un contenitore di informazioni che costituiscono un computer vero e proprio, che include il processore e le sue periferiche per il funzionamento. Il .sof riguarda l'installazione dell'FPGA, il pin assignment, il modello di scheda utilizzato e la timing analysis col file .sdc.

Anzitutto, da Quartus si sceglie il modello di scheda per poter usare i componenti compatibili presenti nel tool Qsys. Dopo si passa all'uso del Platform Designer (Qsys) per collegare i componenti che servono.

Il sistema presenta delle componenti di base essenziali per farlo funzionare, quindi si aggiungono i componenti utili a collegare il display touch con il sistema.

Clock e PLL

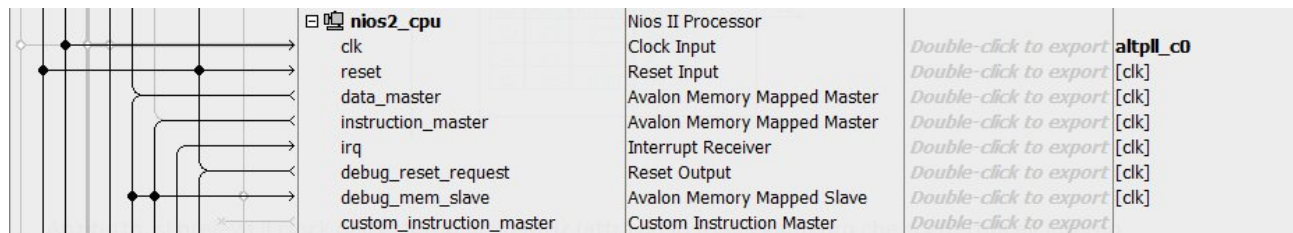


Anzitutto, si preleva il clock dall'esterno (fornito dall'FPGA) di 50 MHz, attraverso il collegamento che si effettua all'interno della top level entity di Quartus, quindi si propaga verso il dispositivo PLL. In particolare è il modulo ALTPLL della libreria. Viene lasciata l'impostazione di default per questo dispositivo, e vengono creati 3 clock derivati: avranno tutti un duty cycle 50-50, ma si differenziano per clock e sfasamenti differenti. C0 sarà da 100MHz; C1 sarà esportato verso il controller della offchip SDRAM, quindi dovrà

essere sfasato di 3ns in anticipo come empiricamente testato per rispettare il timing; C2 è da 10 MHz. Ognuno di loro sarà collegato a diverse periferiche, successivamente ne verrà spiegato il collegamento.

Per quanto riguarda il reset, si esporta il segnale verso l'esterno per venire collegato nello stesso modo del clock. Verrà quindi propagato lungo tutta la catena del computer creato.

Processore

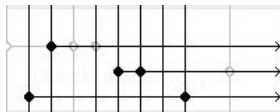


Reset Vector	
Reset vector memory:	onchip_mem.s1
Reset vector offset:	0x00000000
Reset vector:	0x09004000

Exception Vector	
Exception vector memory:	sdram.s1
Exception vector offset:	0x00000020
Exception vector:	0x04000020

E' stato scelto il processore embedded Nios II, collegato al clock C0 del PLL (100MHz). Per quanto riguarda la sua configurazione, non è stato sufficiente istanziare la versione resource optimized economy messa a disposizione, ma è stato necessario scegliere la versione fast. Con la economy la visualizzazione a schermo è risultata rallentata e non propriamente fruibile. E' stato scelto con configurazione di default, ma si è modificata la parte riguardante i vettori di reset e delle eccezioni per portare il pc in una configurazione nota: per il vettore di reset si punta alla memoria on-chip, grazie alla sua non volatilità, mentre quello delle eccezioni punta alla memoria SDRAM a bassa latenza. Queste scelte sono in accordo con i tutorial messi a disposizione da Intel (Nios II Processor Reference Guide n2cpu_nii51004.pdf).

On-chip Memory

	onchip_mem	On-Chip Memory (RAM or ROM)...	
	clk1	Clock Input	<i>Double-click to export</i> altpll_c0
	s1	Avalon Memory Mapped Slave	<i>Double-click to export</i> [clk1]
	reset1	Reset Input	<i>Double-click to export</i> [clk1]

Memory type

Type: RAM (Writable) ▾

☐ Dual-port access

☐ Single clock operation

Read During Write Mode: DONT_CARE ▾

Block type: AUTO ▾

Size

☐ Enable different width for Dual-port access

Slave S1 Data width: 32 ▾

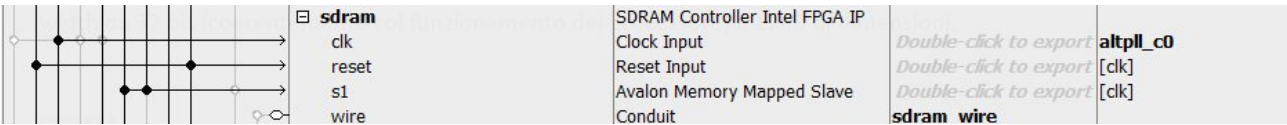
Total memory size: 12228 bytes

☐ Minimize memory block usage (may impact fmax)

La memoria on-chip non ha bisogno di un controller per interfacciarsi col processore ed è formata grazie all'FPGA. Lo slave è pilotato da data ed instruction master del processore, mentre il clock è collegato al C0 da 100 MHz. Le configurazioni sono state: RAM per impostarla come memoria di lettura e scrittura, data width da 32 bit (coerentemente col funzionamento del processore) e 12KB di dimensioni.

E' usata fondamentalmente come memoria per il reset vector del processore in quanto memoria non volatile e veloce (come indicato nel Nios II Processor Reference Guide n2cpu_nii51004.pdf).

SDRAM



Memory Profile Timing

Data Width

Bits: 16

Architecture

Chip select: 1

Banks: 4

Address Width

Row: 13

Column: 10

Generic Memory model (simulation only)

☐ Include a functional memory model in the system testbench

Memory Size = 64 MBytes
33554432 x 16
512 MBits

Memory Profile Timing

CAS latency cycles:: 1
2
3

Initialization refresh cycles: 2

Issue one refresh command every: 7.8125 us

Delay after powerup, before initialization: 100.0 us

Duration of refresh command (t_rfc): 70.0 ns

Duration of precharge command (t_rp): 15.0 ns

ACTIVE to READ or WRITE delay (t_rcd): 15.0 ns

Access time (t_ac): 5.5 ns

Write recovery time (t_wr, no auto precharge): 14.0 ns

La SDRAM (off-chip) viene controllata da un controller apposito, che ha le seguenti configurazioni: come prima, la memoria vera e propria è collegata al clock C0 da 100Mhz; il controller è collegato a C1 (esportato), sottoposto a sfasamento di -3ns così come specificato nei datasheet di Intel per rispettare i timing. Nel conduit di export ci sono i fili da collegare in Quartus per il funzionamento del dispositivo.

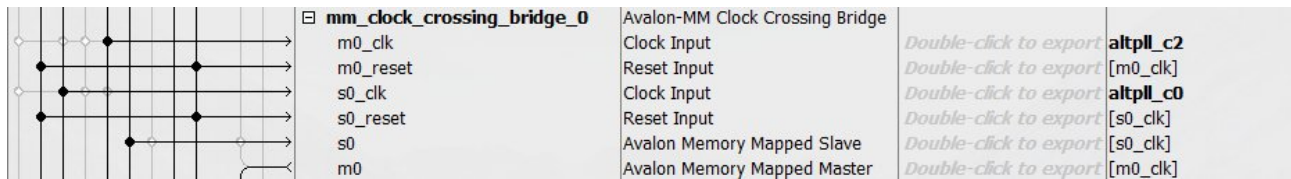
Viene quindi fatta lavorare su 16 bit (halfwords) e configurata in maniera tale da essere conforme con la quantità di memoria a disposizione sulla DE10 Lite, ovvero 64MB. Inoltre, anche le caratteristiche di timing vengono modificate da quelle di default per rispettare le specifiche della SDRAM presente su questa precisa scheda. E' quindi usata anche come exception vector dal processore.

JTAG UART

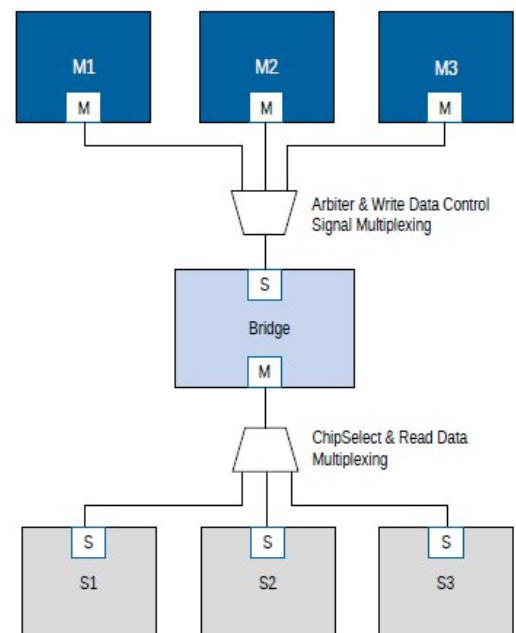


Questo blocco, istanziato con configurazioni di default, è essenziale per interfacciare la scheda con un PC host attraverso la porta USB Blaster: essenzialmente serve per il download del firmware sulla scheda e per operazioni di debug attraverso la console dei software utilizzati.

Clock Crossing Bridge



Data	
Data width:	32
Symbol width:	8
Address	
Address width:	10
<input type="checkbox"/> Use automatically-determined address width	
Automatically-determined address width:	8
Address units:	SYMBOLS
Burst	
Maximum burst size (words):	1
FIFOs	
Command FIFO depth:	8
Response FIFO depth:	8
Master clock domain synchronizer depth:	2
Slave clock domain synchronizer depth:	2



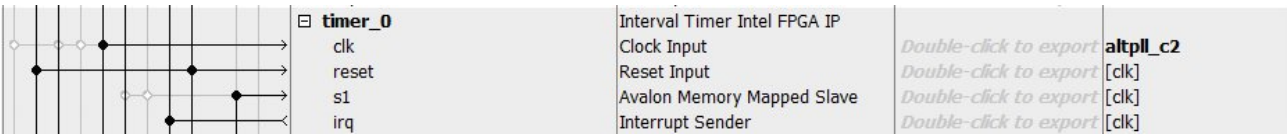
Questo modulo di libreria è stato usato per permettere di far dialogare diversi componenti con domini di clock diversi. Un bridge ha una interfaccia master e una interfaccia slave. Nel nostro caso, l'interfaccia slave è collegata direttamente alla data master del processore attraverso il ponte, mentre la master di questo dispositivo pilota le Avalon MM delle periferiche interessate. Questo bridge usa una logica FIFO asincrona (memoria tampone) per implementare la logica del clock crossing. E' una sorta di smistatore per i segnali di clock.

Il clock bridge permette di collegare una sorgente di clock a svariate interfacce di clock in ingresso.

Il dispositivo avrà quindi collegata la frequenza di clock C0 da 100 MHz del PLL sullo slave, e il suo master che pilota le periferiche a valle sarà collegato a C2 da 10MHz.

I parametri sono stati lasciati di default.

Interval Timer



Timeout period

Period: 1

Units: ms

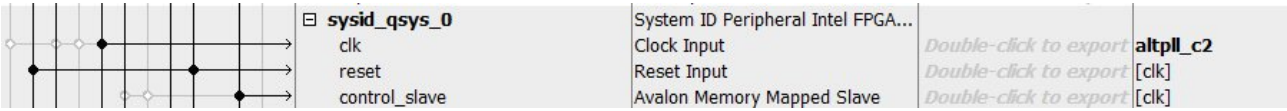
Timer counter size

Counter Size: 32

Questo dispositivo è un contatore che genera un segnale di output quando raggiunge un valore programmato. Serve ad evitare i blocchi di sistema. Il segnale di output generato è un interrupt inviato al processore. Il dispositivo è collegato al clock C2 da 10MHz, con scadenza di 1 ms e una profondità di 32 bit.

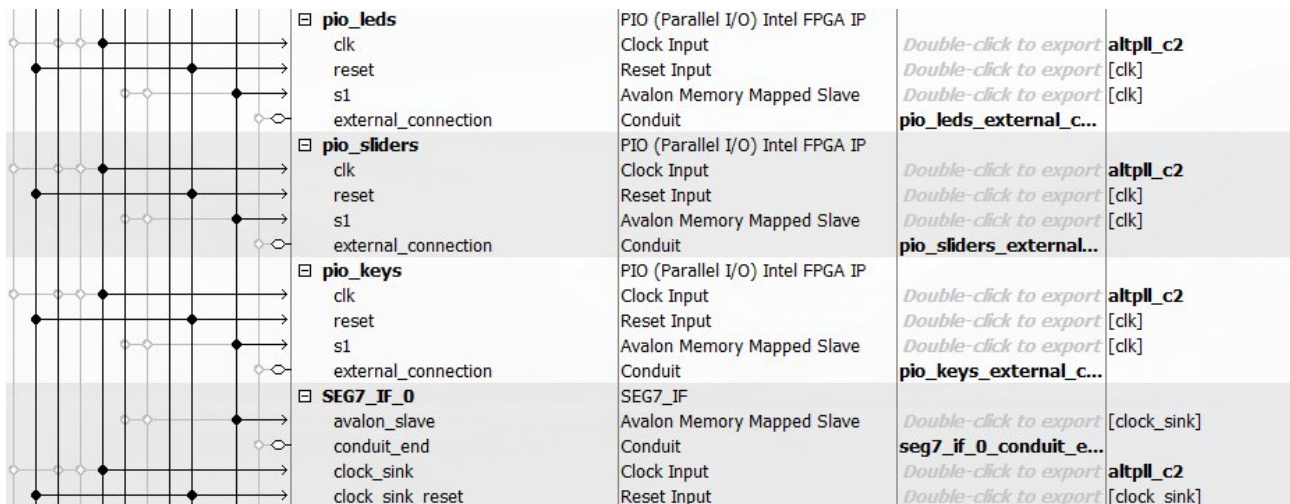
Nel progetto è stato utilizzato come timer di sistema e per la parte riguardante il driver del touch, di cui se ne parlerà nella parte software della relazione.

System ID Peripheral



Questo dispositivo, lasciato con configurazione di default, identifica univocamente, mediante un numero esadecimale a 32 bit, il processore NIOS II utilizzato.

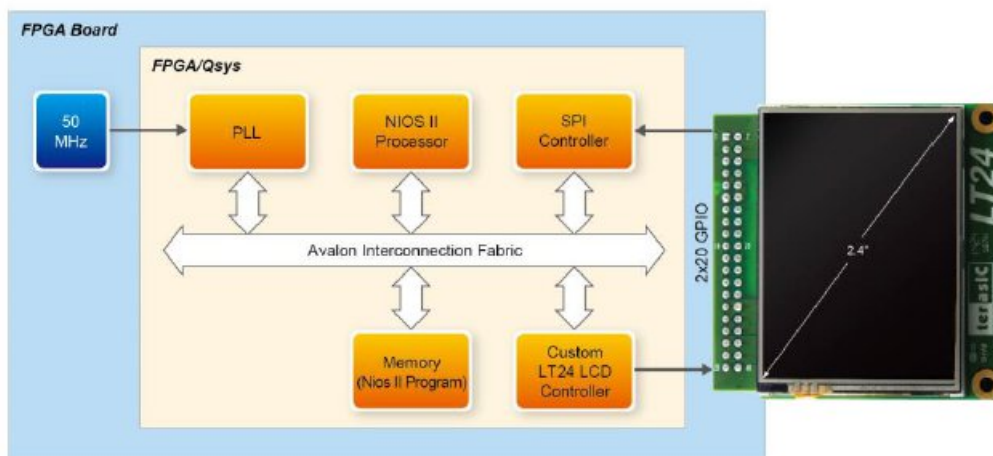
LEDs, sliders, tasti e display 7 segmenti



Queste periferiche sono pilotate attraverso l'interfaccia PIO, sono collegati tutti allo stesso clock C2 da 10 MHz e le connessioni esterne vengono esportate per essere collegate direttamente ai pin della scheda (che saranno collegati fisicamente col file Verilog del top level entity in Quartus). Per quanto riguarda la configurazione, i LED sono impostati come output e ne sono 10, gli slider sono input e ne sono 10, mentre i tasti KEY sono due e configurati come input. Per quanto riguarda il pilotaggio del display a 7 segmenti, per semplicità si è usato un blocco IP della Terasic denominato SEG7_IF. Si troverà il file .v allegato alla relazione.

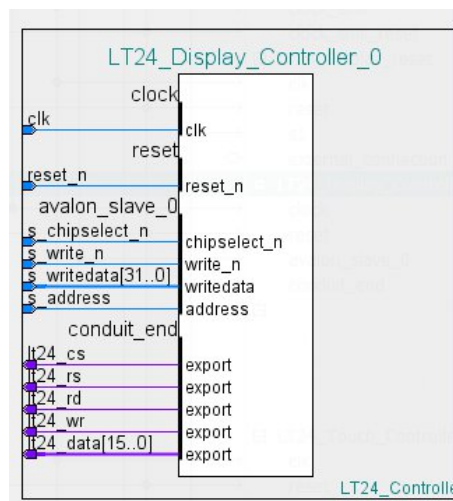
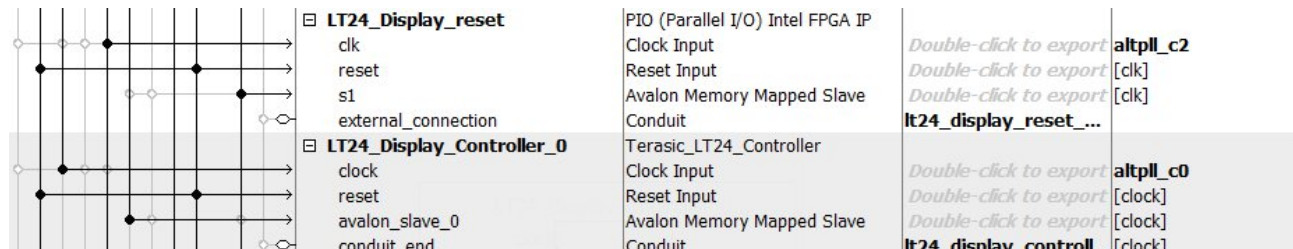
LT24 Display Touch

Si descriverà dapprima la parte riguardante il controller del display, e dopo la parte riguardante il touch. Maggiori dettagli si trovano nel capitolo che recensisce l'LT24. Il disegno sottostante (a scopo illustrativo) fa vedere sommariamente come viene interfacciato il display alla board:



Come accennato già precedentemente, il dispositivo fa uso del driver della ILITEK per il display e del driver della Analog Devices per il touch in SPI attraverso l'ADC. Vengono quindi istanziati in Qsys due moduli separati.

LT24 Display Module di Terasic

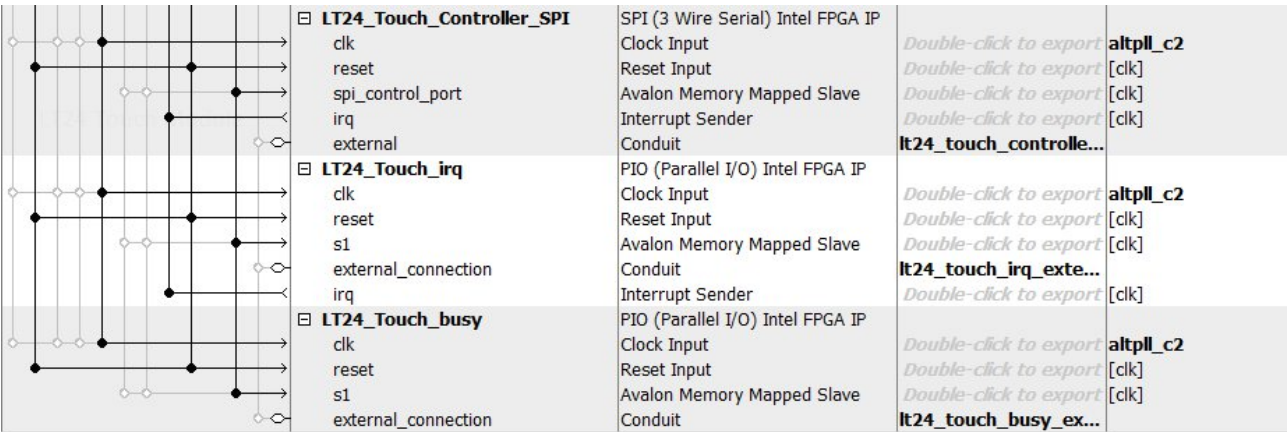


Il blocco istanziato è un blocco fornito da Terasic, di cui il file .v è allegato alla relazione. E' interfacciato attraverso memory mapped. Possiamo notare come il conduit raggruppi più fili da esportare all'esterno, e sono nient'altro che i fili che riguardano le connessioni viste nel capitolo riguardante il display.

L'IP core per LT24 display genera le interfacce dei segnali richiesti per pilotare il driver dell'IC. Il controller prende i dati in ingresso e li scrive sulla RAM grafica del dispositivo.

Alla lista di export che si vedono in figura, mancano i fili riguardanti l'alimentazione e il reset. Per quanto riguarda il filo dell'alimentazione (non presente nelle figure sopra), si è deciso di dare alimentazione costante al pin predisposto attraverso un assign continuo all'interno del file .v top level, bypassando quindi la configurazione di Qsys. Per il reset, invece, si è predisposto un blocco PIO da 1 bit in uscita che verrà esportato per essere manualmente pilotato con un assign collegato al pulsante KEY[1] sempre dalla top level entity. Il file top level entity è allegato a questa relazione.

LT24 Touch Module



Master/Slave

Type: Master

Number of select (SS_n) signals (one for each slave): 1

SPI clock (SCLK) rate: 32000 Hz

Actual clock rate: 31847.0 Hz

☐ Specify delay

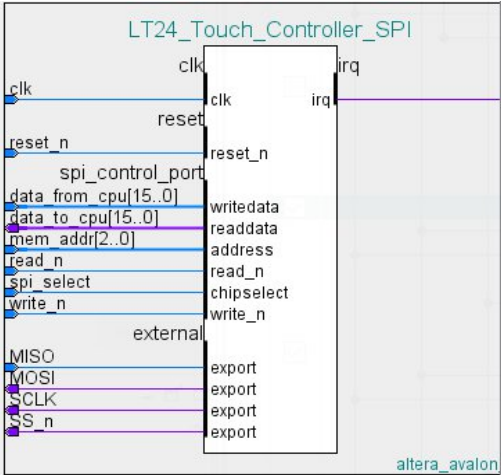
Target delay: 0.0 ns

Actual delay: 0.0 ns

Data register

Width: 8 bits

Shift direction: MSB first



Edge capture register

☒ Synchronously capture

Edge Type: FALLING

☐ Enable bit-clearing for edge capture register

Interrupt

☒ Generate IRQ

IRQ Type: EDGE

Level: Interrupt CPU when any unmasked I/O pin is logic true

Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Per quanto riguarda l'interfacciamento con il touch (e quindi con l'ADC) del display, si usa il blocco di libreria che riguarda il controllo della SPI. Il discorso sui collegamenti è simile a quello fatto con il modulo

del Display: la canalina raggruppa i segnali utili all'ADC verso l'esterno, che verranno connessi manualmente ai pin della scheda.

Anzitutto, l'ADC ha bisogno di funzionare a 10 MHz, e infatti il clock del blocco è collegato a C2 del PLL da 10 MHz. Dal momento che dobbiamo pilotare una periferica esterna, l'abbiamo configurata come master, dando 32 kHz di clock rate, così come richiesto dal datasheet di questo ADC. Inoltre, il registro a scorrimento interno della SPI usato come buffer è da 8 bit, che è una dimensione in genere abbastanza comune.

Sono necessari inoltre due fili che si occupano del segnale di periferica occupata (non essenziale nella parte SW) e dell'interrupt (che rileva il tocco). A tale scopo sono stati configurati due interfacce PIO da 1 bit con configurazione di ingresso. In particolare modo, il segnale di interrupt dispone di edge capture e ovviamente di segnale IRQ diretto verso il processore.

Quartus e top entity Verilog, timing file sdc, risultati compilazione

La top entity di Quartus, ovvero il file tris.v, è stato compilato insieme al file tris.sdc contenente i comandi per la timing analysis: lo screenshot in basso mostra le caratteristiche salienti.

```
61 //lt24 solo display
62 assign reset_n      = KEY[1]; //comandato con questo tasto
63 assign LT24_LCD_ON  = 1'b1;  //default on
64
65
66 LT24PC u0 (
67
68
69     .seg7_if_0_conduit_end_export    ({HEX5[7], HEX5[6:0], HEX4[7], HEX4[6:0],
70                                         HEX3[7], HEX3[6:0], HEX2[7], HEX2[6:0],
71                                         HEX1[7], HEX1[6:0], HEX0[7], HEX0[6:0]}),
72
73     .clk_clk                      (MAX10_CLK1_50),
74     .reset_reset_n                (reset_n),
75
76     .sdram_clk_clk                (DRAM_CLK),
77     .sdram_wire_addr              (DRAM_ADDR),
78     .sdram_wire_ba                (DRAM_BA),
79     .sdram_wire_cas_n             (DRAM_CAS_N),
80     .sdram_wire_cke               (DRAM_CKE),
81     .sdram_wire_cs_n             (DRAM_CS_N),
82     .sdram_wire_dq                (DRAM_DQ),
83     .sdram_wire_dqm               ({DRAM_UDQM, DRAM_LDQM}),
84     .sdram_wire_ras_n            (DRAM_RAS_N),
85     .sdram_wire_we_n             (DRAM_WE_N),
86
87     .pio_leds_external_connection_export    (LEDR),
88     .pio_sliders_external_connection_export (SW),
89     .pio_keys_external_connection_export    (KEY),
90
91
92     .lt24_display_reset_external_connection_export(LT24_RESET_N),
93     .lt24_display_controller_0_conduit_end_cs    (LT24_CS_N),
94     .lt24_display_controller_0_conduit_end_rs    (LT24_RS),
95     .lt24_display_controller_0_conduit_end_rd    (LT24_RD_N),
96     .lt24_display_controller_0_conduit_end_wr    (LT24_WR_N),
97     .lt24_display_controller_0_conduit_end_data  (LT24_D),
98
99     .lt24_touch_busy_external_connection_export  (LT24_ADC_BUSY),
100    .lt24_touch_irq_external_connection_export    (LT24_ADC_PENIRQ_N),
101    .lt24_touch_controller_spi_external_MISO      (LT24_ADC_DOUT),
102    .lt24_touch_controller_spi_external_MOSI      (LT24_ADC_DIN),
103    .lt24_touch_controller_spi_external_SCLK      (LT24_ADC_DCLK),
104    .lt24_touch_controller_spi_external_SS_n      (LT24_ADC_CS_N),
105
106 );
107
108
109 endmodule
```

I valori tra le parentesi sono i pin veri e propri dell'FPGA, i quali vengono collegati con gli export provenienti da Qsys. In basso i comandi per la timing analysis.


```
1
2 create_clock -period "50.0 MHz" [get_ports MAX10_CLK1_50]
3
4 derive_pll_clocks
5
6 derive_clock_uncertainty
7
```


Il risultato della compilazione è questo:


Flow Summary	
<<Filter>>	
Flow Status	EDA Netlist Writer Failed - Mon Jun 22 16:35:20 2020
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	tris
Top-level Entity Name	tris
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	5,214 / 49,760 (10 %)
Total registers	3343
Total pins	155 / 360 (43 %)
Total virtual pins	0
Total memory bits	163,688 / 1,677,312 (10 %)
Embedded Multiplier 9-bit elements	6 / 288 (2 %)
Total PLLs	1 / 4 (25 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)


Nota: la dicitura “EDA netlist writer failed” è dovuta alla mancanza di licenza del prodotto. La compilazione è eseguita con successo.


Il risultato della timing analysis è positivo, vengono rispettati i vincoli sulla frequenza massima, sul tempo di setup e di hold in tutte le condizioni di funzionamento:

Slow 1200mV 85C Model Fmax Summary			
 <<Filter>>			
	Fmax	Restricted Fmax	Clock Name
1	57.62 MHz	57.62 MHz	u0 altpll sd1 pll7 clk[2]
2	112.59 MHz	112.59 MHz	altera_reserved_tck
3	114.25 MHz	114.25 MHz	u0 altpll sd1 pll7 clk[0]
4	288.93 MHz	250.0 MHz	MAX10_CLK1_50

Slow 1200mV 85C Model Setup Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	u0 altpll sd1 pll7 clk[0]	1.247	0.000
2	MAX10_CLK1_50	16.539	0.000
3	u0 altpll sd1 pll7 clk[2]	41.323	0.000
4	altera_reserved_tck	45.559	0.000

Slow 1200mV 85C Model Hold Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	u0 altpll sd1 pll7 clk[0]	0.258	0.000
2	u0 altpll sd1 pll7 clk[2]	0.323	0.000
3	altera_reserved_tck	0.340	0.000
4	MAX10_CLK1_50	0.341	0.000

Fast 1200mV OC Model Setup Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	u0 altpll s...pll7 clk[0]	6.224	0.000
2	MAX10_CLK1_50	18.599	0.000
3	u0 altpll s...pll7 clk[2]	46.092	0.000
4	altera_reserved_tck	48.268	0.000


Fast 1200mV OC Model Hold Summary			
 <<Filter>>			
	Clock	Slack	End Point TNS
1	u0 altpll sd1 pll7 clk[0]	0.101	0.000
2	u0 altpll sd1 pll7 clk[2]	0.141	0.000
3	altera_reserved_tck	0.147	0.000
4	MAX10_CLK1_50	0.149	0.000

Parte software – Eclipse IDE for NIOS II

Come già specificato, si è usata la IDE Eclipse for Nios II come software per la programmazione della scheda. Prima di cominciare, si è creato dal file .sopcinfo creato da Qsys il progetto che include tutto il necessario per il funzionamento del computer e per l'interfacciamento col software che verrà scritto dal programmatore. Quindi si è creato il progetto sulla base di questo BSP (Board Support Package).

Anzitutto, si descriverà il funzionamento dei driver per l'LT24 dello schermo e del touch, poi si parlerà del gioco vero e proprio. Il gioco vedrà una divisione tra ciò che riguarda la grafica e ciò che riguarda la logica. Attraverso l'astrazione permessa dall'HAL (Hardware Abstraction Layer), possiamo interfacciarci con le periferiche direttamente scrivendo nei registri opportuni.

Dal BSP editor ci si è assicurati che il timer di sistema fosse abilitato (in system clock mode), e che le memorie fossero inizializzate come richiesto in Qsys:



hal

sys_clk_timer: timer_0

timestamp_timer: none

stdin: jtag_uart_0

stdout: jtag_uart_0

stderr: jtag_uart_0

☐ enable_small_c_library

☐ enable_gprof

☐ enable_reduced_device_drivers

☐ enable_sim_optimize

Linker Section Mappings				
Linker Section Name	Linker Region Name	Memory Device Name		
.bss	sdram	sdram		
.entry	reset	onchip_mem		
.exceptions	sdram	sdram		
.heap	sdram	sdram		
.rodata	sdram	sdram		
.rwdata	sdram	sdram		
.stack	sdram	sdram		
.text	sdram	sdram		

Linker Memory Regions				
Linker Region Name	Address Range	Memory Device Name	Size (bytes)	Offset (bytes)
onchip_mem	0x09004020 - 0x09006FC3	onchip_mem	12196	32
reset	0x09004000 - 0x0900401F	onchip_mem	32	0
sdram	0x04000020 - 0x07FFFFFF	sdram	67108832	32
sdram BEFORE EXCEPTION	0x04000000 - 0x0400001F	sdram	32	0

Si è successivamente abilitata la small_C_lib, consentendo un risparmio di memoria sul footprint (circa 20KB).

```
Info: (tris.elf) 183 KBytes program size (code + initialized data).
Info:          65346 KBytes free for stack + heap.
```

Driver dello schermo del LT24 (file ILI9341.c)

Piccola ricapitolazione dei segnali usati:

- CSX (attivo basso) è usato per abilitare o disabilitare l'IC della ILITEK;
- RST (attivo basso) è usato per il reset;
- WRX e RDX sono i write e read strobe che usano il data bus parallelo D[17:0];
- D/CX è il selettore per fase Command e fase Data.

Usando due macro per scrivere nel base address in command mode e data mode:

```
#define LCD_WR_REG(value) IOWR(LT24_DISPLAY_CONTROLLER_0_BASE,0x00,value) //write control command
#define LCD_WR_DATA(value) IOWR(LT24_DISPLAY_CONTROLLER_0_BASE,0x01,value) //write control data
```

Si passa brevemente a descrivere il codice commentato delle funzioni più importanti:

LCD_SetCursor

```
void LCD_SetCursor(alt_u16 Xpos, alt_u16 Ypos)
{
    LCD_WR_REG(0x002A);
    LCD_WR_DATA(Xpos>>8);
    LCD_WR_DATA(Xpos&0XFF);
    LCD_WR_REG(0x002B);
    LCD_WR_DATA(Ypos>>8);
    LCD_WR_DATA(Ypos&0XFF);
    LCD_WR_REG(0x002C);
}
```

La funzione LCD_SetCursor serve ad individuare il pixel sulla griglia da 320*240: anzitutto, con le prime due funzioni di scrittura in command mode si vanno ad individuare il Column Address Set e il Page Address Set (ovvero colonna e riga che individuano il pixel), naturalmente passando i parametri X e Y che sono le coordinate. Quindi si registra in memoria questa posizione attraverso l'ultimo comando.

Per capire meglio, è utile servirsi di questa tabella:

	D/CX	RDX	WRX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
Display OFF	0	1	↑	XX	0	0	1	0	1	0	0	0	28h
Display ON	0	1	↑	XX	0	0	1	0	1	0	0	1	29h
Column Address Set	0	1	↑	XX	0	0	1	0	1	0	1	0	2Ah
	1	1	↑	XX	SC [15:8]								XX
	1	1	↑	XX	SC [7:0]								XX
	1	1	↑	XX	EC [15:8]								XX
	1	1	↑	XX	EC [7:0]								XX
	1	1	↑	XX									
Page Address Set	0	1	↑	XX	0	0	1	0	1	0	1	1	2Bh
	1	1	↑	XX	SP [15:8]								XX
	1	1	↑	XX	SP [7:0]								XX
	1	1	↑	XX	EP [15:8]								XX
	1	1	↑	XX	EP [7:0]								XX
	1	1	↑	XX									

LCD_WR_REG quindi attiva la data mode con D/CX a 0, e scrive gli 8 bit finali in modo che si entri nella column set, quindi LCD_WR_DATA individua la colonna del pixel da selezionare usando primo e secondo parametro (SC[15:8] e SC[7:0]). Le operazioni sono commentate nel codice con riferimenti alle pagine del datasheet dell'ILI9341.

Lo stesso procedimento si ripete per il page set (posizione orizzontale). Dopodichè si memorizza il pixel individuato nella memoria dedicata.

Questa funzione base viene usata dalle funzioni più complesse che si occupano di colorare il pixel, e a sua volta dalle funzioni che disegneranno la GUI (non verranno spiegate ma sono facilmente leggibili nel codice sorgente).

LCD_ClearScreen

```
void LCD_Clear(alt_u16 Color)
{
    alt_u32 index=0;
    LCD_SetCursor(0x00,0x0000);
    LCD_WR_REG(0x002C);
    for(index=0;index<76800;index++)
    {
        LCD_WR_DATA(Color);
    }
}
```

Questa funzione semplicemente passa per tutti i pixel e li colora del colore passato come parametro, nel caso si volesse "sbiancare", si passa il colore nero.

LCD_Init

Lo screenshot di questa funzione non viene allegato perché di grandi dimensioni. Sostanzialmente, si fa partire la routine di inizializzazione dell'LCD, che consiste sempre nell'individuare le funzioni per la predisposizione al funzionamento del dispositivo attraverso la command mode. I comandi sono commentati nel codice, specificandone le pagine del datasheet del ILI9341.

Driver del touch del LT24 (file touch_spi.c)

Prima di continuare, si ricapitola il significato dei bit di controllo:

- S (bit 7) è il bit di start della control word corrispondente a DIN. Si può far partire una nuova control word ogni 15 cicli di clock quando si seleziona la conversione a 12 bit (quella usata in questo progetto);
- A2-A0 (bit da 6 a 4) sono i bit di selezione del canale, e assieme al SER/~DEF controllano le impostazioni del multiplexer, degli switch e dei riferimenti;
- MODE (bit 3) è il selettore della risoluzione del convertitore, nello specifico viene posto a 0 per la profondità a 12 bit;
- SER/~DFR (bit 2) è il bit di selezione per single-ended/differential reference. Verrà posto nel nostro progetto sulla differential reference (posto a 0) e come già detto seleziona la modalità di funzionamento del dispositivo. Il funzionamento sarà brevemente spiegato fra poco;
- PD1 e PD0 (bit 1 e bit 0) riguardano i bit del power management del AD7843.

Anzitutto, si è creata una struttura dati:

```
typedef struct{
    alt_u32 spi_base;
    alt_u32 penirq_base;
    alt_u32 penirq_irq;
    alt_u32 irq_mask;
    alt_u16 fifo_front;
    alt_u16 fifo_rear;
    alt_u16 fifo_x[FIFO_SIZE];
    alt_u16 fifo_y[FIFO_SIZE];
    bool pen_pressed;
    alt_alarm alarm;
    alt_u32 alarm_dur;
    alt_u32 next_active_time;
}TERASIC_TOUCH_PANEL;
```

Questa struttura raccoglie tutte le caratteristiche del touch, che saranno memorizzate dalle funzioni a seguire. Viene quindi definito un "oggetto" di tipo TERASIC_TOUCH_PANEL.

Touch_Init

```
TOUCH_HANDLE Touch_Init(const alt_u32 spi_base, const alt_u32 penirq_base, const alt_u32 penirq_irq){
    bool bSuccess = TRUE;
    Terasic_Touch_Panel *p;
    p = malloc(sizeof(TERASIC_TOUCH_PANEL));
    if (!p)
        return p;
    memset(p, 0, sizeof(TERASIC_TOUCH_PANEL));
    p->spi_base = spi_base;
    p->penirq_base = penirq_base;
    p->irq_mask = 0x01; // 1-pin
    p->penirq_irq = penirq_irq;
    p->alarm_dur = alt_ticks_per_second()/SAMPLE_RATE;

    // enable penirq_n interrupt (P1=1, P1=0) and the device
    touch_enable_penirq(p);
    // enable interrupt, 1-pin
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(p->penirq_base, p->irq_mask);
    // Reset the edge capture register
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(p->penirq_base, 0);
    // register ISR
    // register callback function
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
    if ((alt_ic_isr_register(LT24_TOUCH_CONTROLLER_SPI_IRQ_INTERRUPT_CONTROLLER_ID,
    )else{
        DEBUG_OUT(("[TOUCH]register IRQ success\n"));
    }
    if (bSuccess){
        if (alt_alarm_start(&p->alarm, p->alarm_dur, touch_alarm_callback, p) == 0){
            DEBUG_OUT(("[TOUCH]alarm start success\n"));
        }else{
            DEBUG_OUT(("[TOUCH]alarm start fail\n"));
            bSuccess = FALSE;
        }
    }
    if (!bSuccess && p){
        free(p);
        p = NULL;
    }
    return p;
}
```

touch_isr

Questa funzione serve ad inizializzare il touch: si fanno passare come parametri i registri della periferica, quindi si associano alla nuova struttura dinamicamente allocata *p. Quindi si passa alla inizializzazione dell'interrupt. Se è tutto corretto, viene creata l'istanza di TERASIC_TOUCH_PANEL.

Si verifica anche che l'interrupt funzioni registrando l'ISR ed abilitando l'interrupt. La routine specifica per il device è:

```
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
void touch_isr(void* context){
    alt_u8 mask;
    Terasic_Touch_Panel *p = (TERASIC_TOUCH_PANEL *)context;
#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
    alt_ic_irq_disable(LT24_TOUCH_CONTROLLER_SPI_IRQ_INTERRUPT_CONTROLLER_ID, LT24_TOUCH_CONTROLLER_SPI_IRQ);
#else
    // get the edge capture mask
    mask = IORD_ALTERA_AVALON_PIO_EDGE_CAP(p->penirq_base);
    DEBUG_OUT("Touched!!\n");
    usleep(1000);
//#if 0
    // Reset the edge capture register
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(p->penirq_base, 0);

#ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
    alt_ic_irq_enable(LT24_TOUCH_CONTROLLER_SPI_IRQ_INTERRUPT_CONTROLLER_ID, LT24_TOUCH_CONTROLLER_SPI_IRQ);
#else
    alt_irq_enable(id);
#endif
}
}
```

Inoltre, servendosi delle funzioni base HAL del timer istanziato, si esegue una callback della funzione seguente per verificare che venga richiamata periodicamente:

```
alt_u32 touch_alarm_callback(void *context){    //con interval timer
    TERASIC_TOUCH_PANEL *p = (TERASIC_TOUCH_PANEL *)context;

    if (touch_is_pen_pressed(p)) {
        if (alt_nticks() > p->next_active_time)
            touch_get_xy(p);
    } else {
        touch_enable_penirq(p);
        //touch_clear_input(p);
    }
    return p->alarm_dur;
}
```

Sostanzialmente, questa funziona sonda per trovare le coordinate quando il l'interrupt è funzionante, in caso contrario, lo abilita.

Touch_empty_fifo

```
void touch_empty_fifo(TERASIC_TOUCH_PANEL *p) {
    p->fifo_rear = p->fifo_front;
}
```

Questa funzione serve a svuotare la memoria del tocco effettuato. Le due memorie FIFO istanziate nel TERASIC_TOUCH_PANEL servono infatti a memorizzare il tocco, e funzionano unitamente: in generale, appena lo schermo viene toccato, la front viene riempita. Per renderla di nuovo utilizzabile all'utente con una nuova posizione, è necessario svuotarla su un buffer ausiliario rear. Questa è una funzione base che verrà implementata da TouchEmptyFiFo. Vedremo nel codice del gioco come questa operazione verrà effettuata.

Touch_get_xy

Prima di mostrare frammenti di codice, è utile vedere come il registro può essere modificato:

MSB					LSB		
S	A2	A1	A0	MODE	SER/DFR	PD1	PD0

Bit	Mnemonic	Comment
7	S	Start Bit. The control word starts with the first high bit on DIN. A new control word can start every 15th DCLK cycle when in the 12-bit conversion mode, or every 11th DCLK cycle when in 8-bit conversion mode.
6–4	A2–A0	Channel Select Bits. These three address bits, along with the SER/DFR bit, control the setting of the multiplexer input, switches, and reference inputs, as described in Table 5.
3	MODE	12-Bit/8-Bit Conversion Select Bit. This bit controls the resolution of the following conversion. With 0 in this bit, the conversion has a 12-bit resolution, or with 1 in this bit, the conversion has a 8-bit resolution.
2	SER/DFR	Single-Ended/Differential Reference Select Bit. Along with Bits A2–A0, this bit controls the setting of the multiplexer input, switches, and reference inputs, as described in Table 5.
1, 0	PD1, PD0	Power Management Bits. These two bits decode the power-down mode of the AD7843, as shown in Table 7.

Table 5. Analog Input, Reference, and Touch Screen Control

A2 ¹	A1 ¹	A0 ¹	SER/DFR	Analog Input	X Switches	Y Switches	+REF ²	–REF ²
0	0	1	1	X+	OFF	ON	V _{REF}	GND
0	1	0	1	IN3	OFF	OFF	V _{REF}	GND
1	0	1	1	Y+	ON	OFF	V _{REF}	GND
1	1	0	1	IN4	OFF	OFF	V _{REF}	GND
0	0	1	0	X+	OFF	ON	Y+	Y–
1	0	1	0	Y+	ON	OFF	X+	X–
1	1	0	0	Outputs Identity Code, 1000 0000 0000				

```
void touch_get_xy(TERASIC_TOUCH_PANEL *p){
    alt_u16 x, y;
    int result;
    const alt_u8 CommandGetX = 0x92;
    const alt_u8 CommandGetY = 0xD2;
    alt_u16 ResponseX, ResponseY;
    alt_u8 high_byte, low_byte;

    ...

    ...
}
```

Servendoci delle tabelle in alto, scrivendo 0x92 nel registro, che equivale alla sequenza binaria 10010010, allora con questo comando andiamo a prelevare la coordinata x del tocco nel modo differenziale permesso dal dispositivo. Allo stesso modo, il comando 0xD2 serve per la coordinata y.


```

result = alt_avalon_spi_command(p->spi_base, 0, sizeof(CommandGetX), &CommandGetX, 0, 0, ALT_AVALON_SPI_COMMAND_MERGE);
result = alt_avalon_spi_command(p->spi_base, 0, 0, 0, sizeof(high_byte), (alt_u8*)&high_byte, ALT_AVALON_SPI_COMMAND_MERGE);
if (result != sizeof(high_byte)){
    DEBUG_OUT("[TOUCH] failed to get x\n");
    return;
}
result = alt_avalon_spi_command(p->spi_base, 0, 0, 0, sizeof(low_byte), (alt_u8*)&low_byte, ALT_AVALON_SPI_COMMAND_TOGGLE_SS_N);
if (result != sizeof(low_byte)){
    DEBUG_OUT("[TOUCH] failed to get x\n");
    return;
}
ResponseX = (high_byte << 8) | low_byte;

```

Si acquisisce quindi la coordinata x accedendo in scrittura e verificando che effettivamente sia corretta sul registro della SPI. Inoltre, dal comando scritto, si istruisce il dispositivo ad inviare il segnale di interrupt dato che è stato toccato. Stessa cosa si fa per l'ordinata.

In basso invece:

```

if (!touch_is_pen_pressed(p))
    return; // do not use this data

x = (ResponseX >> 3) & 0xFFF; // 12 bits
y = (ResponseY >> 3) & 0xFFF; // 12 bits

// check whether the fifo is full!
if (((p->fifo_front+1)%FIFO_SIZE) == p->fifo_rear){
    // full, pop an old one
    p->fifo_rear++;
    p->fifo_rear %= FIFO_SIZE;
}

DEBUG_OUT("[ ADC] x=%d, y=%d\n", x, y);

// push now
p->fifo_x[p->fifo_front] = x;
p->fifo_y[p->fifo_front] = y;
p->fifo_front++;
p->fifo_front %= FIFO_SIZE;
}

```

Le coordinate vengono quindi codificate su 12 bit così come previsto dalla scrittura sul registro di controllo, quindi viene verificato se è possibile la scrittura sul front buffer, se questo fosse già occupato allora viene liberato, quindi vengono scritti i valori nel front.

Touch_xy_transform

Questa funzione (fornita da Terasic) serve a digitalizzare in maniera corretta il tocco per l'ADC del LT24.

```
void touch_xy_transform(int *x, int *y) {
    int xx, yy;
    const int y_ignore = 200;
    xx = *y;
    yy = *x;

    // scale & swap
    // xx = 4096 - 1 - xx;
    xx = xx * X_RES / 4096;           //profondità ADC a 12 bit

    // yy = 4096 - 1 - yy;

    // special calibrate for LT24
    if (yy > (4096-y_ignore))
        yy = 4096-y_ignore;
    yy = yy * 4095 / (4096-y_ignore);

    yy = yy * Y_RES / 4096;

    // swap
    *x = xx;
    *y = yy;
}
```

Le funzioni viste verranno richiamate da funzioni più generiche usate all'interno del gioco.

Non vengono allegati screenshot riguardanti funzioni per la grafica e i font dato che sono ingombranti. Sono facilmente leggibili dal codice sorgente.

Logica di gioco (file main.c e game.c)

File utili per capire come si è progettato il gioco (e i relativi headers .h):

- Simple_graphics.c;
- Simple_text.c;
- TahomaBold20.c;
- Seg7.c;
- Geometry.c;
- Gui_game.c.

Anzitutto, dal main si inizializzano schermo e touch del LT24, si passa alla schermata di benvenuto, quindi si passa al gioco.

Il gioco è diviso in più stati, che sono (state) START, PLAY, RESULT e GAMEOVER:

- In START, semplicemente si permette ai due giocatori di scegliere tra le figure messe a disposizione;
- In PLAY si effettuano i passaggi di turno tra un giocatore e l'altro;
- In RESULT si verifica la vincita o il pareggio;
- In GAMEOVER vi è l'attesa per selezionare un'altra partita con lo stesso giocatore oppure decidere un nuovo gioco che riporterà alla schermata di START premendo il tasto EXIT.

A sua volta, ci si serve di una macchina a stati secondaria con (player) PLAYER1, PLAYER2, CHECK1 e CHECK2. In particolare si fa uso di questa nello stato di GAME.

- In PLAYERx avviene l'immissione della figura in una casella;
- in CHECKx avviene il controllo della combinazione vincente.

```
typedef enum {START, PLAY, RESULT, GAMEOVER} state_t;
typedef enum {PLAYER1, PLAYER2, CHECK1, CHECK2} player_t;
```

I giocatori vengono identificati attraverso delle strutture semplici PLR di identificazione denominate "player1ID" e "player2ID":

```
typedef struct{
    int fig;
    //vettore per le combinazioni vincenti del giocatore
    int comb[9];
    //int color;
    //altre caratteristiche associabili
}PLR;
```

La valutazione della vittoria si basa sul semplice concetto di riempire un array (comb) in modo tale da far vincere l'uno o l'altro giocatore a seconda della combinazione.

In basso ci sono frammenti dei punti salienti del gioco:

START: selezione giocatore

```
while(1){
    switch(state){
        case START:    //scelta figure

            Touch_EmptyFifo(pTouch);
            //disegno figure scelta
            GUI_DrawPlayerSel(pDisplay);
            GUI_ShowPlayerSelect1(pDisplay);

            while(!chosen1){
                if(Touch_GetXY(pTouch, &X, &Y)){
                    PtSet(&Pt, X, Y);
                    Touch_EmptyFifo(pTouch);
                    if (IsPtInRect(&Pt, &r3)){
                        player1ID.fig = BTN_QUADRATO;
                        vid_draw_box (r1.left, r1.top, r1.right, r1.bottom, BLACK_24, DO_FILL, pDisplay);    //per oscurare
                        chosen1=1;
                    }
                    else if(IsPtInRect(&Pt, &r4)){
                        player1ID.fig = BTN_CERCHIO;
                        vid_draw_box (r2.left, r2.top, r2.right, r2.bottom, BLACK_24, DO_FILL, pDisplay);
                        chosen1=1;
                    }
                    else if(IsPtInRect(&Pt, &r5)){
                        player1ID.fig = BTN_CROCE;
                        vid_draw_box (r3.left, r3.top, r3.right, r3.bottom, BLACK_24, DO_FILL, pDisplay);
                        chosen1=1;
                    }
                    else{
                        vid_print_string_alpha(20, 20, YELLOW_24, BLACK_24, tahomabold_20, pDisplay, "WRONG CHOICE");
                        usleep(500*1000);
                        vid_clean_top(pDisplay, BLACK_24);
                    }
                }
            }
        }
    }
    ...
}
```

Chosen1 rappresenta il flag di avvenuta identificazione del giocatore corrente.

PLAY: gioco e “palleggio” tra gli stati della macchina a stati secondaria

```
case PLAY:

    if(!grid){
        GUI_DrawGrigliaTris(pDisplay);
        grid=1;
    }
    usleep(1000*1000);
    Touch_EmptyFifo(pTouch);

    while(moveCounter && !winFlag){    //finchè ci sono mosse disponibili e qualcuno non ha vinto

        switch(player){

            case (PLAYER1):

                vid_print_string_alpha(GAMEWRITES_H, GAMEWRITES_V, WHITE_24, BLACK_24, tahomabold_20, pDisplay, "P1 turn");
                if (Touch_GetXY(pTouch, &X, &Y)){
                    PtSet(&Pt, X, Y);
                    if (IsPtInRect(&Pt, &sqExit)){
                        winFlag=1;
                        state=GAMEOVER;
                    }
                    else if (IsPtInRect(&Pt, &r0) && !player1ID.comb[0] && !player2ID.comb[0]){
                        switch(player1ID.fig){
                            case BTN_QUADRATO: GUI_PutRect(pDisplay,0, szPallette[BTN_QUADRATO]);break;
                            case BTN_CERCHIO: GUI_PutCircle(pDisplay,0, szPallette[BTN_CERCHIO]);break;
                            case BTN_CROCE: GUI_PutCross(pDisplay,0, szPallette[BTN_CROCE]);break;
                            default:break;
                        }

                        player1ID.comb[0] = 1;    //cioè quella posizione è occupata adesso
                        moveCounter--;
                        player=CHECK1;
                    }
                    else if (IsPtInRect(&Pt, &r1) && !player1ID.comb[1] && !player2ID.comb[1]){
                        ...
                    }
                }
            }
        }
    }
}
```

Come si vede, a seconda di dove viene toccato lo schermo, verrà inserita la figura associata al giocatore in quella casella. Quindi viene occupata tale casella nell'array delle combinazioni di quel giocatore e si passa allo stato CHECK1 sempre della macchina secondaria.

I comandi del frammento in basso sono una porzione delle combinazioni che portano alla vittoria di un giocatore, assegnandone quindi un flag di vittoria che fa fermare il ciclo e porta all'entrata nello stato di RESULT. In caso contrario, si passa al turno dell'altro giocatore.

```
case(CHECK1): //inizio CHECK su player1
    if(player1ID.comb[0]&&player1ID.comb[1]&&player1ID.comb[2]){
        vid_draw_line(0, 40, 240, 40, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[0]&&player1ID.comb[3]&&player1ID.comb[6]){
        vid_draw_line(40, 0, 40, 240, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[6]&&player1ID.comb[7]&&player1ID.comb[8]){
        vid_draw_line(0, 200, 240, 200, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[2]&&player1ID.comb[5]&&player1ID.comb[8]){
        vid_draw_line(200, 0, 200, 240, 2, YELLOW_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[1]&&player1ID.comb[4]&&player1ID.comb[7]){
        vid_draw_line(120, 0, 120, 240, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[3]&&player1ID.comb[4]&&player1ID.comb[5]){
        vid_draw_line(0, 120, 240, 120, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[0]&&player1ID.comb[4]&&player1ID.comb[8]){
        vid_draw_line(0, 0, 240, 240, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else if(player1ID.comb[2]&&player1ID.comb[4]&&player1ID.comb[6]){
        vid_draw_line(240, 0, 0, 240, 2, BLUE_24, pDisplay);
        winFlag=1;
        win1=1;
    }
    else
        player=PLAYER2;
    ...
```

Si passa quindi allo stato di verifica di vittoria o pareggio della macchina principale.

RESULT:

```
case RESULT:

    //controllo vittorie o pareggio
    if(win1){
        winCount1++;
        SEG7_Score(FIRST7SEG, winCount1);
        GUI_ShowPlayerResult(pDisplay, "Player1 WINS");
        usleep(1*1000*1000);
    }
    else if(win2){
        winCount2++;
        SEG7_Score(LAST7SEG, winCount2);
        GUI_ShowPlayerResult(pDisplay, "Player2 WINS");
        usleep(1*1000*1000);
    }

    else if(!moveCounter){ //quando è 0
        GUI_ShowPlayerResult(pDisplay, "TIE");
        usleep(1*1000*1000);
    }
    else {
        GUI_ShowPlayerResult(pDisplay, "Unhandled G.O.");
        usleep(3*1000*1000);
    }

    state=GAMEOVER;
    break;
    ...

    ...
```

Si passa quindi allo stato di gameover.

GAMEOVER:

```
case GAMEOVER:

    Touch_EmptyFifo(pTouch);
    //animato
    while(!Touch_GetXY(pTouch, &X, &Y)){
        vid_print_string_alpha(GAMEWRITES_H, GAMEWRITES_V, WHITE_24, BLACK_24, tahomabold_20, pDisplay, "Gameover");
        usleep(1000*1000);
        vid_print_string_alpha(GAMEWRITES_H, GAMEWRITES_V, WHITE_24, BLACK_24, tahomabold_20, pDisplay, "PressAnywhere");
        usleep(1000*1000);
    }

    Touch_EmptyFifo(pTouch);
    while (!Touch_GetXY(pTouch, &X, &Y));
    PtSet(&Pt, X, Y);
    if (IsPtInRect(&Pt, &sqExit)){ //ritorno alla selezione giocatore solo se si reinizializza il gioco
        winCount1=0;
        winCount2=0;
        SEG7_Score(FIRST7SEG, winCount1);
        SEG7_Score(LAST7SEG, winCount2);
        player1ID.fig=0;
        player2ID.fig=0;
        state=START;
    }
    else { //oppure semplice ritorno a nuova partita
        state=PLAY;
    }

    Touch_EmptyFifo(pTouch);
    int i; //azzeramento variabili
    for (i=0;i<9;i++){
        player1ID.comb[i] = 0;
        player2ID.comb[i] = 0;
    }
    moveCounter = MOVES_MAX;
    winFlag = 0;
    win1=0;
    win2=0;
    contatoreTocchi = 0;
    grid=0;
    vid_clean_screen(pDisplay, BLACK_24);

    usleep(500*1000);
    player=PLAYER1;
```

Come già accennato, se si tocca sul tasto EXIT si ritorna alla selezione del giocatore, altrimenti qualsiasi altro tocco reinizializza la partita con gli stessi giocatori (e le variabili necessarie).

Conclusioni e note

Il sistema è funzionante fluidamente e può essere riutilizzabile per altri giochi oppure altri tipi di progetti.

La scelta del processore non è stata intuitiva come si potesse sperare: la parte software del gioco richiede risorse minime di funzionamento, ma dal punto di vista hardware le funzionalità aggiuntive del processore fast si sono rese quasi essenziali al funzionamento del gioco.

Rispetto al video di demo, è presente qualche animazione aggiuntiva.

Per quanto riguarda la parte HW del progetto, si è fatto uso di referenze di datasheet, mentre per il software, sono state usate alcune librerie che sono state messe a disposizione da Altera.

Bibliografia

1. *ILI3241.pdf ILITEK display datasheet;*
2. *AD7843.pdf Analog Devices touch ADC;*
3. *N2cpu_nii5v1gen2.pdf NIOS Reference guide;*
4. *Ug_embedded_ip.pdf;*
5. *n2cpu_nii51004.pdf Instanciate Nios processor;*
6. *qsys_system_components.pdf Qsys;*
7. *LT24_manual.pdf Display Touch;*
8. *SOPc_builder.pdf;*
9. *DE10_lite_computer_manual.pdf.*