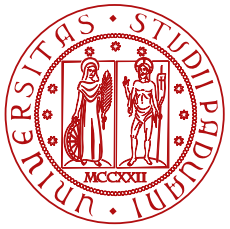


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Laurea triennale in Ingegneria Elettronica

Generatore di forme d'onda realizzato con FPGA

Candidato

Luca Bortolini

Matricola 2000121

Relatore

Prof. Vogrig Daniele

Università di Padova

Anno Accademico

2022/2023

Alla mia famiglia

Quello che per un uomo è "magia",
per un altro è ingegneria.

– Robert Anson Heinlein

Sommario

In questa tesi viene enunciata la realizzazione di un generatore di funzioni e di forme d'onda arbitrarie usando la tecnica della Sintesi Digitale Diretta (DDS). É stato progettato per essere in grado di generare quattro tipi di funzioni: triangolare, quadra, sinusoidale e arbitraria; la forma d'onda arbitraria precaricata è un seno modulato in seno, ma può essere modificata grazie alla comunicazione seriale. Con quest'ultima si ha, inoltre, la possibilità di impostare la frequenza desiderata da visualizzare in uscita e il duty cycle dell'onda quadra. Per quanto riguarda, invece, la selezione della funzione da generare si esegue attraverso gli switch presenti sulla board.

Il progetto è stato realizzato con il linguaggio VHDL grazie all'utilizzo del software Vivado e all'FPGA presente nella scheda Nexys 4 DDR marchiata Digilent.

Per l'implementazione fisica del progetto, oltre all'uso della scheda, è stato progettato un Convertitore Digitale - Analogico a resistenze pesate per fornire la possibilità all'utente finale di poter utilizzare le forme d'onda prescelte.

Alla base del progetto c'è la Sintesi Digitale Diretta, una tecnica molto utilizzata, negli ultimi anni, nella progettazione e realizzazione dei generatori di funzioni. Grazie a questo metodo è possibile variare quasi istantaneamente la frequenza della forma d'onda desiderata, il suo funzionamento specifico verrà spiegato nel primo capitolo di questa tesi.

Indice

Elenco delle figure

Capitolo 1

Sintesi Digitale Diretta

La Sintesi Digitale Diretta (DDS) è una tecnica che pone le sue fondamenta sul clock di sistema della board utilizzata e permette un'alta risoluzione per una vasta gamma di frequenze e una commutazione di frequenza molto rapida.

Al giorno d'oggi il costo competitivo, le alte prestazioni e un packaging piccolo consentono ai prodotti DDS di essere una valida alternativa alle tradizionali soluzioni di sintetizzatori analogici. L'integrazione di un convertitore D/A e un'architettura DDS molto veloci all'interno di un singolo chip consentono a questa tecnologia la possibilità di raggiungere una vasta area di applicazioni e, in molti casi, di sostituirsi ai sintetizzatori analogici basati su PLL.

Un primo punto di forza della Sintesi Digitale Diretta è l'utilizzo di metodi digitali per generare l'uscita desiderata, ciò permette di poter sfruttare l'elasticità della programmazione digitale e avere dei costi contenuti; inoltre, un'architettura digitale, elimina il deterioramento dei componenti dovuto dal passare del tempo o da variazioni di temperatura.

La sua struttura è molto semplice: può essere implementato da soli tre elementi: un accumulatore di fase, un convertitore fase-ampiezza e un convertitore D/A.

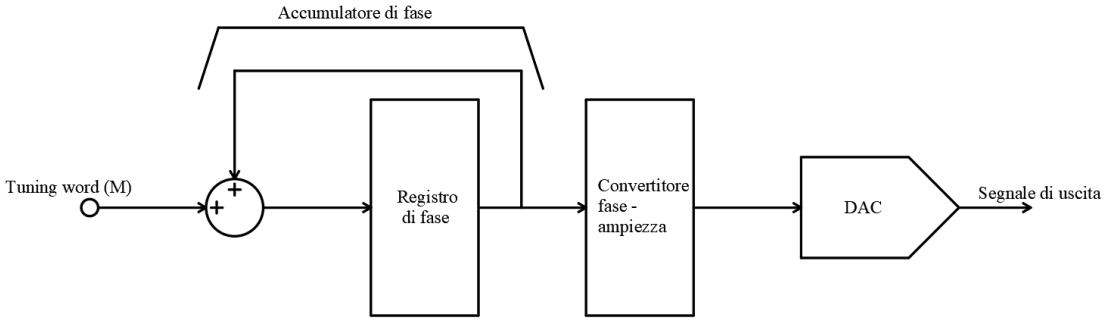


Figura 1.1: Struttura generale dell'architettura della Sintesi Digitale Diretta

1.1 Accumulatore di fase

Il funzionamento di questo metodo è basato sulla conoscenza della fase istantanea della funzione da generare in uscita. Tale valore viene memorizzato in un registro, che nel presente progetto, assume il nome: *Phase_reg*.

La funzione di questo primo modulo è incrementare linearmente il valore della fase all'interno del rispettivo registro. Il passo con cui si deve incrementare la fase prende il nome di *tuning_word* e viene fornito dal "controller" di sistema, il quale a seconda della frequenza impostata dall'utente, calcola il valore della parola binaria (M) da inviare all'accumulatore.

Il registro dell'accumulatore solitamente ha una lunghezza variabile tra 24 e 48 bit. Per questo progetto è stata scelta una lunghezza per la *tuning_word*, e quindi, per il registro di fase, pari a 30 bit. Con questa quantità di bit si ottengono 1 073 741 824 valori possibili per la fase in radianti compresi tra 0 e 2π, di conseguenza la risoluzione della fase è pari a $5.85167 * 10^{-9}$ radianti.

Tale valore ricevuto dall'accumulatore, ad ogni fronte di salita del clock di sistema, viene sommato al valore precente sul registro di fase. Questo valore appena calcolato rappresenta la fase istantanea della funzione di uscita desiderata, da questo si riesce a comprendere uno dei grandi punti di forza di questa tecnica in quanto il ritardo di variazione della frequenza può essere al massimo pari ad un ciclo di clock del sistema (che in questo progetto equivale a 10 μs).

Come per la fase rappresentata in radianti, anche in questo caso i valori della fase sono periodici, il valore massimo, e di conseguenza periodo di ripetizione varia a seconda della lunghezza in bit che assume la fase; come è stato detto sopra, nel nostro caso, il periodo di ripezione della fase sarà di 1 073 741 824 unità. Per ottenerne una rappresentazione grafica è possibile inserire tutti i valori possibili in una circonferenza, così facendo la *tuning_word* può assumere un ulteriore significato, ovvero rappresentare il numero di punti di fase da "saltare" ad ogni aggiornamento della fase istantanea. Per comprendere meglio questo ragionamento fondamentale viene di seguito riportato un esempio semplificato.

Come è raffigurato in Figura 1.2 consideriamo di avere il registro dell'accumulatore con una lunghezza pari a quattro bit, quindi è possibile rappresentare 16 valori diversi di fase istantanea. Ipotiziamo ora che la *tuning_word* calcolata sia pari a uno, allora si avrà che ad ogni periodo del clock il conteggio rappresentante la fase verrà incrementato di un'unità e di conseguenza verranno visualizzati tutti i valori possibili di fase. Per facilità di comprensione a destra della "ruota di fase" viene riportata l'uscita analogica che si avrebbe ipotizzando di aver selezionato la forma d'onda sinusoidale. Se invece viene variata la frequenza desiderata e la parola programmabile che arriva all'accumulatore assume il valore due, ad ogni periodo di clock del sistema ci sara un incremento di

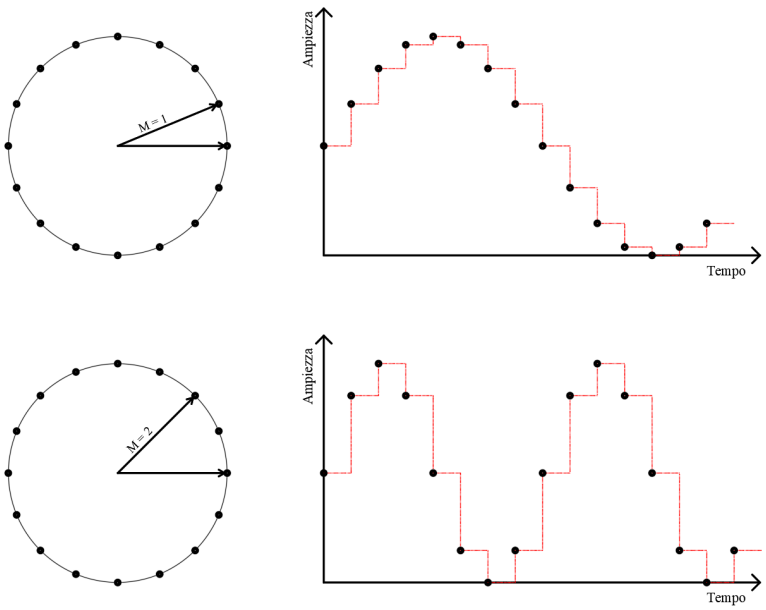


Figura 1.2: Due ruote di fase con *tuning_word* differente

due unità sul registro di fase e verranno quindi saltati alcuni valori di fase come si può facilmente notare dal grafico raffigurante l'uscita.

In realtà il progetto realizzato ha una leggera variazione riguardo questa tecnica, è stato fatto un ragionamento in fase di progettazione per migliorare la qualità della forma d'onda in uscita. Conoscendo il numero di campioni per l'onda sinusoidale che sono salvati in memoria (questo verrà spiegato meglio nella prossima sezione) si è programmato in modo tale che fino ad una frequenza di circa 100 kHz siano mandati in uscita tutti i valori salvati per rappresentare l'onda sinusoidale. Questo è stato fatto per migliorare la risoluzione dell'onda sinusoidale ed evitare di avere salti troppo bruschi di tensione o rischiare di avere troppi pochi valori per poter raffigurare un'onda sinusoidale. L'unico aspetto negativo che questa scelta può portare è che a frequenze molto inferiori di quella limite, i valori dell'uscita verranno mantenuti tali per più cicli di clock.

Fin'ora si è sempre parlato di *tuning_word*, ora vediamo nello specifico quali passaggi sono necessari per calcolarne il valore.

In linguaggio matematico una sinusoide è rappresentata come: $x(t) = sin(wt)$

L'argomento del seno dipende linearmente dal tempo, inoltre la pulsazione w dipende linearmente dalla frequenza secondo la relazione $w = 2\pi f$

Considerando un intervallo di tempo δt la varaizione di fase è pari a $\Delta fase = w\delta t$,

da cui si ricava

$$w = \frac{\Delta fase}{\delta t} = 2\pi f \Rightarrow f = \frac{\Delta fase}{2\pi\delta t} \tag{1.1}$$

L'intervallo di tempo preso in considerazione è un periodo di clock, quindi $\delta t = T_{clk} = \frac{1}{f_{clk}}$

La variazione di fase $\Delta fase$ non è altro che la *tuning_word* M e un periodo della sinusoide al posto di essere espresso in radianti viene espresso in numero di punti presenti nella ruota di fase, ovvero $2\pi = 2^N$.

Infine si ottiene:

$$f_{out} = \frac{M * f_{clk}}{2^N} \tag{1.2}$$

Per calcolare M, quindi, è sufficiente scambiare alcuni operatori e risulta:

$$M = \frac{2^N * f_{out}}{f_{clk}} \tag{1.3}$$

Per ottenere la *tuning_word*, quindi, i calcoli da eseguire sono molto semplici, in particolare nella formula di cui sopra:

- $\mathbf{f_{out}} \longrightarrow$ frequenza di uscita
- $\mathbf{M} \longrightarrow$ valore della parola programmabile (*tuning_word*)
- $\mathbf{f_{clk}} \longrightarrow$ frequenza di clock del sistema (nel nostro caso è pari a 100 MHz)
- $\mathbf{N} \longrightarrow$ numero di bit del registro di fase

Grazie alla formula appena calcolata è possibile ricavare la risoluzione in frequenza del progetto, che corrisponde anche alla minima frequenza erogabile, dato che si impone il salto di fase più piccolo possibile:

$$f_{out} = \frac{M * f_{clk}}{2^N} = \frac{1 * 10^8}{2^{30}} = 0.0931322\,Hz \tag{1.4}$$

Considerato che per rappresentare la fase vengono utilizzati 30 bit, ad ogni valore di fase dovrebbe essere associato un valore di ampiezza, questo comporterebbe la realizzazione di una ROM dalle dimensioni molto elevate. Solitamente, quindi, si "tronca" la fase a un numero inferiore di bit; così facendo si riducono le dimensioni della ROM ma si riesce a mantenere un'elevata risoluzione in frequenza.

1.2 Convertitore Fase - Ampiezza

Come si può facilmente intuire dal nome di questo modulo, ora ci occupiamo della traduzione dalla fase all'ampiezza che andremo effettivamente ad ottenere in uscita dal nostro generatore.

Un paramentro importante per questo modulo è il numero di bit delle parole considerate. Come è stato enunciato appena sopra è impossibile pensare di poter tenere tutti i 30 bit dell'accumulatore di fase, infatti valori tipici del numero di bit per un convertitore fase - ampiezza variano tra i 10 e i 18 bit, in questo caso è stato scelto di impostare il convertitore a 12 bit.

All'interno di questo modulo è presente una memoria ROM, quindi solo leggibile, nella quale in fase di programmazione vengono salvati i valori rappresentanti un solo quarto dell'onda sinusoidale, in particolare vengono salvati i valori del primo quarto di periodo, ovvero la parte ascendente a valori positivi. Viene fatto questo per poter risparmiare in termini di memoria occupata nella FPGA, poichè grazie a semplici operazioni matematiche è possibile ottenere tutta la funzione sinusoidale partendo solamente dai dati sopra citati.

Questo elemento, a seconda della funzione da visualizzare in uscita, utilizza o meno i valori presenti nella memoria ROM, in quanto, come è rappresentato in Figura 1.3, se si desidera visualizzare la forma d'onda triangolare non vengono letti i valori presenti nella memoria, ma si porterà in uscita direttamente il valore della fase istantanea.

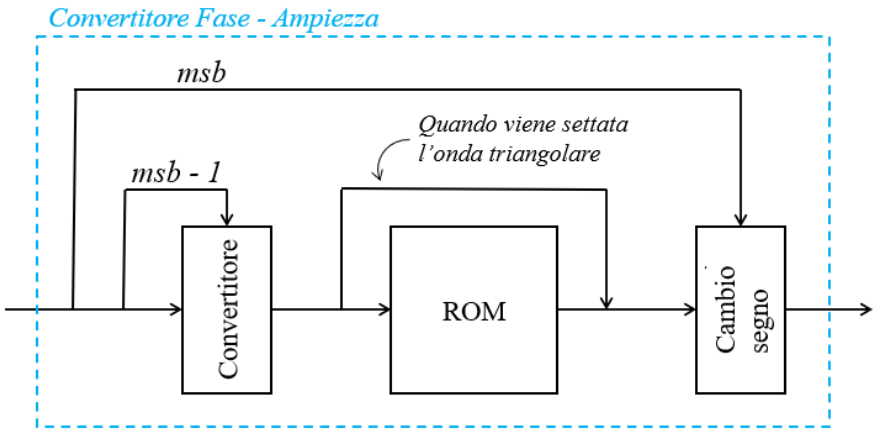


Figura 1.3: Schema interno del Convertitore Fase - Ampiezza

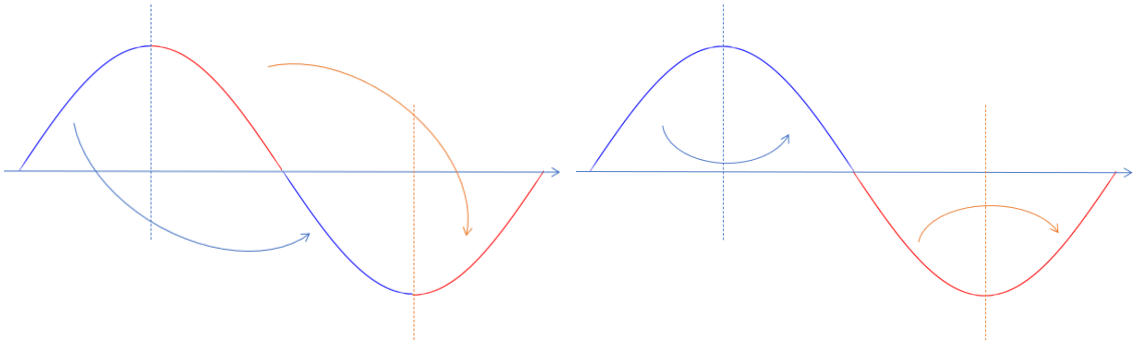
Vediamo ora più nel dettaglio come si riesce a ottenere la forma d'onda sinusoidale nella sua interezza partendo da una sua frazione.

É stato appena detto che dei 30 bit per ogni parola dell'accumulatore di fase il convertitore ne utilizza solamente 12, di quest'ultimi si ha un ulteriore frazionamento, i primi due infatti (partendo dai più significativi) vengono utilizzati nella costruzione dell'onda e solamente i rimanenti 10 vengono utilizzati come indirizzo per la memoria per poter ottenere così l'ampiezza dell'onda sinusoidale. Il bit più significativo rappresenta quando si deve specchiare i valori letti dalla ROM rispetto all'asse x e quindi quando si vuole che l'uscita sia negativa, mentre il bit alla sua destra rappresenta quando si deve specchiare i valori letti rispetto all'asse delle y e quindi ottenere il secondo e l'ultimo quarto di periodo.

Per specchiare i valori delle ampiezze si utilizzano delle formule matematiche davvero molto semplici:

- Specchiare i valori rispetto all'asse X, Figura 1.4a

```
1 rom_in <= t_phase;
2 AMPLITUDE <=  "10000000000" - rom_out;
```



(a) Rotazione rispetto all'asse X (b) Rotazione rispetto all'asse Y

Figura 1.4: Come vengono specchiate le varie frazioni di sinusoide

rom_in è l'ingresso della memoria ROM, ovvero l'indirizzo al quale sarà presente il valore dell'ampiezza che dovrà assumere l'uscita.

t_phase è la parte della fase istantanea che viene usata come indirizzo, ovvero 10 bit che partono dal terzo più significativo.

Con l'operazione sopra riportata si riesce a rovesciare la forma d'onda rispetto a X perchè si pone come valore di offset il valore che si trova a metà dell'uscita finale e si va a togliere il valore che assumerebbe l'uscita se fosse nella parte positiva, così facendo si riesce a ottenere la semionda negativa della sinusoide.

- Specchiare i valori rispetto all'asse Y, Figura 1.4b

```
1 t_phase := std_logic_vector("1111111111" - PHASE(11 downto 2));
```

Per rovesciare la sinusoide rispetto all'asse Y, e ottenere quindi la discesa nella semionda positiva e l'ascesa nella semionda negativa, si modifica l'indirizzo che sarà usato come ingresso dalla memoria ROM, con questa sottrazione si va a leggere la memoria in ordine inverso, ad esempio quando PHASE vale "000..00" l'indirizzo che effettivamente viene mandato alla memoria ROM è "111..11" e quindi è il valore massimo salvato che rappresenta l'apice della sinusoide.

La memoria ROM utilizzata in questo progetto dispone di 1 024 locazioni di memoria, di conseguenza utilizza indirizzi con una lunghezza pari a 10 bit. La scelta di salvare nella ROM solamente un quarto di sinusoide ha permesso di ridurre di quattro volte la dimensione, se non fosse stata fatta questa scelta la ROM avrebbe dovuto presentare 4 096 celle di memoria.

I dati salvati all'interno della memoria sono stati ottenuti grazie ad un semplice Script realizzato in Matlab, il codice leggeva ogni punto della sinusoide da salvare e lo scriveva in un file già in formato binario a 11 bit. Successivamente il file *memory.txt* è stato inserito come file di configurazione in Vivado.

Questo modulo, oltre alla generazione della forma d'onda sinusoidale è in grado di costruire anche la forma d'onda triangolare. Per generare questo secondo tipo di forma d'onda il procedimento è molto più semplice, i dati infatti non devono passare attravero la ROM, ma la fase istantanea che arriva in ingresso viene inviata in uscita a seguito di alcune modifiche.

Anche in questo caso, dei 30 bit che utilizza l'accumulatore di fase ne vengono considerati solamente 12.

Si riscontrano molte somiglianze con la costruzione dell'onda sinusoidale, i due bit più significativi infatti vengono utilizzati dall'FPGA per la realizzazione: il bit più significativo indica quando si deve specchiare il segnale rispetto all'asse X, mentre il secondo bit più significativo indica quando il segnale va specchiato rispetto all'asse Y.

I vari passaggi per il ribaltamento del segnale nelle due direzioni vengono riportate qui di seguito:

- Specchiare i valori rispetto all'asse X

```
AMPLITUDE <= "10000000000" - t_phase;
```

  AMPLITUDE è l'ampiezza del segnale in uscita dal generatore e *t_phase* in questo caso è esattamente la fase istantanea che arriva in ingresso al modulo, ovviamente troncata alla lunghezza necessaria.

  Con questa semplice sottrazione si riesce a ottenere la parte a valori negativi della forma d'onda da generare. Come si può facilmente osservare, si ha una notevole somiglianza con la generazione dell'onda sinusoidale, l'unica differenza è che non viene utilizzata la ROM.

- Specchiare i valori rispetto all'asse Y

```
t_phase := std_logic_vector("1111111111" - PHASE(11 downto 2));
```

  Per specchiare la forma d'onda rispetto all'asse Y si utilizza lo stesso comando già visto durante la generazione della sinusoide, ovvero si va a modificare la fase istantanea ricevuta in ingresso. In questo caso *t_phase* non viene utilizzato come indirizzo per la memoria ROM, il principio di funzionamento rimane del tutto analogo a quello visto in precedenza.

1.3 Simulazione

Una volta programmati i moduli essenziali per realizzare la Sintesi Digitale Diretta si può passare alla simulazione dell'intero blocco. Per ottenere una simulazione più vicina possibile alla realtà è stata eseguita una simulazione *Post-Synthesis*, ovvero prima di eseguire la simulazione vengono sintetizzati tutti i blocchi così da ottenere quello che in un secondo momento verrà implementato all'interno dell'FPGA.

Per poter osservare il comportamento del blocco è stato scritto un file di *testbench*, ovvero un file che simula gli ingressi che riceverà il blocco una volta che verrà implementato nell'FPGA per poter così osservare se le relative uscite corrispondono a quelle desiderate.

Dato che questo modulo è in grado di generare forme d'onda sinusoidali e triangolari, sono state simulate ambedue a diverse frequenze, per poter osservare la precisione nella generazione.

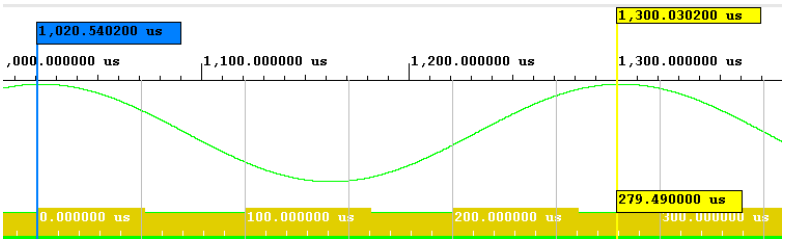


Figura 1.5: Simulazione sinusoide a 3 578 kHz

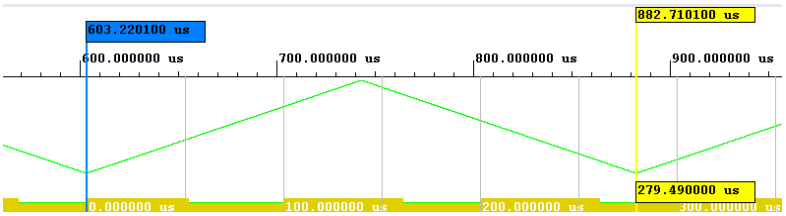


Figura 1.6: Simulazione forma d'onda triangolare a 3 578 Hz

Vengono analizzati ora i risultati rappresentati in Figura 1.5.

In questo primo caso è stata generata una forma d'onda sinusoidale con una frequenza prescelta di 3 578 kHz; dunque, in teoria, la forma d'onda in uscita dovrebbe avere un periodo pari a circa 279.486 μs. Da quanto si osserva grazie ai *marker* (cursori) inseriti nella simulazione di Vivado il periodo ottenuto risulta 279.48μs. Si ha quindi:

$$T_{teorico} = \frac{1}{f} = \frac{1}{3578} \simeq 279.486\,\mu s$$
$$T_{mis} = 279.48\,\mu s \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |279.48 - 279.486| = 0.006\,\mu s.$$
$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.006*10^{-6}}{279.486*10^{-6}} * 100 = 0.002\%$$

L'errore che si ottiene può essere attribuito all'errore di troncamento e alla sensibilità del clock di sistema, in quanto un periodo di clock dura 10 ns. Si può quindi affermare che la realizzazione ha una risoluzione piuttosto elevata.

Come si nota in Figura 1.6 ora si analizza la generazione della forma d'onda triangolare, la frequenza rimane invariata rispetto al caso precedente e dunque non cambia il periodo teorico. É immediato notare che anche il periodo ottenuto non varia, perciò l'errore percentuale $\epsilon_{\%}$ rimarrà immutato allo 0.002%.

Si analizzano ora altri risultati le cui immagini, per non appesantire troppo la tesi, verranno inserite nell'appendice.

- Frequenza impostata a 88 Hz

  Il periodo misurato è il medesimo per funzione sinusoidale e funzione triangolare, di conseguenza anche gli errori sono gli stessi

$$T_{teorico} = \frac{1}{f} = \frac{1}{88} \simeq 11.37\,ms$$
$$T_{mis} = 11.374\,ms \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |11.374 - 11.37| = 0.004\,ms$$
$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.004*10^{-3}}{11.37*10^{-3}} * 100 = 0.035\%$$

- Frequenza impostata a 23 kHz

  Il periodo misurato è il medesimo per funzione sinusoidale e funzione triangolare, di conseguenza anche gli errori sono gli stessi

$$T_{teorico} = \frac{1}{f} = \frac{1}{23000} \simeq 43.478\,\mu\text{s}$$

$$T_{mis} = 11.374\,m\text{s} \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |43.48 - 43.478| = 0.002\,\mu\text{s}$$

$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.002 * 10^{-6}}{43.478 * 10^{-6}} * 100 = 0.005\%$$

- Frequenza impostata a 100 kHz

  Il periodo misurato è il medesimo per funzione sinusoidale e funzione triangolare, di conseguenza anche gli errori sono gli stessi

$$T_{teorico} = \frac{1}{f} = \frac{1}{100000} \simeq 10\,\mu\text{s}$$

$$T_{mis} = 10\,\mu\text{s} \longrightarrow \Delta t = T_{mis} - T_{teorico} = 0\,\mu\text{s}$$

L'errore percentuale $\epsilon_{\%}$ è nullo, in quanto il periodo teorico e periodo misurato con i cursori si eguagliano.

Capitolo 2

Realizzazione fisica

2.1 Scheda FPGA

Come è stato già anticipato nel corso del capitolo precedente la realizzazione dell'intero progetto, a meno della conversione digitale analogica, avviene su una scheda FPGA (Field Programmable Gate Array). Questa è una scheda programmabile e al giorno d'oggi sono in grande diffusione grazie all'alta efficienza, alla flessibilità di un dispositivo programmabile e ai costi contenuti.

L'FPGA è utilizzato in tutte le applicazioni nelle quali è richiesta una performance elevata, ma non è previsto un numero di prodotti elevato, ad esempio: satelliti, base station per reti cellulari, dispositivi di laboratorio e molto altro.

Le FPGA sono dispositivi basati sulla tecnologia CMOS con memoria di configurazione SRAM. L'architettura è simile a quella dei primi CPLD, solo che si ha un numero maggiore di blocchi logici programmabili semplici, questi sono immersi in una rete di interconnessioni in cui sono presenti degli *switching* nelle interconnessioni come si può osservare in Figura 2.1

L'appellativo "Field Programmable" sta ad indicare che la programmazione avviene "sul campo", ovvero l'utente finale è in grado di programmare il dispositivo, in opposizione a quei dispositivi la cui programmazione avviene

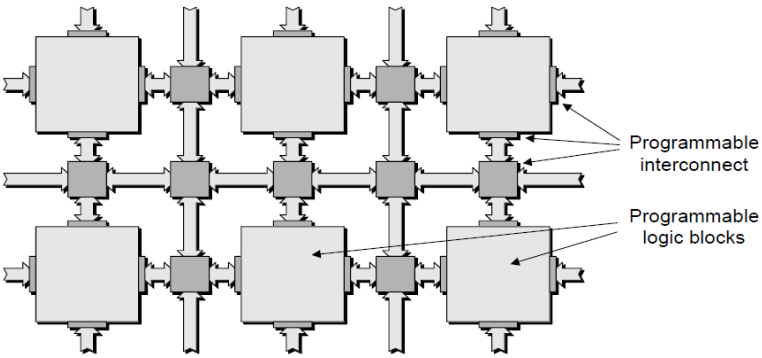


Figura 2.1: Struttura matriciale di una scheda FPGA, fonte: *The Design Warrior's Guide to FPGAs* [6]

dal produttore e l'utente non ha la possibilità di modificarla. In questo modo ogni utilizzatore ha la possibilità di programmarla secondo le proprie necessità e ottimizzarla secondo i propri progetti.

2.1.1 Blocco Logico Programmabile

I blocchi Logici Programmabili sono alla base dell'architettura FPGA e sono connessi tra loro da interconnessioni anch'esse programmabili.

Come si nota in Figura 2.2 una semplice implementazione contiene pochi elementi, si ha: una Look-Up Table (LUT) a tre o cinque ingressi, un multiplexer e un flip-flop. Ogni FPGA contiene un largo numero di questi Blocchi Logici Programmabili, ogni blocco logico può essere configurato per eseguire una differente funzione. Il multiplexer può essere configurato per prendere l'output della LUT o un altro input del blocco logico, inoltre la LUT può essere configurata per rappresentare una qualsiasi funzione logica a tre ingressi.

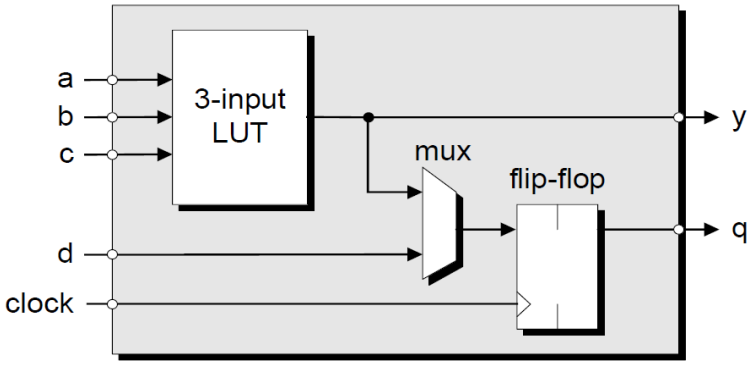


Figura 2.2: Struttura interna di un Blocco Logico Programmabile, fonte: [6]

La Look-Up Table internamente è costituita da 2^n celle SRAM e un multiplexer a n ingressi. Le celle SRAM vegono programmate a seconda della funzione logica che si desidera ottenere. Ad esempio se si vuole ottenere la funzione $y = (a \,\&\, b) \,|\, !c$ si programmeranno le celle con i relativi valori dell'uscita come si nota in Figura 2.3

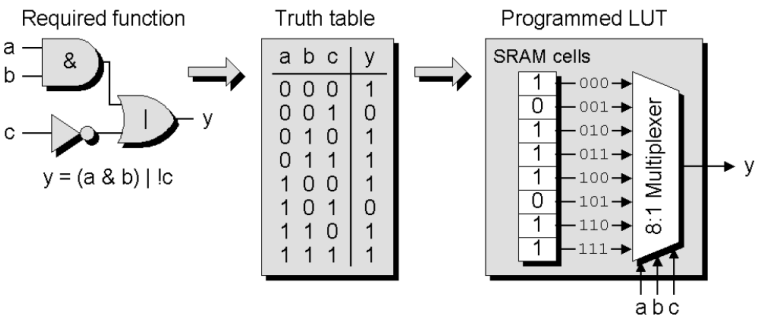


Figura 2.3: Implementazione della funzione $y = (a \,\&\, b) \,|\, !c$ con LUT, fonte: [6]

Le schede FPGA spesso includono anche moduli specifici come: sommatori, moltiplicatori, blocchi MAC, memorie RAM, interfacce I/O veloci.

2.2 Convertitore Digitale - Analogico

Un convertitore digitale-analogico (DAC) è un circuito elettronico in grado di ricevere in ingresso un numero digitale, quindi in formato binario, e resisture un valore analogico corrispondente, quindi un segnale di tensione o di corrente limitato in un range prefissato. Un largo uso domestico dei DAC si ha ad esempio nei riproduttori digitali di suoni, nei controlli di luminosità o volume dei televisori e in tutte quelle situazioni in cui un controllore digitale deve gestire una grandezza di tipo analogico.

Una suddivisione di tali dispositivi viene fatta rispetto a come vengono forniti i valori in ingresso, esistono i convertitori seriali i quali ricevono in ingresso un bit per volta e i convertitori paralleli, i quali ricevono tutti i bit da convertire simultaneamente su più linee di ingresso differenti.

Alcune caratteristiche dei convertitori che assumono una rilevante importanza sono: la *risoluzione*, connessa direttamente al numero di bit di cui dispone il convertitore, risulta banale affermare che maggiore sarà il numero di bit maggiore sarà la risoluzione del convertitore. All'aumentare della risoluzione però aumenta anche il tempo di elaborazione del DAC, quindi più elevata è la risoluzione più lungo sarà il periodo di conversione. Pertanto nella scelta del DAC si deve ottenere un compromesso tra risoluzione e velocità in relazione al campo di applicazione.

In questa tesi viene progettato e utilizzato il secondo modello di convertitore enunciato, in particolare si avranno 11 bit paralleli in ingresso e il segnale di uscita potrà variare tra una tensione minima di - 10 Volt e una tensione massima pari a + 10 Volt.

Esistono molteplici implementazioni di Convertitori Digitale - Analogico, alcune sono: DAC a resistenze pesate, DAC a rete R - 2R, DAC flash, DAC Sigma-Delta le quali si differenziano tra loro per complessità di realizzazione, tempi di conversione e difficoltà di implementazione.

Il DAC scelto per questa tesi è il *DAC a resistenze pesate*, la cui implementazione è piuttosto semplice, ma si perde un po' di risoluzione. Le scelte circuitali e implementative verranno enunciate nei prossimi paragrafi.

2.2.1 Schematico

Il circuito schematico del DAC realizzato è raffigurato in Figura 2.4, come si nota oltre all'amplificatore operazionale in configurazione di sommatore inver-

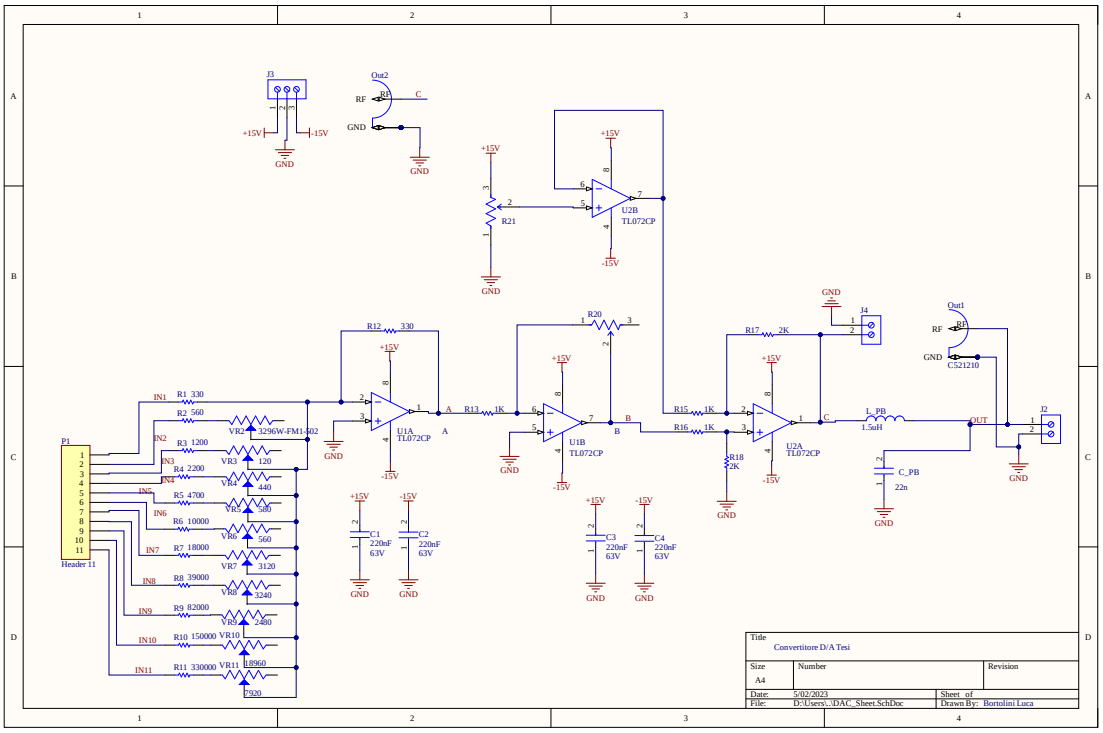


Figura 2.4: Circuito schematico del DAC a resistenze pesate

tente che è alla base del convertitore, sono stati inseriti altri operazionali con lo scopo di condizionare il segnale per renderlo utilizzabile dall'utente finale.

In particolare è stato aggiunto:

- Amplificatore invertente: per ottenere tutti i segnali a valori positivi e ottenere così la forma d'onda desiderata completamente nel piano positivo delle tensioni;
- Amplificatore sottrattore: per togliere la tensione di *offset* presente nella forma d'onda e di conseguenza far in modo che sia centrata rispetto alla tensione nulla;
- Amplificatore buffer: per separare circuitalmente la generazione della tensione di offset dal circuito sottrattore, in questo modo la tensione sul potenziometro non è influenzata dalla tensione sul sottrattore.

In fase di progettazione del convertitore sono stati presi in considerazione molteplici fattori, ad esempio: velocità degli amplificatori operazionali, tensione di uscita della scheda Xilinx, tensione di alimentazione del convertitore, range di tensione di uscita della forma d'onda.

Si entra ora più nel dettaglio della scelta dei componenti e dei calcoli effettuati.

Per quanto riguarda gli amplificatori operazionali, la scelta è stata fatta considerando due fattori principali: la velocità di commutazione e il tipo di alimentazione. Per la velocità di commutazione i limiti vengono imposti dalla

generazione dell'onda quadra, infatti in questo caso l'OPAMP deve passare da una tensione di -10V a una tensione di +10V, la massima frequenza generabile deve avvicinarsi il più possibile ai 100 kHz. Dunque il periodo in uscita risulta pari a 10 μs, e il tempo di transizione deve essere al massimo il 15% del periodo del segnale, altrimenti non si riesce a vedere la forma d'onda quadra. Quindi l'amplificatore deve almeno essere in grado di compiere una transizione con la seguente specifica:

$$\frac{\Delta V}{t} = \frac{20V}{0.15*10\mu s} = 13.3V/\mu s$$

Per quanto riguarda l'alimentazione, invece, il vincolo viene imposto dal range di tensioni che si hanno in uscita; dato che l'uscita finale varia tra i - 10 V e i + 10 V, per avere maggior certezza che gli operazionali lavorino nella zona lineare della transcaratteristica, è stato deciso di avere un'alimentazione duale da - 15 V a + 15 V.

L'OPAMP TL072 ha uno *slew rate* tipico pari a 13 V/μs e un'alimentazione duale compatibile con quella decisa da progetto, risulta quindi essere il componente adatto per la realizzazione.

Si passa ora al dimensionamento delle resistenze della configurazione invertente e differenziale e al settaggio del valore della tensione di offset. Per fare ciò si è utilizzato un sistema di equazioni:

$$\begin{cases} ((-6.6)*(-A) - Off)*B = 10 \\ ((-3.3)*(-A) - Off)*B = 0 \\ ((0)*(-A) - Off)*B = -10 \end{cases}$$

Nel quale si ha:

- A $\longrightarrow$ guadagno di amplificazione della configurazione invertente
- B $\longrightarrow$ guadagno di amplificazione della configurazione differenziale
- Off $\longrightarrow$ tensione di offset da togliere al segnale generato per centrarlo sugli 0V
- il primo valore racchiuso nelle parentesi tonde è la tensione che si ha al punto 1 in Figura 2.4, in particolare si hanno - 6.6 V quando la parola generata dall'FPGA è "111..1", - 3.3 V quando la parola generata è "100..0" (ovvero quello che alla fine del condizionamento diventerà 0 V) e si hanno 0 V quando la parola generata è "000..0".
- il valore a destra dell'uguale è la tensione che si desidera ottenere alla fine della conversione e del condizionamento del segnale.

Risolvendo il sistema si ottengono i seguenti risultati:

$$\begin{cases} A = 1.51 \\ B = 2 \\ Off = 5V \end{cases}$$

Per ottenere questi valori con un'alta precisione si è optato di posizionare un potenziometro rotativo al posto della resistenza di retroazione nella configurazione invertente in modo tale da riuscire ad ottenere con più precisione il fattore 1.51. Anche per quanto riguarda la generazione della tensione di offset si è utilizzato un potenziometro rotativo connesso tra i + 15 V di alimentazione del DAC e il pin di Ground.

Come è stato enunciato, i valori generati dalla scheda e di conseguenza emessi dal DAC vegnono aggiornati ogni periodo di clock, dunque per tutto il periodo di clock tali valori rimangono costanti e successivamente vengono variati di almeno un *passo di quantizzazione*. Questo valore è la minima differenza di tensione che si riesce a ottenere in uscita variando solamente l'LSB in ingresso. Tale passo dipende dal range di tensioni che il DAC genera e dal numero di bit che possiede il convertitore.

Nel caso di questa tesi, il convertitore avrà 11 bit di ingresso e una tensione di uscita che può variare tra i - 10V e i + 10V, quindi la differenza totale è pari a 20V. Si calcola ora il passo di quantizzazione che possiede il DAC in esame:

$$\frac{\Delta V}{2^N} = \frac{20}{2^{11}} = 9.77\, mV/LSB$$

Questo però comporta che il segnale d'uscita non sarà una sinusoide, ma sarà una sinusoide a gradini come è mostrato in Figura 2.5

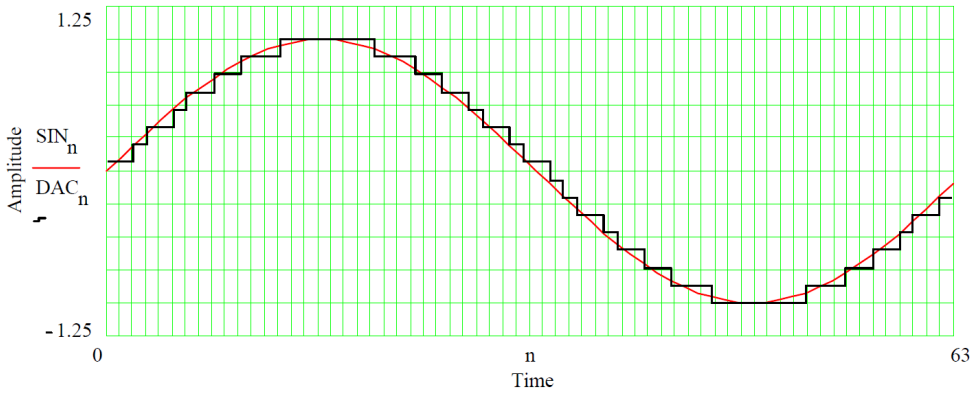


Figura 2.5: Risoluzione di un DAC rispetto a un'onda sinusoidale, fonte: *A Tecnical Tutorial on Digital Signal Synthesis* [2]

Per ottenere in uscita un segnale che assomigli il più possibile a una sinusoide è stato, quindi, inserito un filtro passa - basso in serie al DAC, questo taglia le alte frequenze, colpevoli del cambiamento istantaneo di tensione in uscita permettendo così una transizione più "morbida" tra i vari valori di uscita del DAC.

É stato scelto e progettato un filtro passa-basso del secondo ordine, LC, per due aspetti principali: non attenua il segnale in banda passante e dopo la

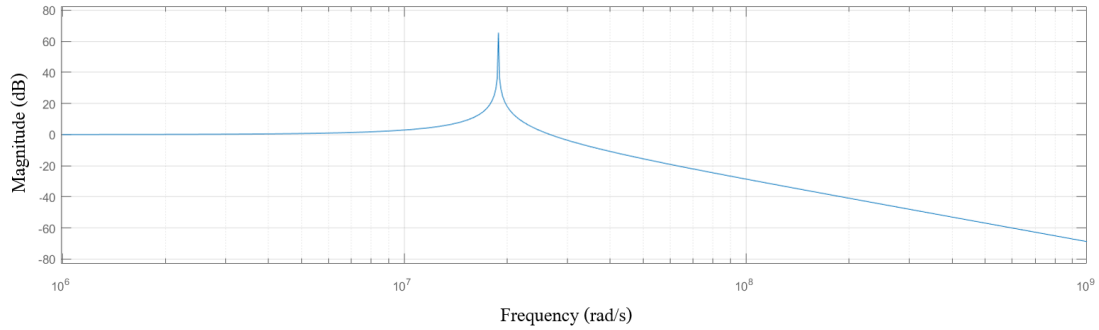


Figura 2.6: Filtro passa basso

frequenza di taglio il modulo scende con 40 dB/dec. Entrambe queste caratteristiche si possono notare nel digramma di Bode del modulo della rispettiva funzione di trasferimento.

$$F(s) = \frac{1}{1 + s * L/R + s^2 * LC} conw_r = \frac{1}{\sqrt{LC}} \tag{2.1}$$

2.2.2 Printed Circuit Board - PCB

Per la realizzazione del DAC è necessario realizzare il PCB della scheda, mostrato in Figura 2.7.

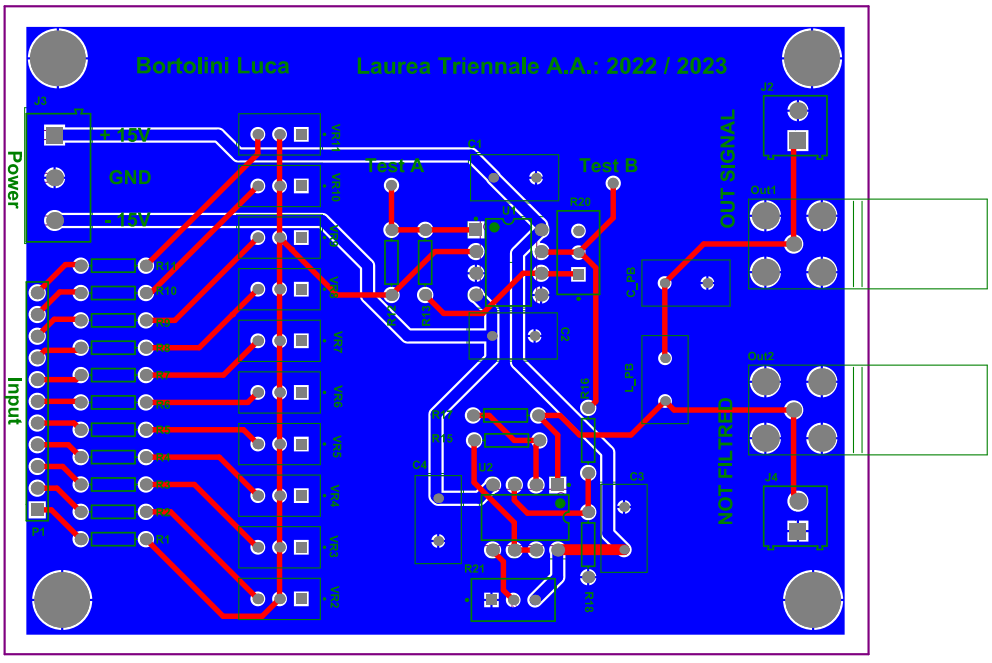


Figura 2.7: PCB del DAC a resistenze pesate

Sul piano inferiore sono state posizionate le piste relative ai + 15V e ai -15V ed è stato realizato un piano di massa. Sul layer superiore, invece, sono state posizionate tutte le piste relative ai segnali.

Per quanto riguarda Input e Output sono stati scelti dei morsetti a vite per facilitare i collegamenti. Inoltre sono stati predisposti tre punti per facilitare la fase di debug, questi riportano: uno il segnale che si ha all'uscita del sommatore invertente, uno il segnale che si ha dopo la configurazione invertente e l'ultimo il segnale che si ha appena prima del filtro passa basso.

Per una maggiore comodità, sono state inserite anche due uscite per cavo BNC, così da facilitare di gran lunga i collegamenti in fase di test della scheda. Queste due uscite riportano l'uscita finale del convertitore e il segnale che si avrebbe prima della fase di filtraggio.

Capitolo 3

Schema a blocchi FPGA

Nelle prossime sezioni verrano analizzati i principali blocchi del progetto.

3.1 Ricevitore UART

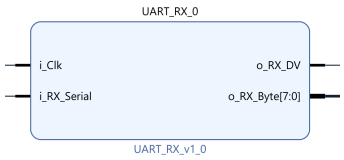


Figura 3.1: Ricevitore UART

In questa sezione viene trattato il modulo del Ricevitore UART, che ha il compito di ricevere le informazioni in modalità seriale dal PC e trasferirle ai moduli di composizione, che verranno mostrati in seguito.

Questo modulo ha due segnali di ingresso e due segnali di output:

- i_Clk ⟶ riceve in ingresso il segnale di clock del sistema
- i_RX_Serial ⟶ rivece in ingresso i singoli bit trasmessi in modalità seriale
- o_RX_DV ⟶ genera in uscita un impulso
- o_RX_Byte ⟶ genera in uscita la parola a otto bit ricevuta in ingresso

Com'è di facile intuizione la tipologia di comunicazione utilizzata è la comunicazione UART (*Universal Asynchronous Reciver and Trasmitter*), quindi è una comunicazione asincrona e dispone di un solo collegamento per trasmettere il segnale, l'altro riferimento è il potenziale di massa.

La linea seriale quando non si ha trasmissione è a livello logico '1', per iniziare la comunicazione, quindi, viene inviato un bit di start di valore logico '0'. A seguire il bit di start ci sono otto bit che contengono l'informazione da trasmettere e infine si ha un bit di stop, ovvero la linea viene riportata al livello logico '1'. La trasmissione dei bit, come si nota in Figura 3.2 avviene dal meno significativo (LSB) al più significativo (MSB).

Dato che la trasmissione è asincrona è necessario il segnale di clock per potersi sincronizzare con i dati, Altro parametro fondamentale per poter decodificare

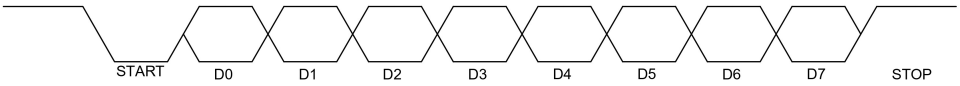


Figura 3.2: Trasmissione seriale di otto bit

i dati ricevuti è il *baud rate*, il quale indica la velocità di trasmissione ovvero il numero di bit al secondo che vengono trasmessi. Nel progetto in esame tale parametro è stato fissato a 115 200 bit/s, inoltre la comunicazione è stata impostata con otto bit per parola e un solo bit di stop; il bit di parità è stato tralasciato.

Il ricevitore, quando rileva la presenza di un bit di start, si "posiziona" al centro del periodo di ogni bit e legge tutti i valori ricevuti salvandoli uno per volta in un array. Quando si arriva alla fine della trasmissione, e quindi si riceve il bit di stop, gli otto bit ricevuti vengono trasmessi in uscita attraverso il segnale *o_RX_Byte*. Appena dopo viene inviato un impulso sul segnale *o_RX_DV* per far sapere ai moduli successivi che il segnale trasmesso è stato ricevuto ed è pronto per essere lavorato.

3.2 Compositore di frequenza

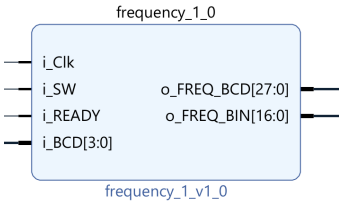


Figura 3.3: Compositore di frequenza

In questa sezione viene trattato il modulo del compositore di frequenza, che legge i valori emessi dal modulo ricevitore UART e compone la frequenza desiderata dall'utente.

Questo modulo ha quattro segnali di ingresso e due segnali di output:

- i_Clk ⟶ riceve in ingresso il segnale di clock del sistema
- i_SW ⟶ legge lo switch d'ingresso per selezionare la variazione di frequenza
- i_READY ⟶ è collegato al segnale *o_RX_DV*
- i_BCD ⟶ riceve in ingresso i quattro bit meno significativi degli otto ricevuti attraverso la comunicazione seriale
- o_FREQ_BCD ⟶ genera in uscita la frequenza selezionata dall'utente in codifica BCD
- o_FREQ_BIN ⟶ genera in uscita la frequenza selezionata dall'utente in codifica binaria

In uscita dal modulo si ha sia codifica BCD che codifica binaria in quanto all'interno del progetto sono necessarie entrambe e convertirle in fase di costruzione è più semplice rispetto a tradurle in seguito. La codifica BCD serve per visualizzare la frequenza sui display a sette segmenti, mentre la codifica binaria serve al controller per eseguire tutti i calcoli neccesari a una corretta generazione della forma d'onda.

Il funzionamento di questo modulo è molto semplice:

1. Per abilitare la modifica della frequenza si deve attivare l'ingresso *i_SW* attraverso lo switch presente sulla board;
2. Una volta attivato, il modulo attende che ci sia un impulso sul segnale di ingresso *i_READY*, questo significa che è disponibile una cifra della nuova frequenza;
3. A questo punto, quindi, per generare la frequenza BCD si fa uno shift a sinistra di quattro posizioni della frequenza preesistente e si posizionano nei quattro bit meno significativi i bit appena ricevuti attraverso il segnale *i_BCD*.
4. Per generare invece la frequenza in codifica binaria si moltiplica per dieci la frequenza binaria preesistene e si somma il valore appena ricevuto in ingresso.

Se c'è stato un fronte di salita sullo switch per attivare il modulo significa che si vuole cambiare la frequenza e di conseguenza il valore della frequenza preesistente verrà posto a zero.

N.B.: La frequenza BCD viene mandata in uscita ad ogni modifica (ovvero ad ogni nuovo valore che viene aggiuto) così l'utente ha un feedback immediato relativamente all'invio delle cifre desiderate.

3.3 Compositore di dutyCycle

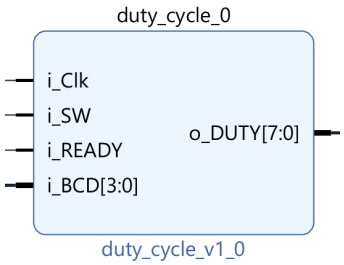


Figura 3.4: Compositore di DutyCycle

Si analizza ora il modulo di composizione del DutyCycle, questo modulo ha i seguenti segnali di ingresso e uscita:

- i_Clk ⟶ riceve in ingresso il segnale di clock del sistema
- i_SW ⟶ legge lo switch d'ingresso per selezionare la variazione di frequenza
- i_READY ⟶ è collegato al segnale *o_RX_DV*

- i_BCD ⟶ riceve in ingresso i quattro bit meno significativi degli otto ricevuti attraverso la comunicazione seriale
- o_DUTY ⟶ genera in uscita il DutyCycle su una parola binaria a otto bit

Il funzionamento di questo modulo è del tutto analogo a quello precedente, si discosta solamente per un particolare: l'uscita è solamente in codifica BCD, quindi la parola binaria è costituita dall'accostamento dei due segnali ricevuti in ingresso quando il modulo è abilitato.

3.4 Compositore di dati per RAM

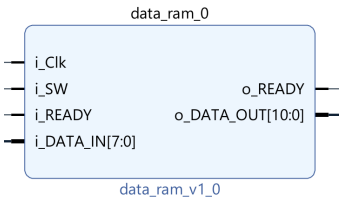


Figura 3.5: Compositore di parole da salvare in RAM

Viene preso in esame ora l'ultimo modulo compositore, in particolare quest'ultimo realizza le parole che verranno in seguito salvate nella memoria RAM. I segnali di ingresso e uscita non si discostano molto da quelli precedenti, infatti si ha:

- i_Clk ⟶ riceve in ingresso il segnale di clock del sistema
- i_SW ⟶ legge lo switch d'ingresso per selezionare l'abilitazione a modificare i dati salvati nella RAM
- i_READY ⟶ è collegato al segnale *o_RX_DV*
- i_DATA_IN ⟶ riceve in ingresso gli otto bit ricevuti in ingresso attraverso la comunicazione seriale
- o_READY ⟶ genera in uscita un impulso quando è disponibile il dato in uscita
- o_DATA_OUT ⟶ genera in uscita la parola binaria a 11 bit da salvare nella memoria

Questo modulo riceve in ingresso tutti gli otti bit ricevuti attraverso comunicazione seriale perchè la RAM dispone di celle di memoria di lunghezza pari a undici bit, vengono quindi considerate due parole a otto bit dove nella prima si prendono tutti i bit, mentre nella seconda ricevuta si considerano solo i tre bit meno significativi. Questi undici bit verranno emessi attraverso l'uscita *o_DATA_OUT* e saranno ricevuti da un modulo che avrà lo scopo di fare da interfaccia con la memoria RAM.

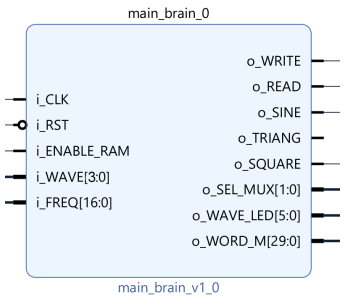


Figura 3.6: Main controller del sistema

3.5 Controller

Viene analizzato ora il modulo centrale del progetto, questo modulo "decide" la forma d'onda da generare in uscita, la frequenza a cui generarla e l'accensione di alcuni LED che facilitano l'utente finale nell'uso del prodotto. Vengono di seguito elencati gli ingressi e le uscite di questa parte di progetto:

- i_Clk ⟶ riceve in ingresso il segnale di clock del sistema
- i_RST ⟶ riceve in ingresso un eventuale segnale di reset di sistema
- i_ENABLE_RAM ⟶ è collegato a uno switch che seleziona quando modificare i dati salvati nella RAM
- i_WAVE ⟶ è collegato agli switch sulla scheda che selezionano la forma d'onda che si desidera in uscita
- i_FREQ ⟶ riceve in ingresso la frequenza inserita in codifica binaria
- o_WRITE ⟶ è attivo alto quando si vuole scrivere sulla RAM, quindi variandone le parole salvate
- o_READ ⟶ è attivo alto quando si vuole leggere dalla RAM
- o_SINE ⟶ è attivo alto quando si seleziona la funzione sinusoidale
- o_SQUARE ⟶ è attivo alto quando si seleziona la forma d'onda quadra
- o_SEL_MUX ⟶ genera in uscita un segnale per selezionare quale forma d'onda deve essere trasmessa in uscita fra tutte quelle realizzabili
- o_WAVE_LED ⟶ genera in uscita un segnale che accende il LED sulla scheda relativamente alla forma d'onda selezionata
- o_WORD_M ⟶ genera in uscita la *tuning_word* per ottenere le forme d'onda sinusoidale e triangolare

Per ottenere un sistema robusto è stata creata una scala di priorità rispetto ai segnali che si ricevono in ingresso dagli switch, ovvero:

1. il segnale di *Reset* ha priorità massima, se questo è attivo tutte le uscite vengono poste a valore logico zero;

2. il segnale *i_ENABLE_RAM* ha priorità subito inferiore quindi, se attivato, le uscite relative alle forme d'onda vengono poste a valore logico zero e viene attivato il segnale *o_WRITE* che abilità la scrittura sulla RAM;

3. infine ci sono gli switch di selezione delle forme d'onda, che devono essere attivati in modo esclusivo, in caso contrario non viene generata alcuna forma d'onda in uscita.

Per il calcolo della *tuning_word* si è utilizzata la Formula (1.3), data però l'impossibilità della precisione infinita si è optato per apportarle una semplificazione:

```
p := (i_FREQ * std_logic_vector(to_unsigned(10995,23)));
WORD_M <= p(39 downto 10);
```

Per ottenere questa formula sono state fatte le seguenti semplificazioni:

$$M = \frac{2^N * f_{out}}{f_{clk}} = \frac{2^{30} * f_{out}}{10^8} = f_{out} * \frac{2^N}{f_{clk}} = f_{out} * 10.73741824 = \\ = f_{out} * 10.73741824 * \frac{2^{10}}{2^{10}} = f_{out} * \frac{10\,995}{2^{10}} = (f_{out} * 10\,995) >> 10 \tag{3.1}$$

Per ottenere la formula finale si è quindi moltiplicato e diviso per 2^{10}, così facendo $\frac{2^N}{f_{clk}}$ è diventato approssimabile con un valore intero. Per dividere poi per 2^{10} è sufficiente eseguire uno shift verso destra di dieci posizioni, per far ciò, come si può notare dalle righe di codice sopra riportate, sono stati presi i bit più significativi del risultato della moltiplicazione, tralasciando i dieci bit meno significativi.

3.6 Costruzione onda quadra

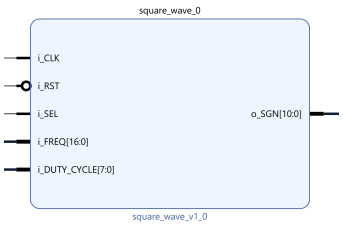


Figura 3.7: Generatore di forme d'onda quadra

Viene ora esposto il modulo che si occupa della costruzione della forma d'onda quadra, vengono elencati i segnali di input e output:

- i_Clk ⟶ riceve in ingresso il segnale di clock del sistema
- i_RST ⟶ riceve in ingresso un eventuale segnale di reset di sistema
- i_SEL ⟶ è collegato all'uscita *o_SQUARE* che abilita o meno la generazione in uscita della forma d'onda quadra
- i_FREQ ⟶ riceve in ingresso la frequenza inserita in codifica binaria

- i_DUTY_CYCLE ⟶ riceve in ingresso la tipologia di duty-cycle che deve avere il segnale in uscita
- o_SGN ⟶ genera in uscita la forma d'onda desiderata

Come prima azione, quando l'ingresso *i_SEL* è attivo, viene controllato se è stata cambiata la frequenza desiderata; se si è verificato questo, si calcola il nuovo *switching_count* ovvero il numero di fronti di salita del segnale di clock che compongono un periodo del segnale desiderato. Dopodichè utilizzando uno switch case basato sul valore di duty-cycle ricevuto in ingreso si vanno ad assegnare i valori alle variabili *high_count* e *low_count*, le quali rappresentano il numero di fronti di salita del segnale di clock rispettivamente del periodo a valore logico uno e del periodo a valore logico zero.

I valori possibili di duty-cycle sono: 0%, 25%, 33%, 50%, 66%, 75%, 100%; qualsiasi altro valore inserito verrà settato di default a un duty-cycle pari al 50%.

Una volta calcolati tutti questi dati si incrementa la varaibile *count* ad ogni fronte di salita del segnale di clock e quando l'uscita è a valore logico zero e si è superato il valore salvato in *low_count*, si passa al valore logico uno e viceversa, come mostrato di seguito:

```
if state = '1' and count >= high_count then
  SGN <= (others => '0');
  count := 0;
  state := '0';
elsif state = '0' and count >= low_count then
  SGN <= (others => '1');
  count := 0;
  state := '1';
end if;
```

3.7 Interfaccia per memoria RAM

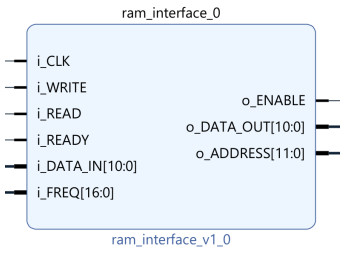


Figura 3.8: Modulo di interfacciamento con la RAM

Questo modulo è stato progettato con lo scopo di interfacciarsi con la memoria RAM, per gestire le due fasi: lettura e scrittura. I segnali di ingresso e uscita di questo modulo sono:

- i_CLK ⟶ riceve in ingresso il segnale di clock del sistema
- i_WRITE ⟶ è collegato al segnale *o_WRITE* del main controller, abilita la scrittura nella RAM

- i_READ ⟶ è collegato al segnale *o_READ* del main controller, abilita la lettura dalla RAM
- i_READY ⟶ è collegato al segnale *o_READY* del compositore di parole (Figura 3.5)
- i_DATA_IN ⟶ è collegato al segnale *o_DATA_OUT* del compositore di parole (Figura 3.5)
- i_FREQ ⟶ riceve in ingresso la frequenza selezionata dall'utente in codifica binaria
- o_ENABLE ⟶ quando è attivo alto abilita la scrittura sulla RAM
- o_DATA_OUT ⟶ emette in uscita la parola binaria da salvare nella memoria RAM
- o_ADDRESS ⟶ genera in uscita l'indirizzo della memoria al quale si deve leggere o scrivere il dato

Il codice di questo modulo è stato diviso in due processi: uno si occupa del calcolo dello *switching_count*, l'altro si occupa della generazione dell'indirizzo da fornire alla memoria RAM, del segnale *o_ENABLE* e del segnale *o_DATA_OUT*.

Per il calcolo dello *switching_count*, il calcolo da svolgere risulta molto semplice ed è riportato di seguito:

```
1 switching_count <= integer(F_clock / (to_integer(unsigned(FREQ)) *
    4096));
```

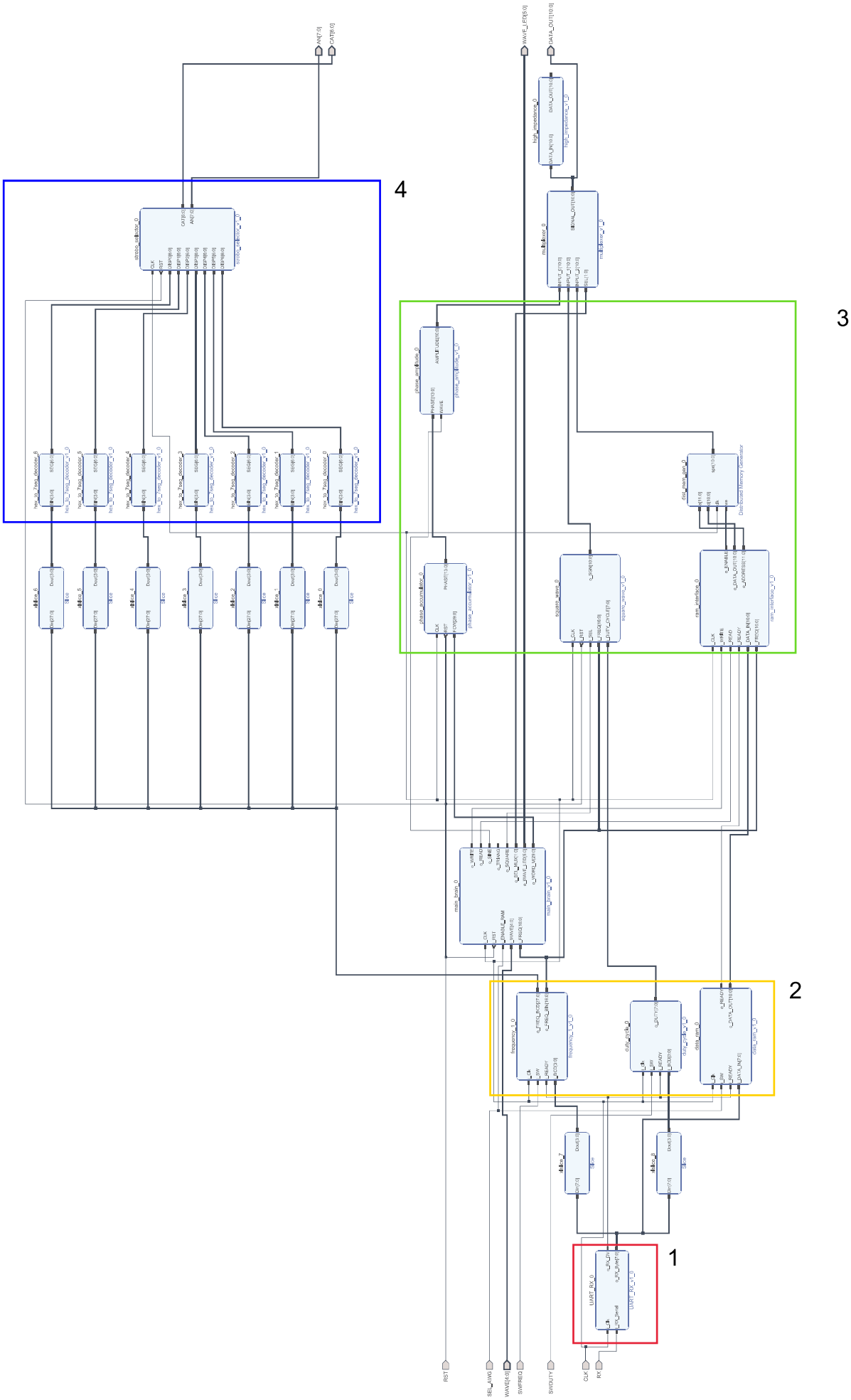
Per quando riguarda, invece, i segnali di output del modulo se ne occupa il secondo processo.

In fase di lettura ad ogni fronte di salita del segnale di clock si incrementa la variabile del conteggio *r_count*, quando questa supera il valore salvato nello *switching_count* si incrementa il valore dell'indirizzo e si riporta il contatore al valore zero.

In fase di scrittura, se il segnale *i_READY* è attivo allora è pronto un dato in ingresso, quindi il segnale ricevuto in *i_DATA_IN* viene propagato in *o_DATA_OUT*, il segnale di enable viene posto al valore logico uno e viene incrementato di un'unità il valore dell'indirizzo della RAM.

Di seguito viene riportato il Block Design dell'intero progetto, sono evidenziate le quattro sezioni più importanti, in particolare si ha:

1. Modulo di ricezione della comunicazione seriale
2. Moduli relativi alla composizione di frequenza, duty cycle e parole per la RAM
3. Sezione di generazione delle forme d'onda
4. Sezione di visualizzazione della frequenza selezionata dall'utente sui display a sette segmenti presenti sulla board


4
3
2
1
RST
SEL_AWG
WAVE[1:0]
SWFREQ
SWDUTY
CLK
RX

Capitolo 4

Simulazione del progetto

4.1 Simulazione virtuale

Con simulazione virtuale si fa riferimento alla simulazione che viene eseguita grazie al software Vivado, la simulazione viene eseguita con il progetto già sintetizzato, così facendo si ottiene una simulazione che si avvicina il più possibile al comportamento che si riscontrerà una volta programmata l'FPGA.

La simulazione delle forme d'onda sinusoidale e triangolare è già stata discussa all'interno della sezione *1.3 - Simulazione*, sarebbe quindi rindondante riportare anche in questo capitolo i medesimi risultati.

Si passa quindi alla simulazione ed analisi delle forme d'onda rimanenti, ovvero la forma d'onda quadra e la forma d'onda arbitraria.

4.1.1 Forma d'onda quadra

La forma d'onda quadra ha due gradi di libertà, la frequenza e il duty cycle. I valori della frequenza hanno lo stesso range delle altre forme d'onda, ovvero la frequenza massima generabile è pari a 100 000 Hz. Per quanto riguarda il duty cycle si ha un numero finito di valori che possono essere assunti, tali valori sono stati scelti secondo frazioni caratteristiche; i valori tra cui si può scegliere sono: 0%, 25%, 33%, 50%, 66%, 75%, 100%.

Di seguito verranno analizzate alcune simulazioni in cui si varierà frequenza e duty cycle, si analizzerà l'errore che si ottiene rispetto a entrambe le caratteristiche. Le frequenze simulate sono le medesime della simulazione delle onde sinusoidale e triangolare. Come prima frequenza si ha quindi 3 578 Hz, la forma d'onda ottenuta in uscita è rappresentata in Figura 4.1.

Dai *marker* inseriti si ottengono i seguenti risultati:

$$T_{teorico} = \frac{1}{f} = \frac{1}{3578} \simeq 279.486\,\mu\text{s}$$
$$T_{mis} = 279.48\,\mu\text{s} \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |279.48 - 279.486| = 0.006\,\mu\text{s}.$$

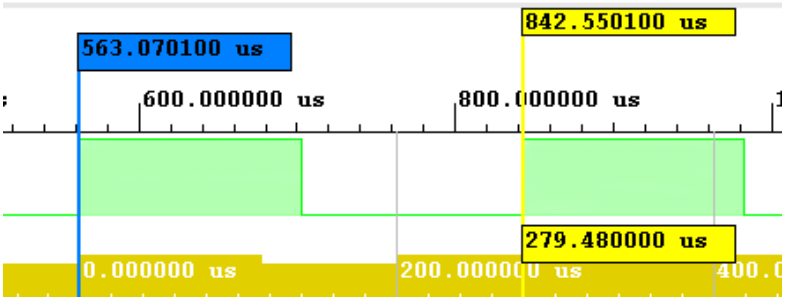


Figura 4.1: Simulazione forma d'onda quadra a 3 578 Hz

L'errore che si ottiene può essere attribuito all'errore di troncamento e alla sensibilità del clock di sistema, in quanto un periodo di clock dura 10 ns.

Si analizzano ora altri risultati spaziati all'interno del range di frequenze generabili, le cui immagini, verranno inserite nell'appendice alla fine di questa tesi.

- Frequenza impostata a 88 Hz

$$T_{teorico} = \frac{1}{f} = \frac{1}{88} \simeq 11.364\,ms$$

$$T_{mis} = 11.364\,ms \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |11.364 - 11.364| = 0\,ms$$

$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = 0\,\%$$

In questo caso si ha dunque la massima precisione.

- Frequenza impostata a 23 kHz

$$T_{teorico} = \frac{1}{f} = \frac{1}{23*10^3} \simeq 43.478\,\mu s$$

$$T_{mis} = 43.46\,\mu s \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |43.46 - 43.478| = 0.018\,\mu s$$

$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.018*10^{-6}}{43.478*10^{-6}} * 100 = 0.004\%$$

- Frequenza impostata a 100 kHz

$$T_{teorico} = \frac{1}{f} = \frac{1}{100000} \simeq 10\,\mu s$$

$$T_{mis} = 10\,\mu s \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = 0\,\mu s$$

L'errore percentuale $\epsilon_{\%}$ è nullo, in quanto il periodo teorico e periodo misurato con i cursori si eguagliano.

Dai valori sopra riportati si può concludere che la generazione della forma d'onda quadra, al variare della frequenza, non comporta un errore considerevole.

Si passa ora ad un'analisi al variare del duty cycle. Per quest'analisi la frequenza è stata mantenuta costante a 1 kHz, dunque il periodo teorico è pari a $T_{teorico} = \frac{1}{1000} = 1\,ms$.

Di seguito viene riportata la forma d'onda con duty cycle pari al 25%, dunque $t_{high} = T_{teorico} * 0.25 = 250\,\mu$s. Come si nota in Figura 4.2 il tempo a valore logico 1 coincide esattamente con il t_{high} appena calcolato, si ha perciò un errore nullo sul duty cycle.

L'errore nullo appena ottenuto si riscontra anche con tutti gli altri valori di duty cycle, le cui immagini sono riportate nell'appendice al termine della tesi.

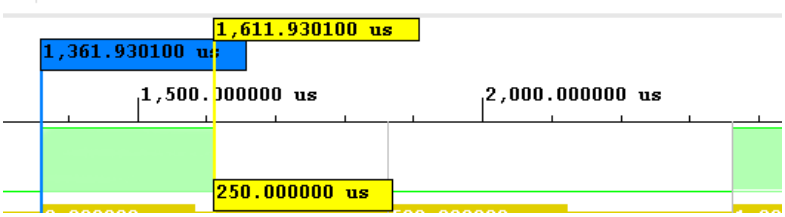


Figura 4.2: Simulazione forma d'onda quadra con duty cycle del 25%

4.1.2 Forma d'onda arbitraria

Come forma d'onda arbitraria è stato scelto un seno modulato in seno. Per identificare i campioni da salvare in memoria è stato fatto un campionamento del segnale prescelto attraverso un semplice script Matlab. Sono stati presi tanti campioni quante le allocazioni di memoria disponibili nella RAM, ovvero 4 096 valori; tali campioni vengono precaricati in memoria in fase di programmazione dell'FPGA.

É possibile modificare la forma d'onda arbitraria, sia in modo parziale che nella sua integrità, attraverso la comunicazione seriale. Dopo aver attivato il relativo switch per l'abilitazione alla scrittura nella memoria RAM è possibile inviare i bit da sostituire; la sovrascrittura partirà dalla cella di memoria che ha indirizzo 000...000 e proseguirà incrementando di un'unità l'indirizzo di memoria ad ogni valore ricevuto in ingresso.

Si analizzano ora alcune simulazioni, a diverse frequenze, della generazione della forma d'onda arbitraria.

Come prima frequenza analizzata si ha $f = 3\,578\,Hz$, l'uscita ottenuta è riportata in Figura 4.3

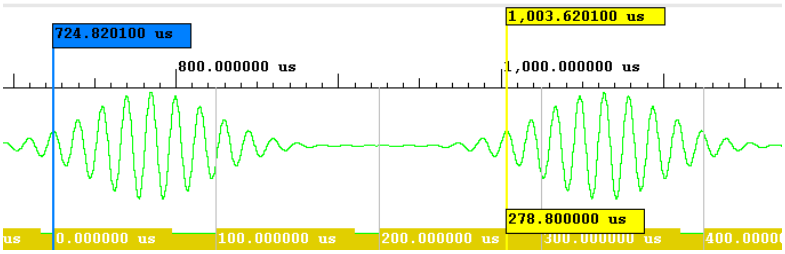


Figura 4.3: Simulazione forma d'onda arbitraria a 3 578 Hz

$$T_{teorico} = \frac{1}{f} = \frac{1}{3578} \simeq 279.486\,\mu s$$

$$T_{mis} = 278.80\,\mu s \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |278.80 - 279.486| = 0.686\,\mu s$$

$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.686*10^{-6}}{279.486*10^{-6}} * 100 = 0.25\,\%$$

Le immagini relative ai successivi dati sono riportate in appendice a questa tesi.

- Frequenza impostata a 88 Hz

$$T_{teorico} = \frac{1}{f} = \frac{1}{88} \simeq 11.364 ms$$

$$T_{mis} = 11.387\,ms \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |11.387 - 11.364| = 0.023\,ms$$

$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.023*10^{-3}}{11.364*10^{-3}} * 100 = 0.2\,\%$$

- Frequenza impostata a 23 kHz

$$T_{teorico} = \frac{1}{f} = \frac{1}{23*10^3} = 43.478\,\mu s$$

$$T_{mis} = 43.48\,\mu s \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = |43.48 - 43.478| = 0.002\,\mu s$$

$$\longrightarrow \epsilon_{\%} = \frac{\Delta t}{T_{teorico}} * 100 = \frac{0.002*10^{-6}}{43.478*10^{-6}} * 100 = 0.005\,\%$$

- Frequenza impostata a 100 kHz

$$T_{teorico} = \frac{1}{f} = \frac{1}{100*10^3} = 10.0 \mu s$$

$$T_{mis} = 10.0 \mu s \longrightarrow \Delta t = |T_{mis} - T_{teorico}| = 0\,\mu s$$

Come nel caso dell'onda quadra l'errore percentuale $\epsilon_{\%}$ è nullo, in quanto il periodo teorico e il periodo misurato con i cursori si eguagliano.

Con quest'analisi si ricava che anche la forma d'onda arbitraria ha un errore molto piccolo, che può essere interamente attribuito alla sensibilità del clock, come era stato enunciato nella sezione precedente.

Come è stato affermato in precedenza è possibile variare la forma d'onda salvata in memoria, tale operazione viene svolta inviando un valore per volta attraverso la comunicazione seriale. Il sistema si autoregola per far in modo che quando si inizia a sovrascrivere la memoria si parta sempre dal primo valore salvato.

I valori in memoria hanno una lunghezza pari a 11 bit, con la comunicazione seriale però si inviano parole lunghe 8 bit. Per risolvere tale problema si inviano due parole per salvarne una in memoria, della prima parola ricevuta si vanno a considerare tutti i bit di dato, mentre della seconda parola ricevuta si valutano solamente i primi tre bit. Così facendo al termine delle due parole si hanno gli 11 bit da salvare in memoria RAM.

Di seguito, in Figura 4.4, la forma d'onda arbitraria è stata modificata solamente in una sua frazione, i valori iniziali sono stati sostituiti con una rampa crescente.

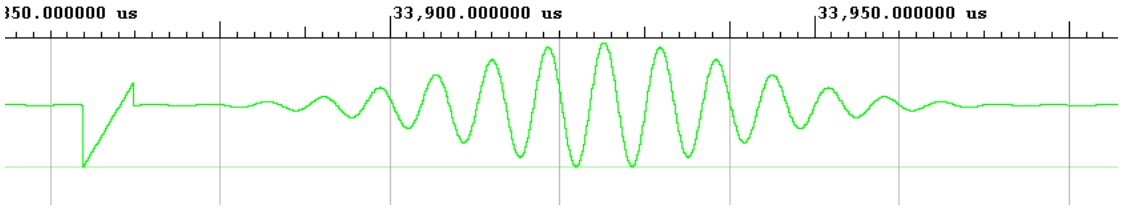


Figura 4.4: Simulazione forma d'onda arbitraria in cui alcuni valori sono stati modificati

4.2 Simulazione fisica

Per l'analisi delle forme d'onda ottenute all'uscita dell'FPGA, ovvero la forma d'onda finale che potrà utilizzare l'utente, viene presa in considerazione la forma d'onda quadra. Questa viene simulata a diverse frequenze e variando il valore del duty cycle, le varie frequenze sono state scelte per avere un'ampia spazzolata delle frequenze ottenibili in uscita da progetto.

Come si può facilmente notare in Figura 4.5 la forma d'onda ottenuta presenta un errore rispetto alla forma d'onda ideale, infatti:

$$F_{teorica} = 12\,Hz,\ F_{misurata} = 12.02\,Hz \rightarrow \Delta F = F_{misurata} - F_{teorica} = 0.02\,Hz$$

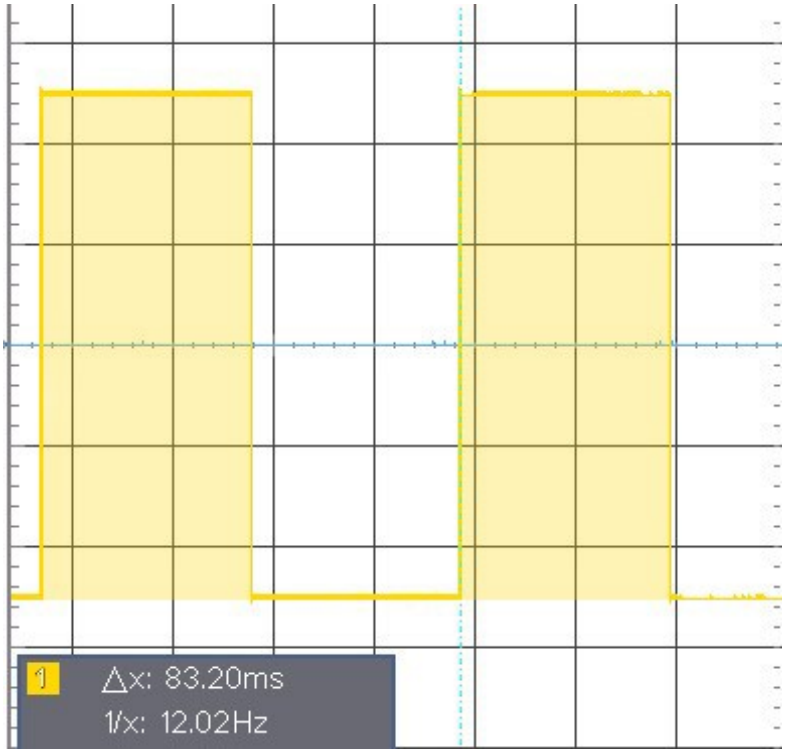


Figura 4.5: Forma d'onda quadra in uscita all'FPGA

L'errore ottenuto è del tutto trascurabile dato che il progetto è di un prototipo e non è stato ottimizzato per ottenere un errore infinitesimo, nel successivo

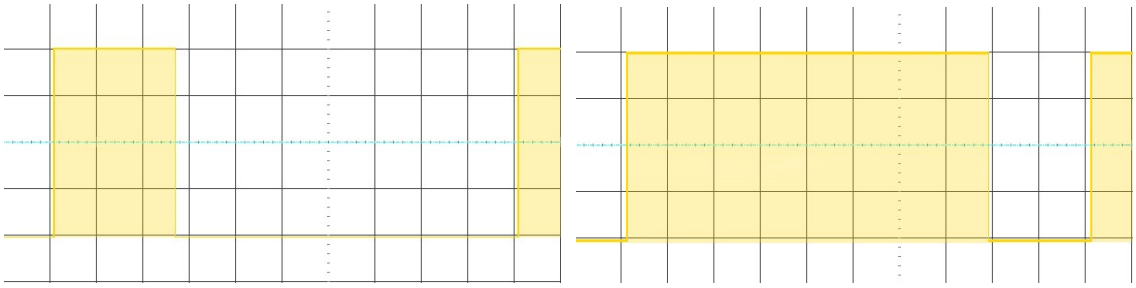
capitolo, nel quale si parla di possibili miglioramenti, vengono enunciate alcune possibili implementazioni per migliorare tale errore.

Sono state simulate anche altre frequenze per avere un'analisi a più ampio spettro, i risultati ottenuti sono i seguenti:

- Frequenza pari a 1 000 Hz
  $F_{teorica} = 1\,000\,Hz$, $F_{misurata} = 996\,Hz \rightarrow \Delta F = F_{misurata} - F_{teorica} = 4\,Hz$
- Frequenza pari a 23 000 Hz
  $F_{teorica} = 23\,000\,Hz$, $F_{misurata} = 23\,040\,Hz \rightarrow \Delta F = F_{misurata} - F_{teorica} = 40\,Hz$
- Frequenza pari a 83 000 Hz
  $F_{teorica} = 83\,000\,Hz$, $F_{misurata} = 83\,330\,Hz \rightarrow \Delta F = F_{misurata} - F_{teorica} = 330\,Hz$

Le misurazioni che sono state riportate appena sopra trovano conferma nelle immagini che vengono caricate nella relativa appendice al termine di questa tesi.

Generando un'onda quadra è possibile variare il duty cycle, sono stati generati due valori differenti di duty cycle: 25 % e 75 %. Le immagini dei risultati ottenuti sono riportati di seguito.



(a) Duty cycle del 25 % (b) Duty cycle del 75 %

Figura 4.6: Generazione della forma d'onda quadra a diversi duty cycle

Capitolo 5

Conclusioni

5.1 Implementazione

L'intero progetto è stato realizzato utilizzando l'FPGA presente sulla scheda Nexys 4DDR della Xilinx. L'impostazione della frequenza da generare in uscita viene fatta attraverso la comunicazione seriale, in fase di simulazione è stato utilizzato il software MobaXterm per attuare la comunicazione seriale e la scheda è stata collegata al PC attraverso il cavo USB. La frequenza impostata viene visualizzata sui display a sette segmenti presenti sulla board. Per selezionare la forma d'onda desiderata si attiva lo switch corrisponedente sulla scheda, come indicato in Figura 5.1.

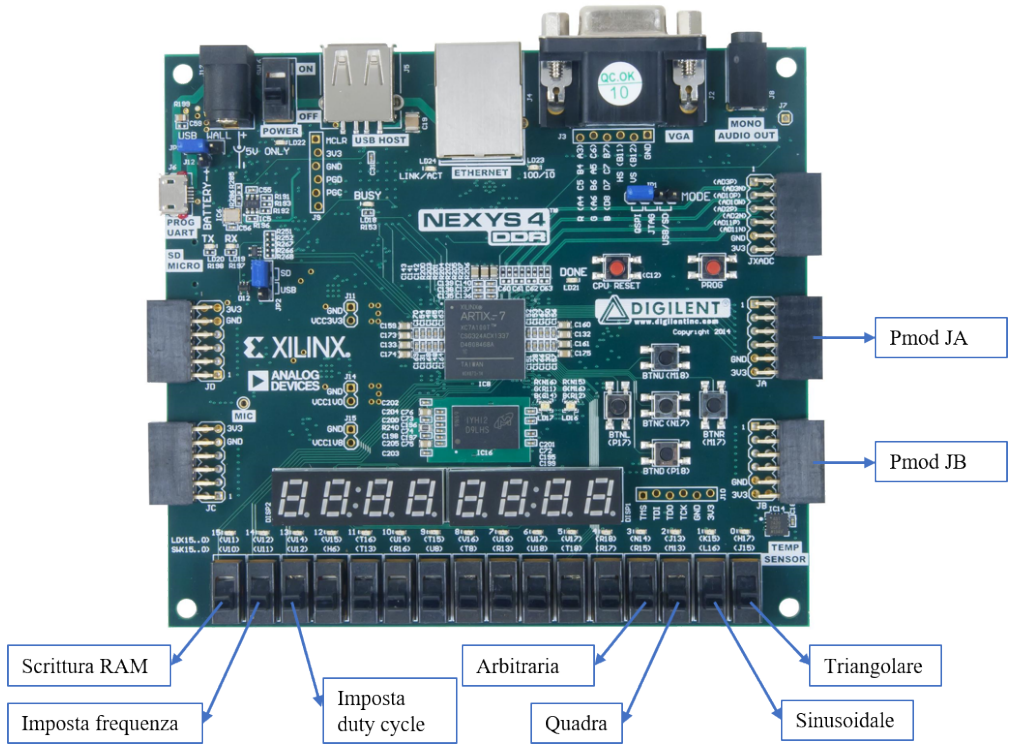


Figura 5.1: Abbinamento switch - funzione, fonte immagine della scheda: *Nexys 4 DDR Reference Manual* [5]

In Figura 5.1 sono riportati anche gli accoppiamenti switch-azione, ovvero quegli switch che permettono: l'inserimento della frequenza o del duty cycle prescelti e l'abilitazione alla sovrascrittura dei valori salvati in memoria per la forma d'onda arbitraria. All'attivazione di uno degli switch si accende il LED posto sopra a tale switch per identificare la corretta attivazione.

Una volta comunicata la frequenza desisderata e attivato lo switch relativo alla forma d'onda da generare, l'utente può ottenere l'uscita dalle porte Pmod presenti sulla scheda. In particolare è stata utilizzata la porta Pmod JA nella sua integrità, il bit meno significativo del segnale d'uscita si trova al pin 1 di questa porta, e la porta Pmod JB in una sua frazione, sono stati utilizzati i primi tre pin e il bit più significativo del segnale d'uscita si trova al pin tre di questa porta.

Dalle simulazioni che sono state riportate nel corso di questa tesi e dalle analisi che sono state fatte su tali simulazioni si può affermare che la generazione di tutte le forme d'onda è piuttosto accurata, infatti variando la frequenza in tutto lo spettro possibile del progetto l'errore percentuale della forma d'onda ottenuta è sempre di molto inferiore all' 1%. Quest'accuratezza può essere attribuita all'uso della tecnica DDS, la quale permette di avere una precisione pari al periodo del segnale di clock, in questo caso pari a $10\,ns$.

5.2 Futuri miglioramenti

In questa sezione vengono riportate alcune possibili modifiche che possono migliorare il progetto, sia per quanto riguarda l'utilizzo da parte dell'utente sia in merito alla precisione della generazione delle forme d'onda.

Una prima modifica per aumentare la portabilità del prodotto è inserire un potenziometro multigiri per selezionare la frequenza desiderata, grazie a questo si riuscirebbe ad eliminare la necessità della comunicazione seriale fin tanto che si utilizzano le forme d'onda presenti in memoria della board. É altrsì vero che quest'aggiunta comporta un aumento della complessità del sistema, è necessario infatti inserire ulteriori blocchi nel sistema per la ricezione e l'analisi del segnale analogico proveniente dal potenziometro.

Questo sistema inoltre è in grado di variare solamente la frequenza della forma d'onda, un parametro fondamentale da aggiungere sarebbe la possibilità di cambiare l'ampiezza del segnale generato: quando si sta analizzando un circuito con un generatore di forme d'onda si desidera valutare la risposta a diverse ampiezze del segnale d'ingresso. Tale modifica va apportata sulla parte analogica del progetto, perchè nella parte digitale si sfrutta la massima dinamica offerta; successivamente utilizzando un opportuno aplificatore si va a modificare l'ampiezza del segnale.

Appendice A

Immagini simulazione DDS

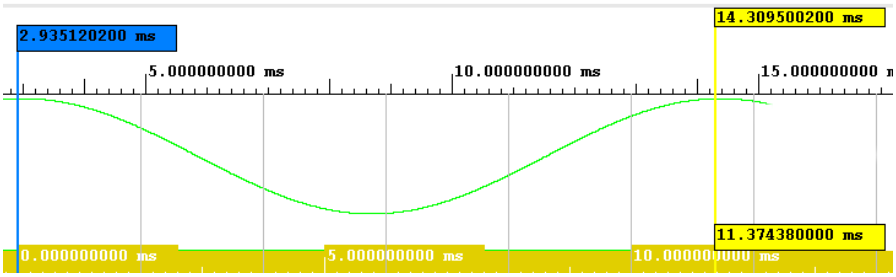


Figura A.1: Simulazione forma d'onda sinusoidale a 88 Hz

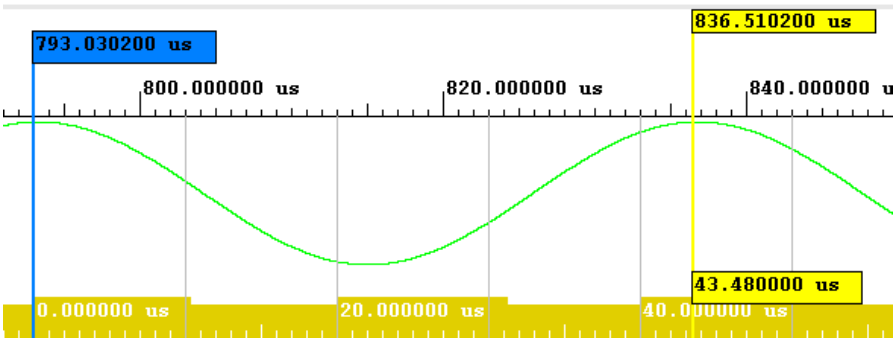


Figura A.2: Simulazione forma d'onda sinusoidale a 23 kHz

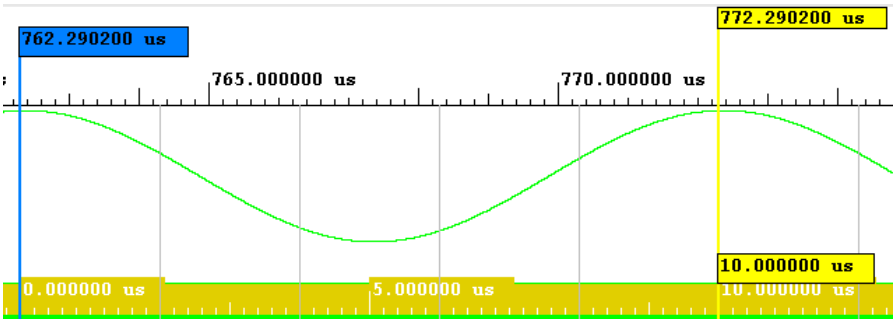


Figura A.3: Simulazione forma d'onda sinusoidale a 100 kHz

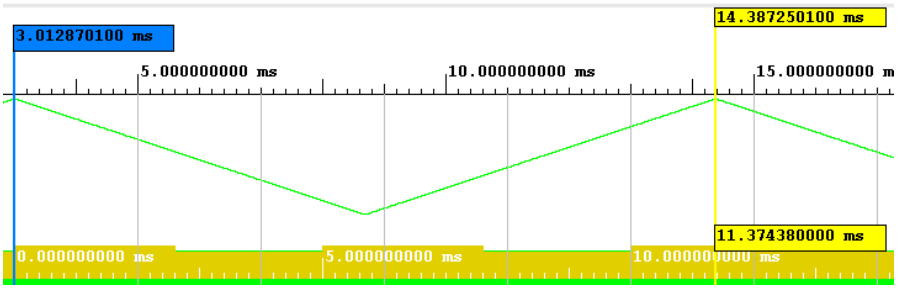


Figura A.4: Simulazione forma d'onda triangolare a 88 Hz

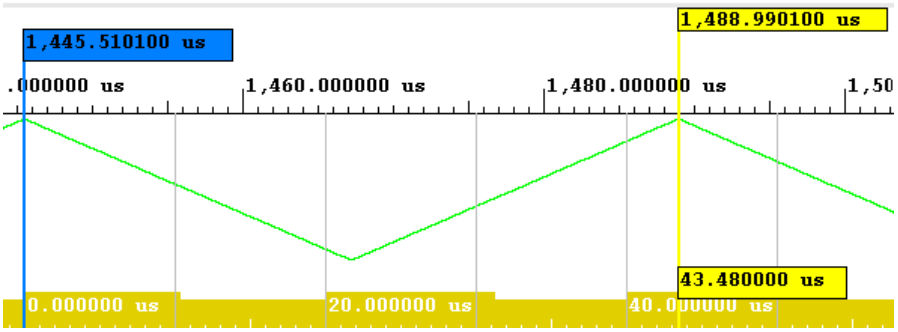


Figura A.5: Simulazione forma d'onda triangolare a 23 kHz

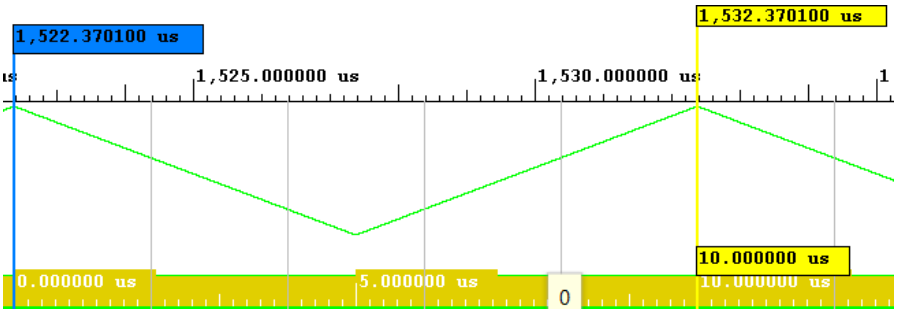


Figura A.6: Simulazione forma d'onda triangolare a 100 kHz

Appendice B

Immagini simulazione forme d'onda quadra e arbitraria

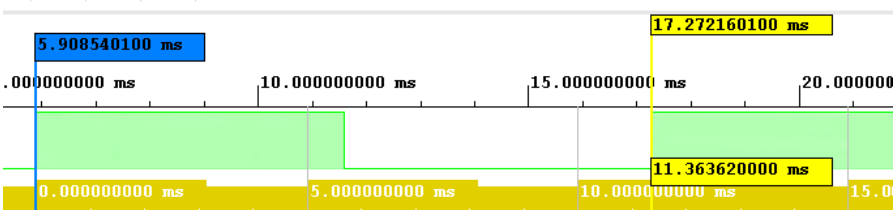


Figura B.1: Simulazione forma d'onda quadra a 88 Hz

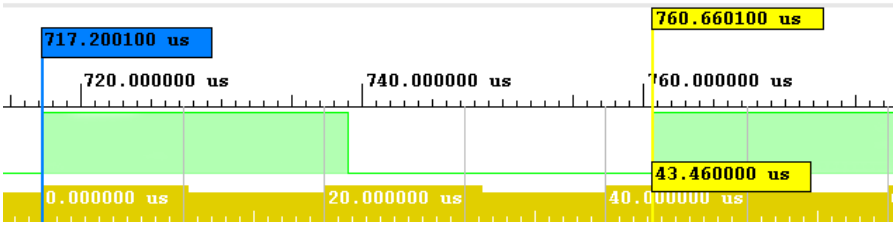


Figura B.2: Simulazione forma d'onda quadra a 23 kHz

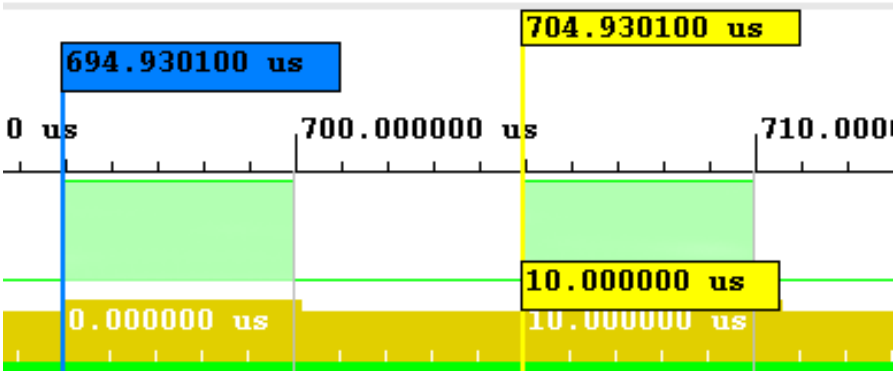


Figura B.3: Simulazione forma d'onda quadra a 100 kHz

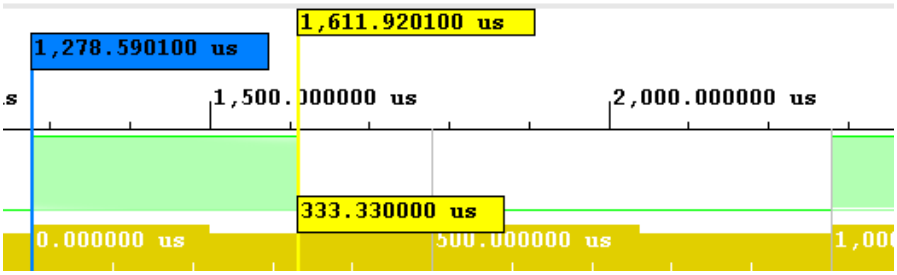


Figura B.4: Simulazione forma d'onda quadra con duty cycle 33%

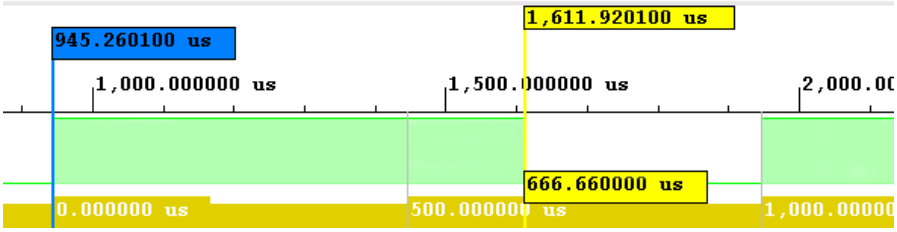


Figura B.5: Simulazione forma d'onda quadra con duty cycle 66%

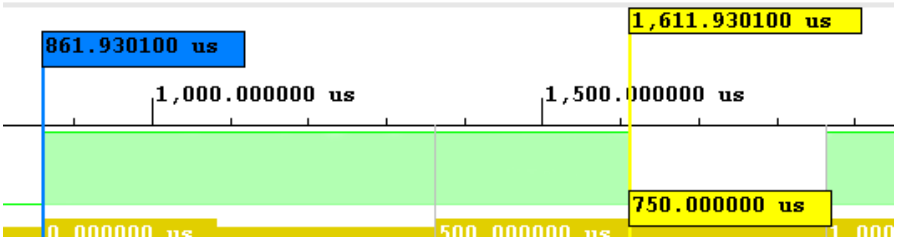


Figura B.6: Simulazione forma d'onda quadra con duty cycle 75%

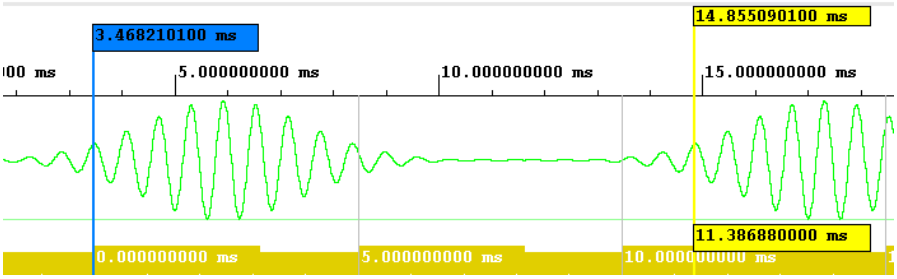


Figura B.7: Simulazione forma d'onda arbitraria a 88 Hz

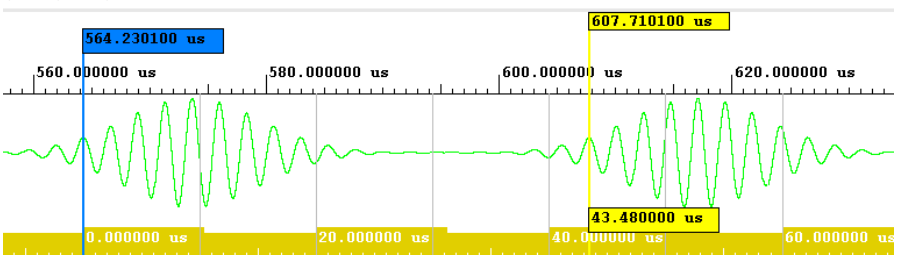


Figura B.8: Simulazione forma d'onda arbitraria a 23 kHz

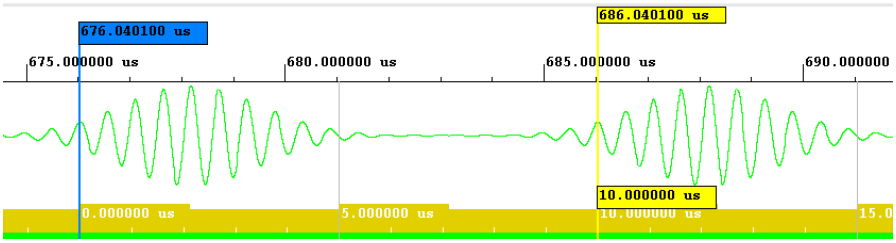


Figura B.9: Simulazione forma d'onda arbitraria a 100 kHz

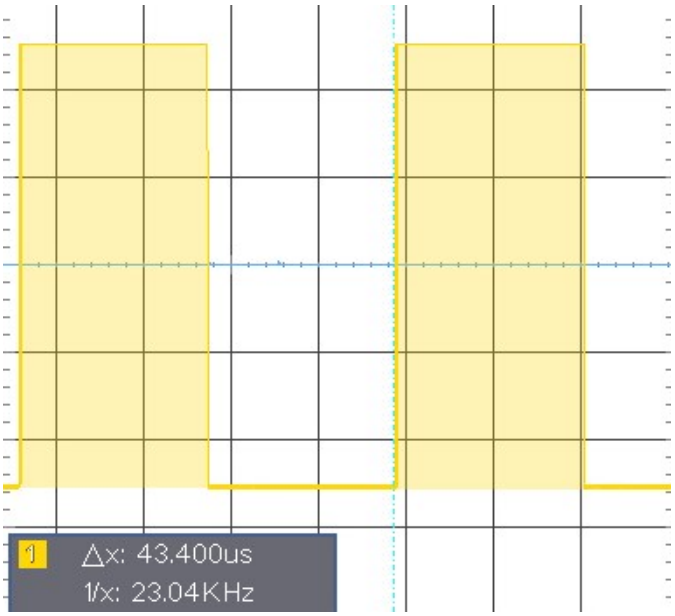


Figura B.10: Simulazione forma d'onda quadra nella realtà a 23 kHz

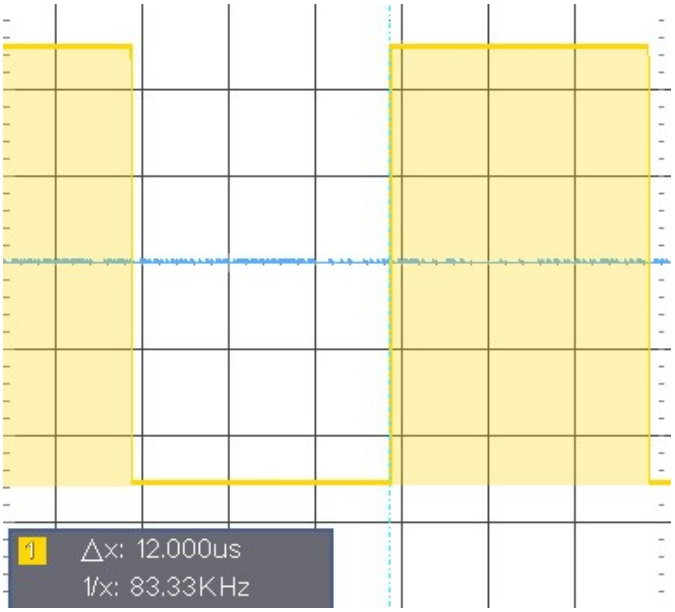


Figura B.11: Simulazione forma d'onda quadra nella realtà a 83 kHz

Bibliografia

[1] Eva Murphy, Colm Slattery, (2004)
All About Direct Digital Synthesis, Analog Dialogue 38-08
http://www.analog.com/library/analogdialogue/archives/38-08/dds.pdf

[2] Analog Devices, Inc, (1999)
A Technical Tutorial on Digital Signal Synthesis
http://www.analog.com/static/imported-files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf

[3] Digi-Key, (2019)
I principi base dei sintetizzatori digitali diretti (DDS), come sceglierli e usarli.
https://www.digikey.it/it/articles/the-basics-of-direct\-digital-synthesizers-ddss

[4] Arrow, (2017)
Che cos'è la sintesi digitale diretta?
https://www.arrow.com/it-it/research-and-events/videos/what-is-direct-digital-synthesis#:~:text=La%20sintesi%20digitale%20diretta%20(DDS,commutazione%20rapida%20tra%20queste%20frequenze.

[5] Digilent
Nexys 4 DDR Reference Manual
https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual

[6] Clive "Max" Maxfield, (2004)
The Design Warrior's Guide to FPGAs
https://blog.aku.edu.tr/ismailkoyuncu/files/2017/04/01_ebook.pdf

Ringraziamenti

In conclusione a questa tesi desidero ringraziare in primo luogo il relatore, Professor Vogrig Daniele, per i consigli, le indicazioni e il sostegno che ha dimostrato in questi mesi sia nella progettazione che nella stesura di questo elaborato.

Un rigraziamento particolare va alla mia famiglia che mi ha sempre sostenuto e mi è sempre stata vicina in tutti i momenti della vita. Grazie mamma e grazie Giulia per avermi supportato e spronato a dare sempre il massimo in questi anni di studio intenso ma affascinante.

É doveroso ringraziare gli amici di sempre, Luca, Max, Davide, Lorenzo; per essere sempre stati al mio fianco e per avermi fatto vivere momenti indimenticabili.

Nella vita ogni persona si crea delle seconde famiglie, la mia è il meraviglioso gruppo della Lego; grazie per le meravigliose esperienze di vita che ho condiviso con voi, e in particolare grazie a Michele per avermi dato la magnifica idea del progetto che è stato realizzato ed analizzato in questa tesi.

Grazie anche ai nuovi amici nati tra le aule dell'università, insieme abbiamo condiviso innumerevoli gioie e dolori.