Misuratore di distanza ad ultrasuoni di Naclerio Pasquale

Consegna

Attraverso un circuito logico su DE1 pilotare opportunamente un sensore HC-SR04 ad ultrasuoni per misurare la distanza tra il sensore ed un oggetto. Successivamente salvare il dato sulla memoria di un processore implementato sempre sulla DE1 e che poi verrà prelevato per essere visualizzato sul display a 7 segmenti.

Premessa sul sensore

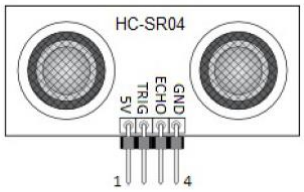

Figura 1: disegno del sensore.

Per la misurazione della distanza attraverso gli ultrasuoni ho usato un comune sensore HC-SR04, è un sensore digitale che usa due pin per la misurazione, un pin di Trigger che invia il segnale per la misura e un pin di ECHO che riceve il segnale di risposta. Il sensore permette di misurare distanze che vanno dai 450 cm ai 2 cm con risoluzione di 3mm.

Caratteristiche tecniche:
Tensione di lavoro: 5 Vdc.
Corrente assorbita: >2 mA.
Frequenza di lavoro: 40 Khz.
Distanza max: 450 cm.
Distanza min: 2 cm.
Risoluzione: 3 mm.
Angolo di misura: 15°.
Ingresso: Trigger 10us Impulso TTL.
Uscita: Echo segnale PWM TTL. [4]

Funzionamento:

1. Si invia un impulso di 10 us sul pin di Trigger.
   1.1. Il sensore invierà 8 impulsi ad ultrasuoni ad una frequenza di 40 Khz.
   1.2. Si attende circa 50-60 us prima di inviare un nuovo impulso di 10us sul pin di Trigger.
2. Il sensore risponde sul pin Echo con un impulso alto della durata corrispondente a quella

di viaggio delle onde sonore.

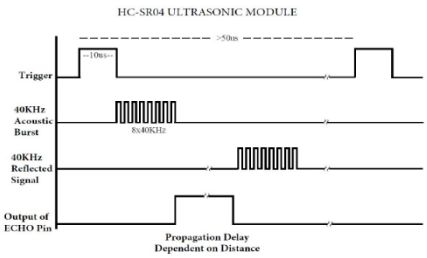

Figura 2: rappresentazione dei segnali di pilotaggio del sensore.

Per calcolare lo spazio percorso occorre conoscere la durata del segnale di Echo e usarlo nella formula:

$$Spazio = \frac{Tempo \times Velocità\ del\ Suono}{2}$$

La velocità del suono in ara è 343,4 m/s a 20 °C cioè 0,0343 cm/microsecondi. Divido per due in quando devo tenere conto del tempo di andata e ritorno del segnale dall'oggetto della misura. Ottengo così una costante di 0,01715 oppure 1/58. La formula diventa:

$$Spazio = \frac{Tempo}{58}$$

Schema Generale del sistema di misura.

Il sistema in generale si compone da due generatori di clock da 200kHz e 1MHz che sulla loro base andranno a costruire un segnale di Trigger e un clock che fornirà la base su cui si misurerà il tempo del segnale di Echo. L'uscita del conteggio del segnale di Echo sarà un numero di campioni, campionato a una frequenza di 1MHz che corrisponde al numero di millisecondi del segnale di Echo. Questo valore viene quindi dato alla RAM del processore che successivamente andrà a prelevarlo per calcolare lo spazio in centimetri e visualizzarlo sul display a 7 segmenti.

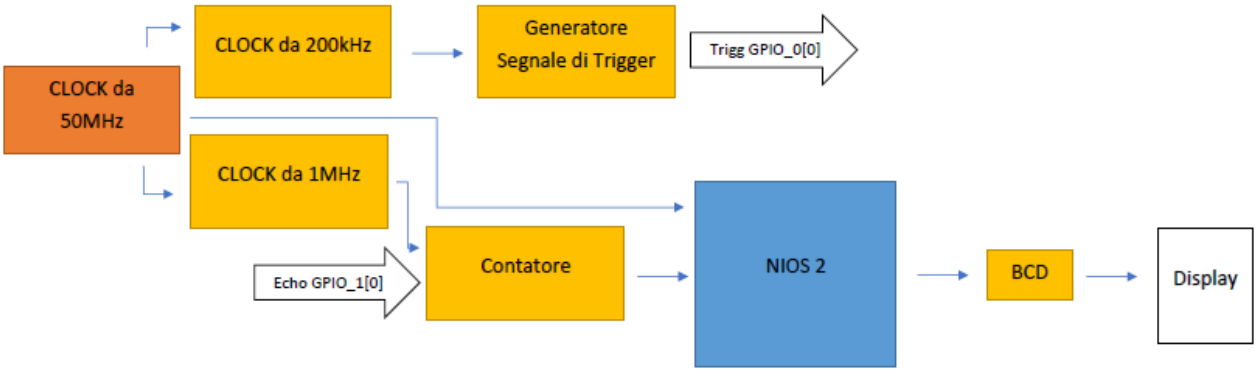

Figura 3: schema generale del sistema di misura.

Generazione del Segnale di Clock da 1MHz e 200kHz.

In mancanza di un generatore di segnali è stata implementata un modulo che generi una frequenza impostata da un segnale di clock dell'FPGA. Questo modulo è stato denominato gen_sig; dal clock

della scheda DE1 (50MHz) genera un clock ad una frequenza scelta. Per farlo si farà un contatore che conta in dietro da una costante fino a zero. Questa costante determina il numero di impulsi di clock che devono stare all'interno del fronte di salita del segnale di clock da generare. Questa costante che definiremo con C è determinata dalla metà del numero di impulsi del clock in input in un secondo e dalla frequenza desiderata.

$C = \frac{(\frac{n.impulsi\ clock\ input\ al\ secondo}{2} - 1)}{frequenza\ clock\ out}$. Da un clock di 50MHz voglio ottenere un clock a 20Hz allora avrò $C = \frac{24999999}{20} = 1250000$, questi sono gli impulsi di clock di input che dovrò contare prima di far finire il fronte di salita del clock di output.

```
module gen_sig(clock, trig);
input clock;
output reg trig;
parameter freq = 1;
reg [24:0] counter;

initial begin
        counter = 0;
        trig = 0;
end
always@ (posedge clock) begin
        if (counter == 0) begin
                counter <= 24999999/freq;
                trig <= ~trig;
        end
        else begin
        counter <= counter -1;
        end
end
endmodule
```

Figura 4: codice verilog per il generatore del segnale di clock a frequenza desiderata.

Con questo modulo si è quindi generato il segnale di test desiderato, da 1MHz e da 200kHz. Parametrizzando in fine il valore freq si è potuto quindi realizzare un modulo facilmente implementabile e modificabile secondo il bisogno senza dover mai cambiare il codice del modulo.

Generatore Segnale di Trigger.

Devo generare un segnale che è alto per 10us e basso per 40 us almeno in modo che complessivamente il segnale risulti di durata 50us .

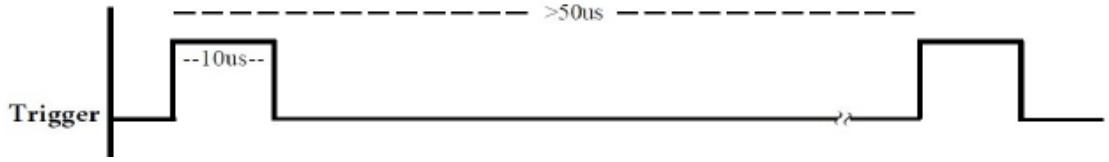

Figura 5: dettaglio del segnale di trigger necessario per pilotare il sensore.

Allora l'idea è quella di usare due contatori, uno che conti [cont1] da 0 a 1 a una frequenza di 200kHz e cioè con un periodo di 5us in modo da avere due periodi dentro il segnale alto e quindi generare un segnale alto della durata di 10us e quindi di frequenza 100kHz. Il secondo contatore [cont5] conta da 0 a 4 alla frequenza di 100kHz generata precedentemente ma questa volta resta basso per tutto il conteggio. Così facendo si ottiene un segnale di Trigger che soddisfa le specifiche.

```
module trigger(clock, reset, enable, trig);
input clock, reset, enable;
output trig;
wire t0;
count1 contatore1(
        .clk(clock),
        .enable(enable),
        .reset(reset),
        .clk_out(t0)
);
count5 contatore2(
        .clk(t0),
        .enable(enable),
        .reset(reset),
        .clk_out(trig)
);
endmodule
```

Figura 6: codice verilog per la generazione del segnale di trigger.

```
module count1
	(
		input clk, enable, reset,
		output reg clk_out,
		output reg [7:0] count
	);

	always @ (posedge clk or posedge reset)
	begin
		if (reset)
		count <= 0;
		else if (enable == 1'b1)
				begin
					if (count < 1)
						begin
							count <= count + 1;
							clk_out <= 1'b0;
						end
					else
						begin
							count <= 0;
							clk_out <= 1'b1;
						end
				end
	end
endmodule
```

Figura 7: codice verilog per la generazione della parte alta del segnale di trigger da 10us.

```
module count5
	(
		input clk, enable, reset,
		output reg clk_out,
		output reg [7:0] count
	);

	always @ (posedge clk or posedge reset)
	begin
		if (reset)
		count <= 0;
		else if (enable == 1'b1)
				begin
					if (count < 4)
						begin
							count <= count + 1;
							clk_out <= 1'b0;
						end
					else
						begin
							count <= 0;
							clk_out <= 1'b1;
						end
				end
	end
endmodule
```

Figura 8: codice verilog per la generazione della parte bassa del segnale di trigger da 40us.

Contatore per il segnale di Echo.

Il contatore per il segnale di Echo riceve la finestra di segnale su cui fare il conteggio. Ho quindi implementato un semplice contatore che conta fino a che il segnale resta alto e successivamente restituisce un impulso di fine conteggio [clk_out] e l'ultimo valore del conteggio [tempo_10us]. La variabile è[tempo_10us] è a 16 bit in quanto la misura massima che si può eseguire è di 240 cm e quindi il valore massimo di conteggio è 13920. Grazie al clock da 1MHz ho che il numero del conteggio è il tempo in microsecondi della durata del mio segnale di Echo.

```
module echo
        (
                input clk, enable, reset,
                output reg clk_out,
                output reg [15:0] tempo_10us);
        reg [15:0] cont;
        always @ (posedge clk or posedge reset)
        begin
                if (reset)
                tempo_10us <= 0;
                else if (enable == 1'b1)

                                begin
                                                cont <= cont + 1;
                                                tempo_10us <= cont + 1;
                                                clk_out <= 1'b1;
                                                end
                                else
                                                begin
                                                        cont <= 0;
                                                        clk_out <= 1'b0;
                                                end
                                end
endmodule
```

Figura 9: codice verilog per il conteggio di impulsi da 1us contenuti dentro il segnale di echo ricevuto per il calcolo del tempo impiegato al segnale di trigger a tornare al sensore.

Processore Nios II.

Utilizzando il tutorial 4 [3] ho realizzato un semplice processore Nios II, con una RAM da 16384 Bytes e un Data Witdh da 32 bits, un interfaccia JTAG UART e due interfacce parallele di I/O da 16 bits, una di ingresso per il conteggio del numero di microsecondi e una per l'uscita con il valore della distanza spaziale calcolata dal processore.

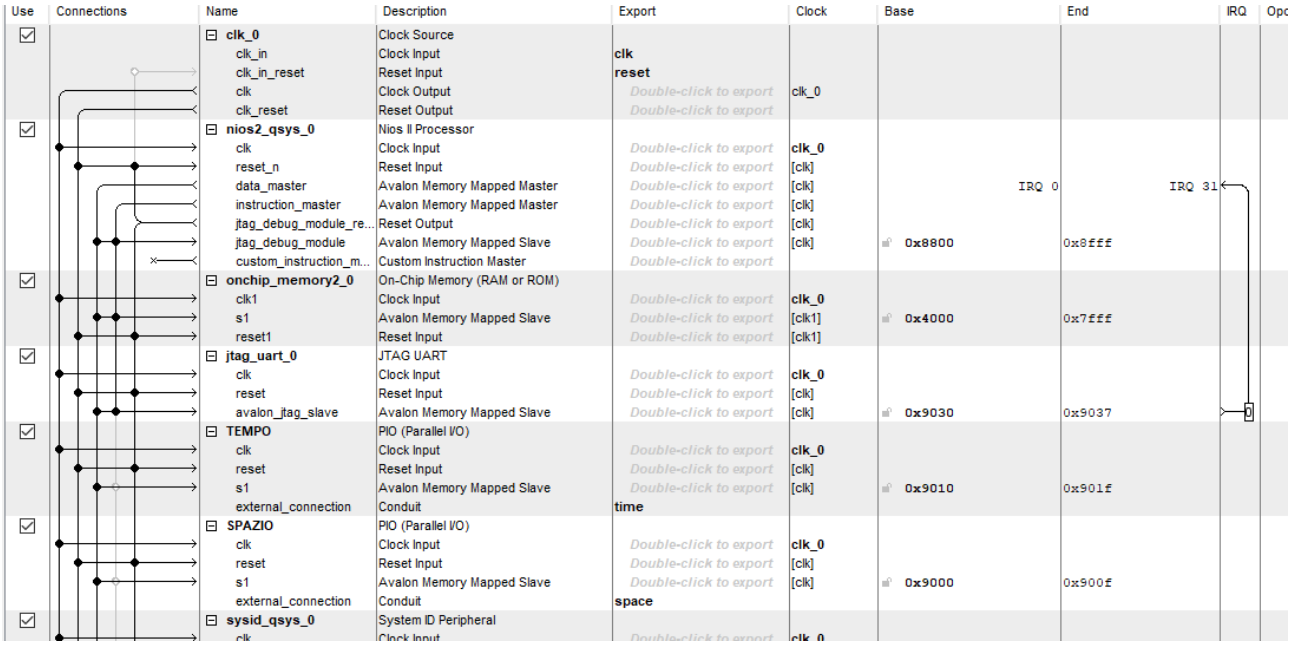

Figura 10: processore Nios II in fase di implementazione su Qsys.

Questo processore sarà programmato con un semplice programma in C che andrà a prelevare i dati di conteggio nella RAM effettuerà la divisione per il fattore 58, ottenendo così la misura di distanza in centimetri. Salverà la misura in una nuova locazione della RAM che corrisponde all'uscita per output [space]. L'uso di un processore permette di demandare ad una elettronica di controllo esterna al processore il compito di misurare la distanza e il processore può così semplicemente leggere il dato ed effettuare calcoli più consoni ad un processore. Il processore quindi andrà solo a leggere il dato in RAM ed eseguire il calcolo o la gestione di eventuali altre periferiche, senza dover preoccuparsi di generare i segnali opportuni per la misura.

```
#define time (volatile int *) 0x0009010

#define space (int *) 0x0009000

        int main() {

        int a;

        while (1)

                {a = *time/58;

                *space = a;}}
```

BCD

Dal processore ho un valore da 16 bit che andrò a convertire in 32 bit, attraverso un modulo apposito. L'informazione contenuta è la distanza in centimetri, questo valore viene convertito attraverso un modulo BCD. Il modulo è stato scritto in moda da poter convertire numeri fino a decine di milioni restituendo 4 bit per ogni cifra significativa, lo fa seguendo semplicemente l'algoritmo di conversione. Nel mio caso avendo solo 4 display a 7 segmenti le uscite a 4 bit che andranno nel modulo di controllo dei display saranno solo le unità, le decine , le centinaia e le migliaia.

Display a 7 segmenti

In fine l'ultimo modulo è il modulo del tutorial 2 [2] che va a pilotare i display. Trasforma le stringe di 4 bit che arrivano dal convertitore BCD e le trasforma nelle stringe a 7 bit necessarie per pilotare i display in modo opportuno. La parte rilevante è la temporizzazione di visualizzazione dei display, infatti il clock che serve per temporizzarli è il clock del segnale finestra ma negato in modo da avere la visualizzazione solo alla fine del conteggio. Questo punto ha avuto particolare attenzione per avere la migliore visualizzazione e stabilità di ciò che è rappresentato.

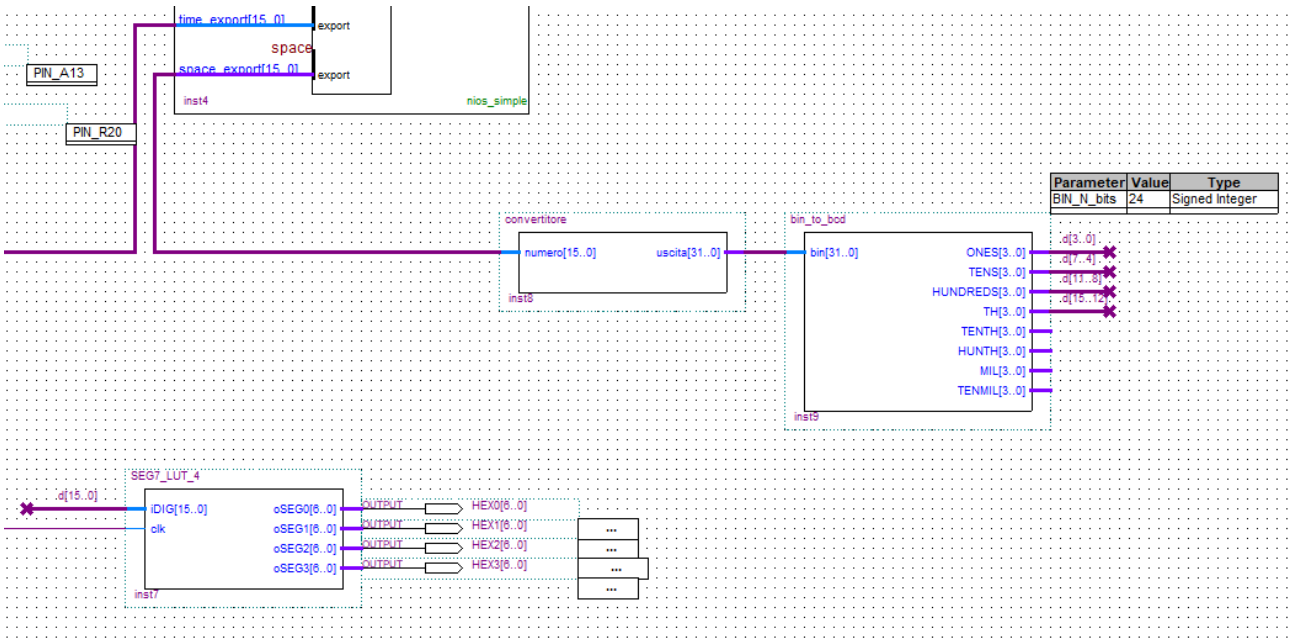

Figura 11: particolare dello schematico in cui si vedono il modulo di conversione a 32 bits, il convertitore in BCD e il modulo demandato al controllo del display a 7 segmenti.

GPIO, generazione e acquisizione del segnale.

Utilizzando il manuale DE1_reference_manual [1] ho potuto assegnare i pin di ingresso e di uscita per la GPIO. La DE1 ha due GPIO, la GPIO_0 e GPIO_1, la prima lo usata per i segnali di uscita del segnale di Trigger la seconda per i segnali in ingresso del segnale di Echo. Ho previsto una sola uscita al pin 1 (GPIO_0[0], PIN _A13) per la GPIO_0 e per i segnali di ingesso sulla GPIO_1 il pin 1 (GPIO_1[0], PIN _H12). Sempre grazie alla

GPIO ho potuto sfruttare il pin 11 per l'alimentazione del sensore a 5V e il pin 12 per la massa GND del sensore.

TEST.

1. Connettere opportunatamente il sensore ai pin:

Sensore	PIN GPIO_0	PIN GPIO_1
5V	11	
TRIG	1	
ECHO		1
GND	12	

2. Porre alto lo switch 0 in modo da abilitare il segnale di trigger
3. download su FPGA del progetto "distanza".
4. Programmare il processore attraverso "Altera Monitor Program"
5. Avviare il sistema.

Da questo momento il sistema misura e visualizza la misurazione sul display.

Sono state previste per il test 5 misure, rispettivamente a: 10cm, 15cm, 20cm, 23cm e 25cm; a queste è stato posto una barriera perpendicolare al piano in modo che l'ultrasuono possa colpirlo per effettuare la misura.

NOTA: il sensore è molto sensibile e accurato bastano pochi millimetri, la non perfetta perpendicolarità della barriera o il fatto che la barriera non sia perfettamente parallela al sensore per causare una misura poco stabile e di qualche unità discordante.

Il risultato del test è il seguente:

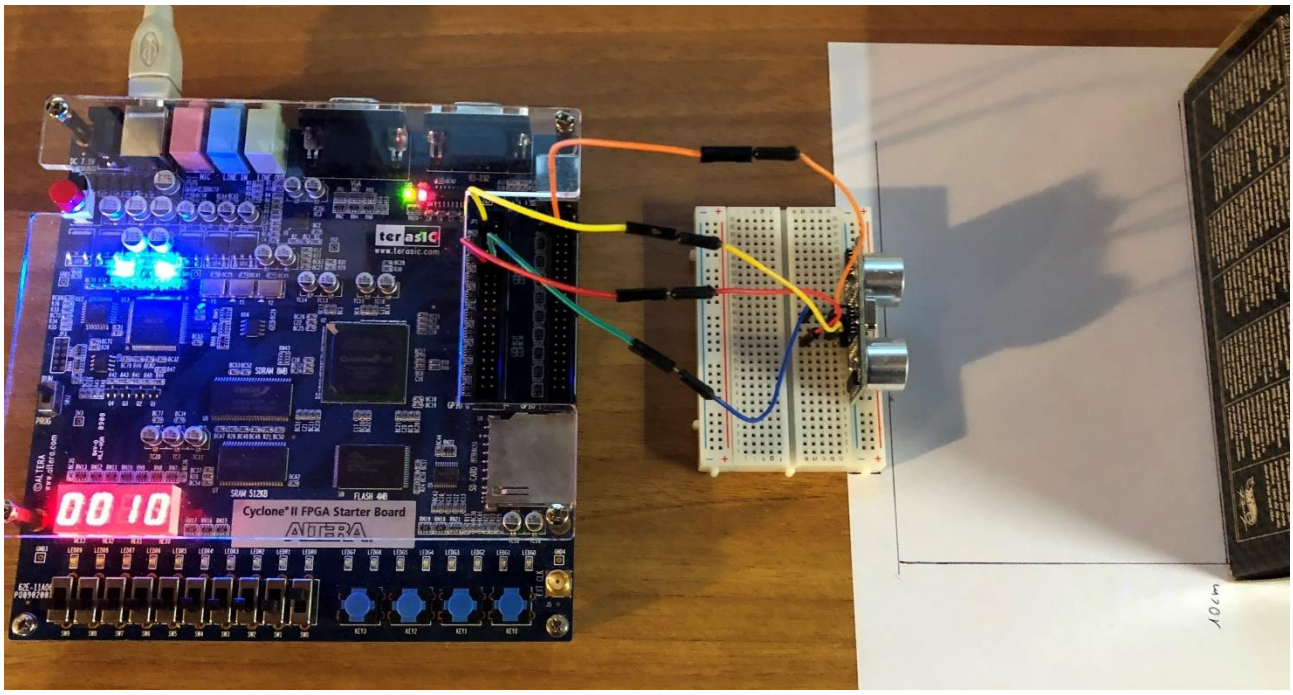

Figura 12: misura a 10 cm.

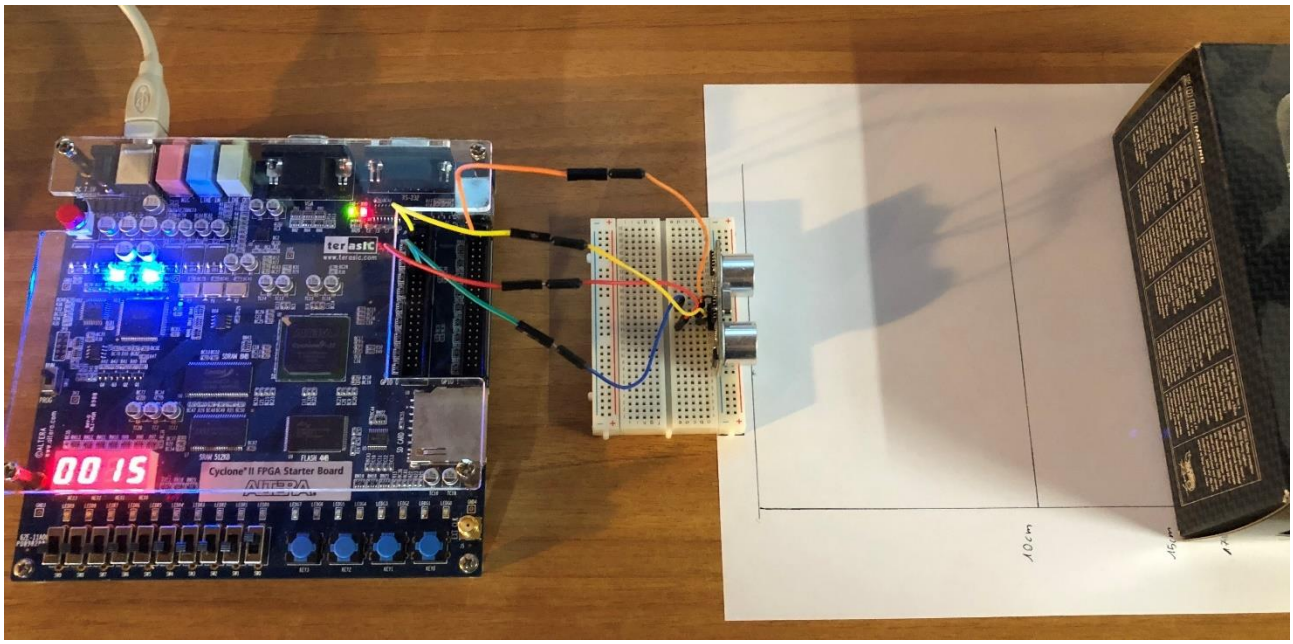

Figura 13: misura a 15 cm.

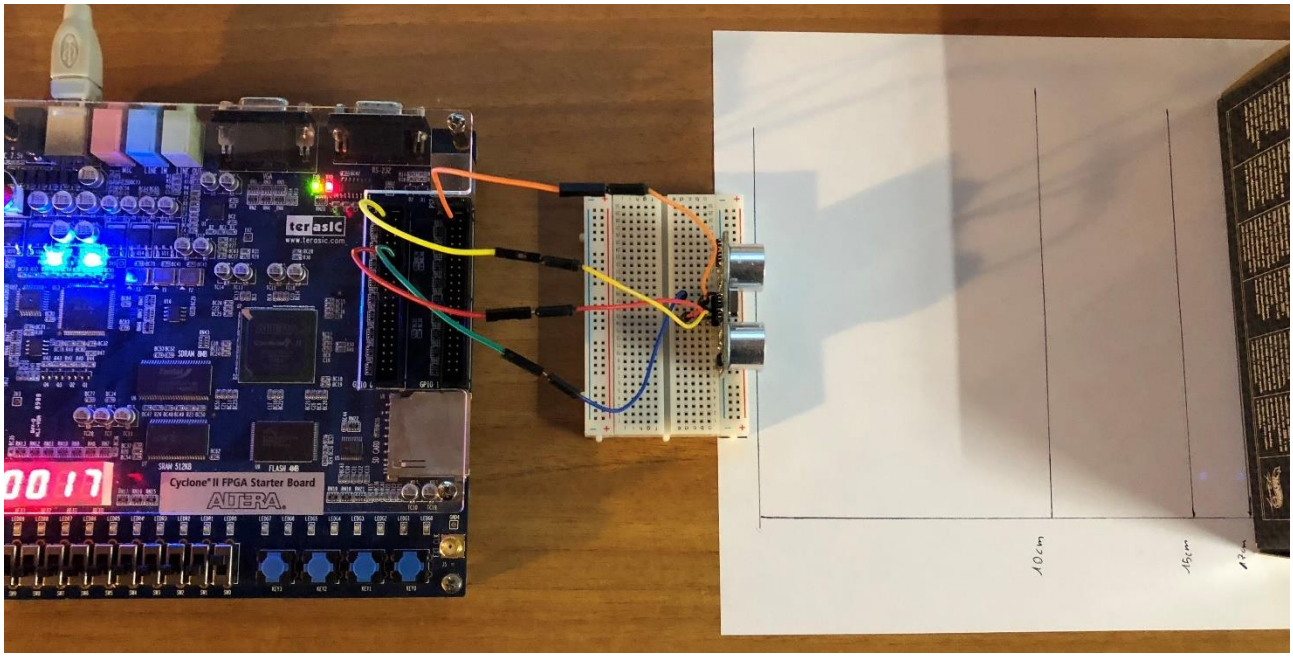

Figura 14: misura a 17 cm.

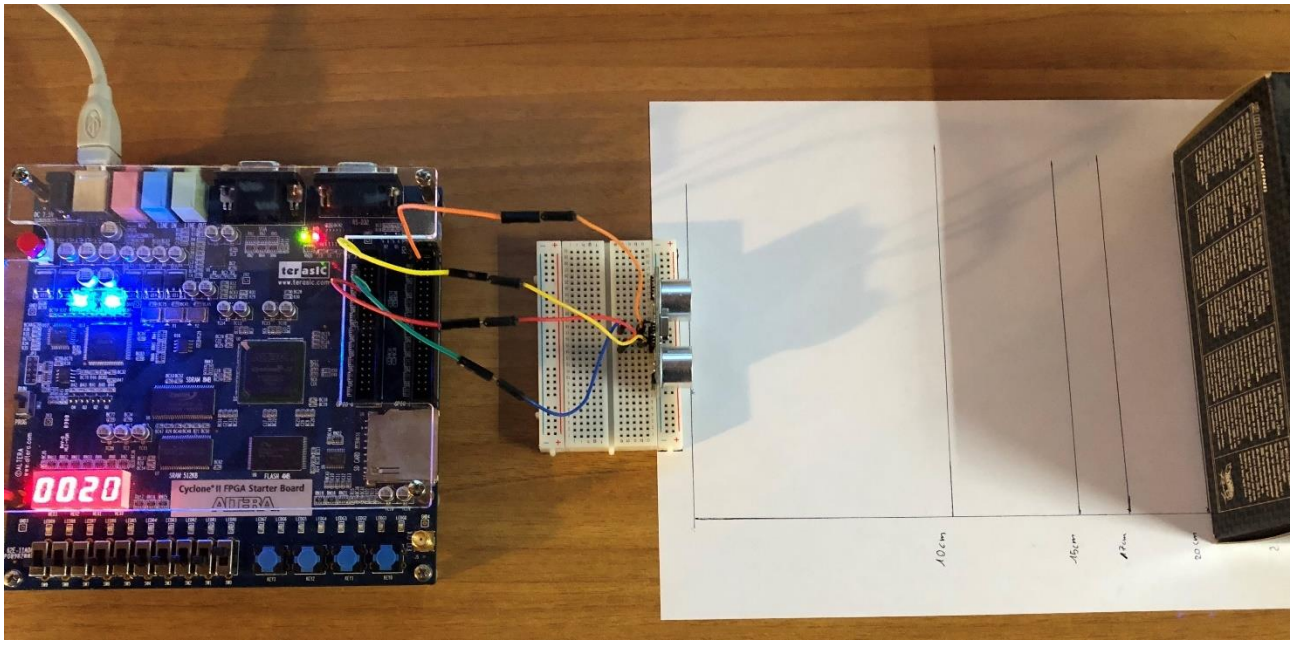

Figura 15: misura a 20 cm.

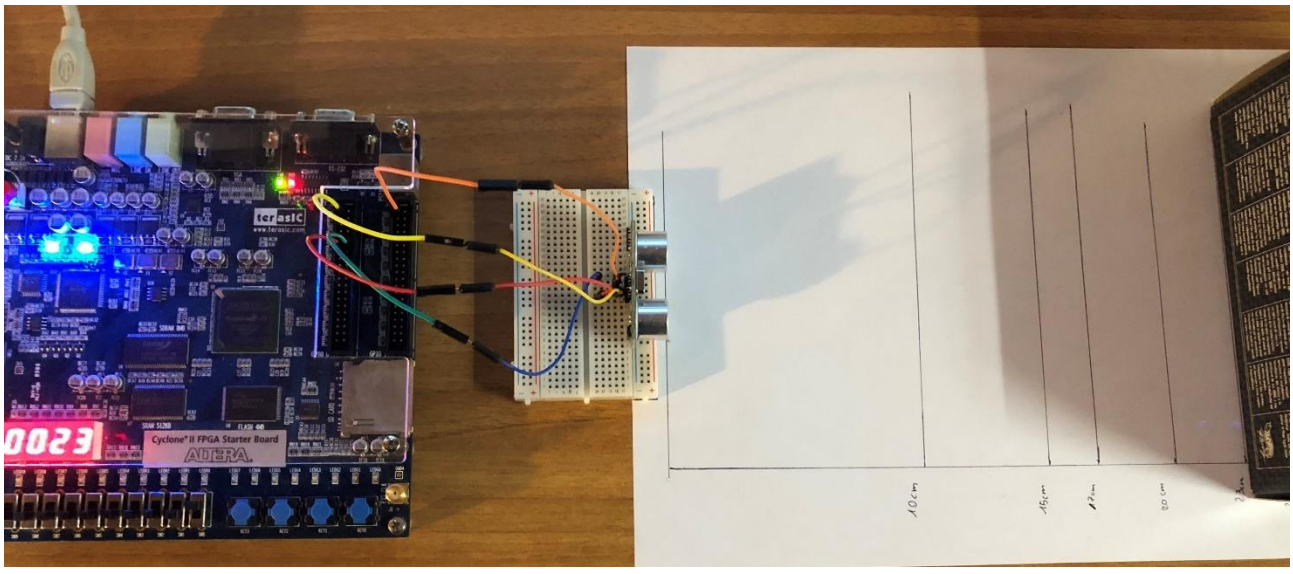

Figura 16: misura a 23 cm.

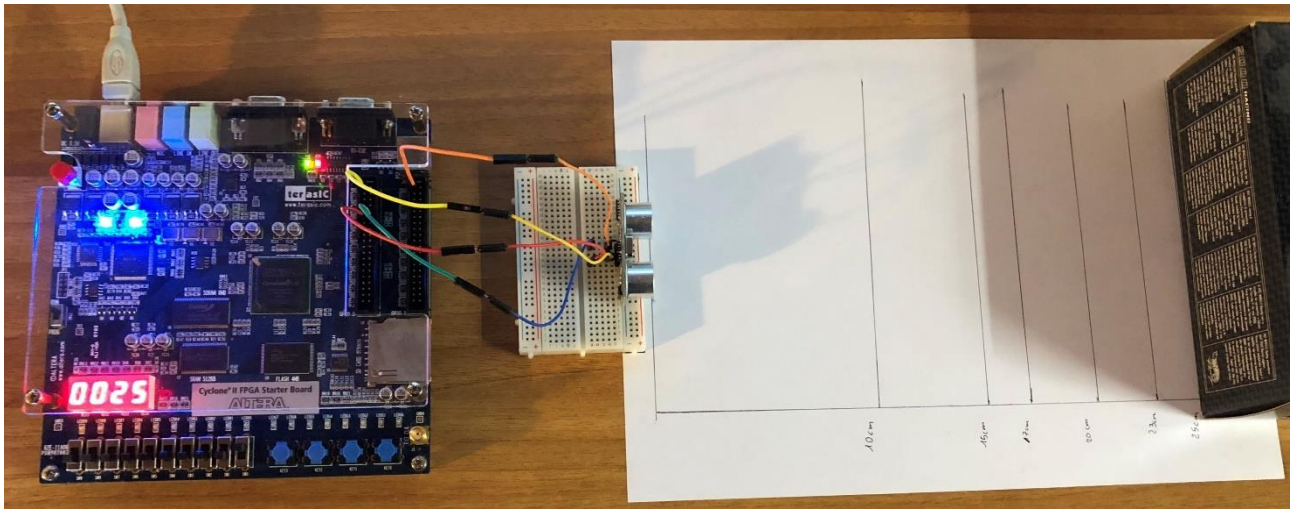

Figura 17: misura a 25 cm.

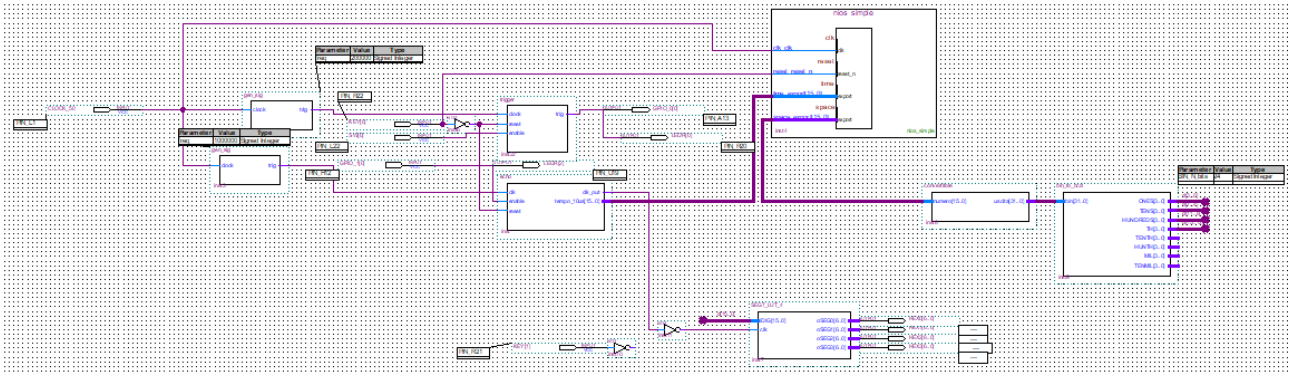

Figura 18: schematico completo del progetto.

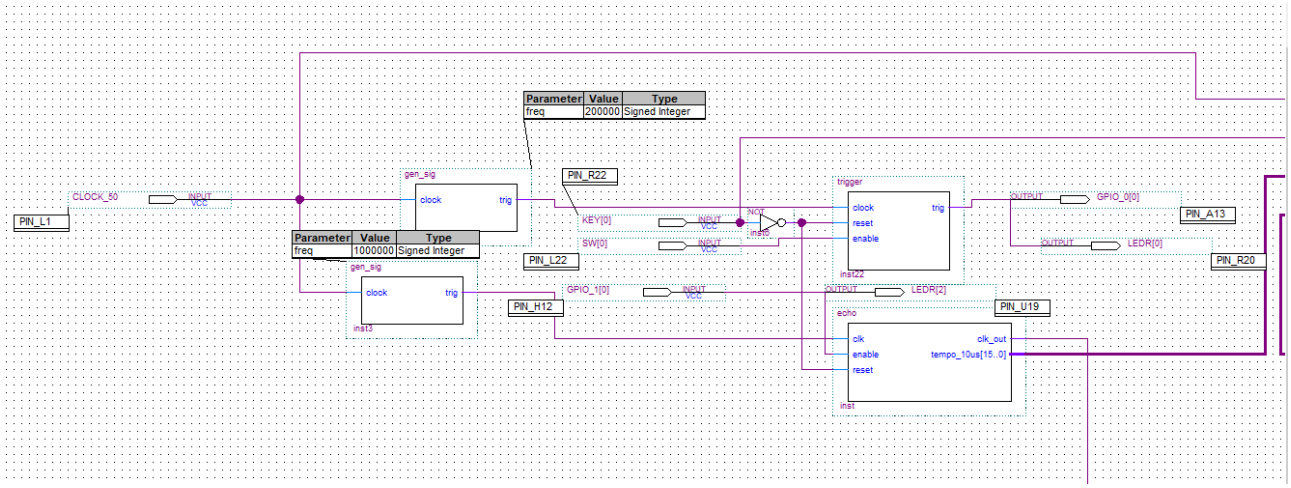

Figura 19: particolare dello schematico in cui si vedono i blocchi demandati alla generazione del segnale di Trigger, alla ricezione del segnale di Echo e al contatore del tempo trascorso all'invio del segnale di Trigger.

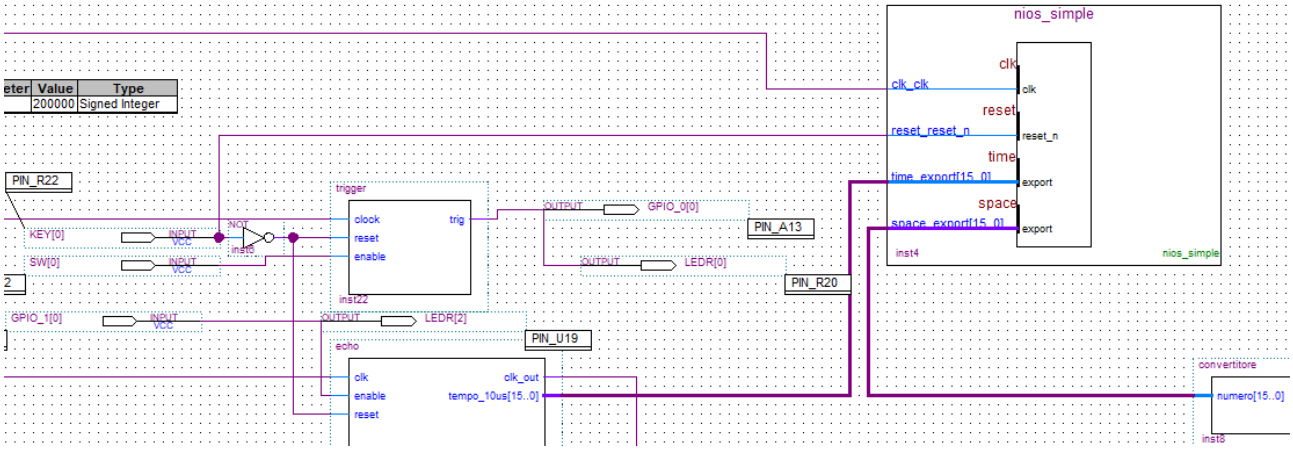

Figura 20: particolare dello schematico in cui si vede il modulo del processore NIOS II e di come si interfaccia al resto dei moduli.

Un nuovo programma in C

L'uso del processore permette di far si che lui si occupi del calcolo dello spazio e che gestisca l'input per il display. Il semplice programma in C si occupa solo della lettura dell'indirizzo di memoria in RAM del processore e di riscrivere in un altro indirizzo il dato letto facendo la divisione per 58; poi può essere prelevato dall'interfaccia parallela I/O dell'output. Per mostrare in modo più rilevante l'intervento del processore in questo progetto, sono andato a modificare il processore prevedendo altre due interfacce parallele I/O di output, una da 10 bit per comandare i led rossi dell'FPGA e una da 8 bit per comandare i led verdi dell'FPGA. L'idea è fare in modo che i led rossi indichino progressivamente l'avvicinamento di un oggetto al sensore, spegnendosi man mano che l'oggetto si avvicini. Sullo schema sono stati semplicemente aggiunte le uscite ai led LEDR[9..0] e LEDG[7..0]. Dopo questa semplice modifica circuitale il vero lavoro di gestione degli eventi è demandato al processore e quindi ad un opportuno programma in C.

```
#define time (volatile int *) 0x0009010
#define space (int *) 0x0009000
#define progressione (int *) 0x0009040
#define impatto (int *) 0x0009050
#include <math.h>
#include <stdio.h>
        int main()
        { int a;
    *impatto = 0;
        while (1)
                {a = *time/58; *space = a;
                        if(a >= 10){
                        *impatto = 255;}
                        else{*impatto = 0;}
                        if(a >= 28){
                        *progressione = 1023;}
                        else if(a >= 26){
                        *progressione = 511;}
                        else if(a >= 24){
                        *progressione = 255;}
                        else if(a >= 22){
                        *progressione = 127;}
                        else if(a >= 20){
                        *progressione = 63;}
                        else if(a >= 18){
                        *progressione = 31;}
                        else if(a >= 16){
                        *progressione = 15;}
                        else if(a >= 14){
                        *progressione = 7;}
                        else if(a >= 12){
                        *progressione = 3;}
                        else if(a >= 10){
                        *progressione = 1;}
                        else{*progressione = 0;}}}
```

Il programma in C sopra riportato aggiunge i due nuovi indirizzi di memoria “progressione” e “impatto” e va a gestire le casistiche per ottenere lo spegnimento in progressione dei Led Rossi.

Distanza dall’oggetto impattante	Valore Progressione	Led Rossi Accesi
10 cm	1	1
12 cm	3	11
14 cm	7	111
16 cm	15	1111
18 cm	31	11111
20 cm	63	111111
22 cm	127	1111111
24 cm	255	11111111
26 cm	511	111111111
28 cm	1023	1111111111

Conclusione

In conclusione il sistema genera in modo indipendente dal processore tutti i segnali per il funzionamento del sensore, calcola il tempo che il segnale impiega per tornare al sensore, salva il dato nella RAM del processore e controlla la visualizzazione sul display. Il processore invece si occupa di leggere i dati in memoria e di effettuare il calcolo della distanza. Questo progetto mi ha permesso di cimentarmi nell’implementazione dei vari blocchi per il controllo del sensore e il relativo conteggi del tempo impiegato per il ritorno, generando e acquisendo i segnali dall’esterno dell’FPGA. Ho potuto cimentarmi nella realizzazione di un processore NIOS, progettarlo e implementarlo sull’FPGA, esonerandolo da compiti ripetitivi come la comunicazione con il sensore e demandandolo al solo compito di lettura/scrittura su memoria e calcolo della distanza sui dati in memoria. Ho affrontato la difficoltà di generare i segnali opportuni sia per il pilotaggio del sensore che per il calcolo del tempo, le difficoltà inerenti alla realizzazione di un processore ad hoc per il mio progetto gestendomi le risorse disponibili senza sovradimensionarle. Concludendo l’esperienza mi ha permesso di realizzare un sistema completo e versatile, e modificando il programma in C è possibile aggiungere o modificare il progetto senza doversi occupare della gestione del sensore come si può evincere in modo ancora più rilevante dalla modifica del programma in C per prevedere una forma di avviso al progressivo avvicinamento di un oggetto al sensore fino ad impattarvici.

Software Utilizzato

Il software utilizzato è Quartus II versione 13.0 e Altera Monitor Program per la versione di Quartus 13.0.

Bibliografia

[1] Altera Corporation: "**DE1 Development and Education Board - User Manual**" *https://www.terasic.com.tw/attachment/archive/83/DE1_UserManual_v1018.pdf*

[2] TUTORIAL FPGA n.2, Marsi Stefano, Università degli Studi di Trieste

[3] TUTORIAL FPGA n.4, Marsi Stefano, Università degli Studi di Trieste

[4] Datasheet Sensore HR-SR04: "**Ultrasonic Ranging Module HC - SR04**" http://www.micropik.com/PDF/HCSR04.pdf