Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea in Ingegneria dell'Informazione

Tesi di laurea triennale

Implementazione su FPGA di un generatore di forme d'onda programmabile basato su tecnica DDS

Laureando:
Michele Brian
Matricola INF-597524

Relatore:
Prof. Daniele Vogrig

Anno Accademico 2012-2013

A tutte le persone a me care.

“ Scientists investigate that which already is;
Engineers create that which has never been. ”
- Albert Einstein

Sommario

In questa Tesi viene presentato lo sviluppo di un generatore di forme d'onda basato sulla tecnica di *sintesi digitale diretta* (DDS). Il sistema è in grado di produrre tre tipi di forme d'onda, sinusoidale, triangolare e quadra. La selezione dell'onda e la frequenza di uscita sono impostate tramite comunicazione seriale.

Il progetto è stato sviluppato in linguaggio VHDL e implementato su un dispositivo FPGA della *Xilinx*. L'utilizzo di questo strumento consente di ottenere un circuito digitale facilmente riconfigurabile con la possibilità di essere implementato in un progetto più ampio.

Il vantaggio principale della sintesi digitale diretta è che frequenza, fase e ampiezza del segnale di uscita possono essere manipolate in modo preciso e rapido attraverso un controllo completamente digitale. Inoltre il recente progresso tecnologico consente di costruire dispositivi DDS molto compatti con consumi di potenza molto ridotti. Queste principali caratteristiche hanno reso la tecnologia DDS molto popolare in un'ampia gamma di applicazioni in campi come la medicina, l'industria, la strumentazione, le comunicazioni e la difesa.

Indice

Capitolo 1

Dispositivi

1.1 FPGA

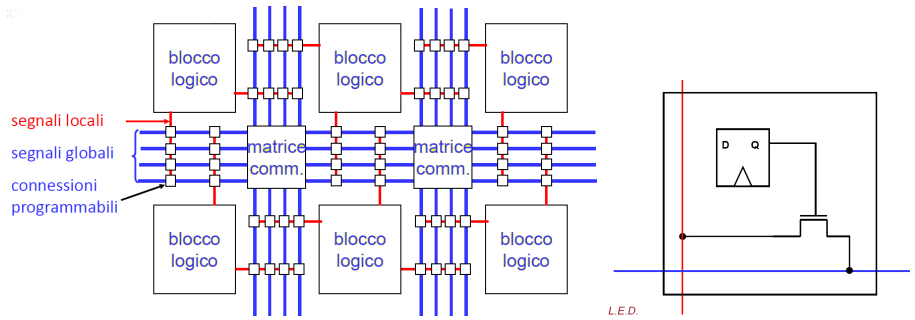
1.1.1 Generalità

I dispositivi FPGA (*Field Programmable Gate Array*) sono circuiti integrati digitali costituiti da blocchi logici configurabili collegati tra loro da interconnessioni programmabili. Il termine "Field Programmable" sta ad indicare che la programmazione avviene "sul campo", ossia da parte dell'utente finale, contrariamente a quei dispositivi le cui funzionalità interne sono prefissate dal produttore (es. processore). Questo consente al progettista di riconfigurare la risorse logiche disponibili per implementare funzioni o sistemi adatti a specifiche applicazioni. I primi FPGA furono introdotti dalla Xilinx nel 1985, con l'arrivo dell'XC2064.

Tali elementi presentano caratteristiche intermedie tra i dispositivi ASIC (*Application Specific Integrated Circuit*) e i precedenti PLD (*Programmable Logic Devices*) come i PAL (*Programmable Array Logic*), GAL (*Generic PAL*) e i CPLD (*Complex Programmable Logic Devices*). Questo perchè la loro funzionalità può essere configurata sul campo, come i PLD, e possono contenere milioni di porte logiche ed essere usati per implementare funzioni estremamente grandi e complesse che, precedentemente, potevano essere realizzate solo usando dispositivi ASIC.

I vantaggi principali sono la velocità di implementazione e il fatto che permettono di apportare modifiche al sistema semplicemente riprogrammando il dispositivo in qualsiasi momento e in pochi secondi [1]. Questo garantisce una notevole flessibilità e una riduzione di costi e tempi di progettazione, che per un ASIC sono molto dispendiosi, oltre al fatto che su questi il sistema

[1] Una recente applicazione è quella di fare svolgere al sistema differenti funzioni in differenti istanti di tempo, ossia la logica è modificata dinamicamente (*Reconfigurable logic*). Invece di avere tutte le funzionalità sempre disponibili, una parte di hardware può essere riconfigurato per implementare le diverse funzioni.



(a) (b)

Figura 1.1: (a) Struttura interna di una FPGA. (b) Schema di una connessione programmabile basata su SRAM, a seconda del valore memorizzato nel registro l'nMOS crea o meno il collegamento tra i due segnali.

è inciso sul silicio e quindi non modificabile. Tuttavia gli FPGA presentano costi non competitivi per applicazioni con elevati volumi di produzione (milioni di pezzi) e la loro generalità si paga con prestazioni inferiori e una minore ottimizzazione delle risorse, contrariamente agli ASIC che permettono di ammortizzare gli elevati costi di progettazione e forniscono prestazioni e livelli di integrazione elevati.

La programmazione degli FPGA (ma anche la progettazione di ASIC) si appoggia a strumenti di sviluppo software (EDA, *Electronic Design Automation*) in grado di eseguire due operazioni principali: la sintesi, cioè la traduzione delle specifiche di progetto in una sua implementazione reale; la simulazione, ossia la verifica che l'implementazione segua le specifiche imposte. Le specifiche possono essere rappresentate attraverso il disegno dello schema progettuale (*schematic capture*), oppure in forma testuale utilizzando un linguaggio di descrizione dell'hardware (HDL) come il VHDL o il Verilog. Quasi tutte le case produttrici di FPGA (ad esempio Xilinx, Altera ed Actel) forniscono versioni gratuite degli strumenti di sviluppo software per la loro programmazione, fornendo anche una serie di strutture logiche pre-progettate (moduli IP) direttamente implementabili al loro interno.

La loro flessibilità e la continua crescita in fatto di prestazioni li rende adatti a molte applicazioni come elettronica di consumo, telecomunicazioni, automotive, DSP, aerospaziale, militare e biomedicale. Inoltre sono spesso utilizzati per la prototipazione (*prototyping*) di dispositivi ASIC o come piattaforme per verificare l'implementazione fisica di nuove funzioni o algoritmi. Gli sviluppi tecnologici e il crescente livello di integrazione hanno portato ad un continuo incremento della quantità di logica integrabile su un singolo chip, tanto da affiancare ai blocchi logici memorie dedicate, moltiplicatori, veloci interfacce di I/O oltre a veri e propri microprocessori embedded, consentendo la realizzazione di interi sistemi su chip (SoC, *System on Chip*) contenenti sia elementi hardware che software.

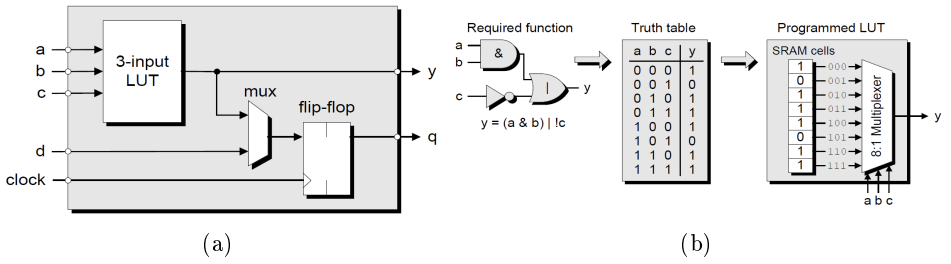


Figura 1.2: Blocco logico base ed esempio di implementazione di una funzione logica con LUT

1.1.2 Struttura interna e programmazione

Un dispositivo FPGA è costituito da un elevato numero di blocchi logici programmabili, circondati da un rete di interconnessioni programmabili. I singolo blocchi logici possono essere configurati per realizzare semplici funzioni e collegati opportunamente tra loro programmando le interconnessioni, realizzando così funzioni più complesse.

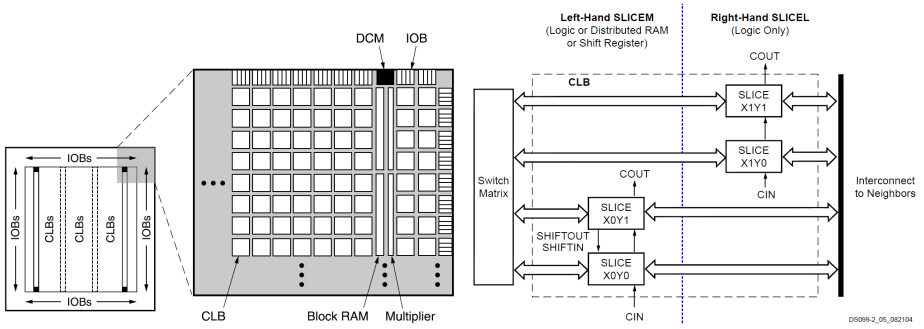
La struttura base di un blocco logico è mostrata in Figura 1.1a. La tabella di verità (LUT, Look Up Table) è composta da celle SRAM, che sono configurate per implementare una generica funzione combinatoria(vedi Figura 1.1b). L'uscita della LUT può essere utilizzata direttamente oppure può essere memorizzata in un flip-flop di tipo D (configurabile anche come latch), implementando così un circuito sequenziale. Inoltre è presente un ingresso ausiliario per combinare più funzioni logiche attraverso un multiplexer (MUX).

Solitamente si utilizzano delle celle SRAM per configurare il componente; la funzione logica svolta da ciascun blocco e i collegamenti tra i terminali di questi, sono determinati commutando degli interruttori programmabili costituiti da un MOSFET comandato da una memoria SRAM. Dato che tale meccanismo è volatile e le memorie perdono il loro contenuto quando il dispositivo viene sconnesso dall'alimentazione, è necessario ricaricare la configurazione (*bistream*[2]) da una memoria esterna non volatile ogni volta che si accende il componente. I dati di programmazione sono trasferiti in modo seriale alla FPGA e una volta caricate tutte le celle di memoria[3], il circuito implementato è pronto per funzionare.

I blocchi logici delle FPGA moderne hanno strutture molto più complesse, inoltre i dispositivi contengono ulteriori risorse le cui funzionalità complementano quelle generali dei blocchi logici, ad esempio memorie, moltiplicatori, interfacce di I/O o addirittura core di processori. Nel prossimo

[2]Il circuito che si vuole implementare viene descritto tramite linguaggio VHDL e tradotto automaticamente in un file di configurazione (bitstream) dal software di sviluppo.

[3]Duurante questa fase si può interpretare la memoria della FPGA come un enorme registro a scorrimento.



(a) (b)

Figura 1.3: (a) Architettura interna di un FPGA Xilinx della famiglia Spartan-3. (b) Architettura interna di un CLB.

paragrafo viene illustrata brevemente la struttura di un dispositivo della famiglia Spartan-3 della Xilinx, poichè su tale FPGA è stato implementato il progetto, argomento di questa tesi.

1.1.3 Xilinx Spartan-3

L'elemento di base di un dispositivo Spartan-3 è la cella logica (LC, *Logic cell*), che ha caratteristiche simili al blocco logico descritto in precedenza. Essa contiene una LUT a quattro ingressi, un flip-flop di tipo D (configurabile anche come latch) e un multiplexer (wide-function multiplexer) in grado di combinare più LUT per realizzare funzioni logiche complesse. In aggiunta, la cella logica contiene un circuito per la logica di riporto (carry circuit) e alcune porte logiche, utilizzati per implementare funzioni aritmetiche in modo efficiente.
Due celle logiche sono unite a formare uno *slice* e quattro slice sono raggruppati a coppie asimmetriche per formare un CLB (*Configurable Logic Block*). I CLB costituiscono la risorsa logica principale della FPGA e non sono altro che i blocchi logici configurabili necessari per implementare circuiti combinatori o sequenziali, collegati tra loro dalla rete di connessioni programmabili. Le due coppie di slice in un CLB appartengono a due colonne diverse, con percorsi di riporto separati. Le LUT della coppia di slice di sinistra (detta SLICEM) possono essere configurate come una memoria RAM statica da 16 bit[4] oppure come shift register da 16 bit. La restante coppia di slice (SLICEL) è dedicata esclusivamente alla realizzazione di funzioni logiche.
Ogni CLB è connesso ai CLB confinanti tramite un bus e contemporaneamente ad una matrice di interconnessione per realizzare collegamenti tra CLB distanti.

[4]Più LUT connesse realizzano una memoria RAM distribuita.

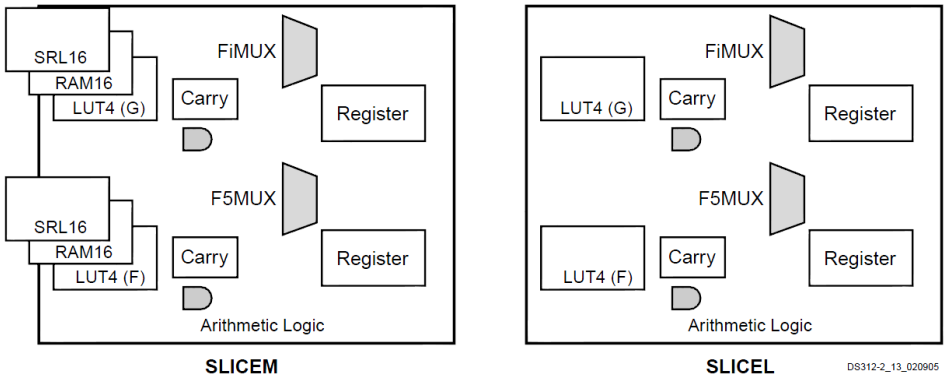


Figura 1.4: Risorse all'interno dei due tipi di slice.

Il dispositivo Spartan-3 contiene ulteriori risorse (macro cells o macro blocks):

- Moltiplicatori in grado di eseguire la moltiplicazione di due numeri da 18 bit;
- Blocchi di RAM da 18 Kbit sincrone utilizzabili in varie configurazioni;
- Blocchi DCM (*Digital Clock Manager*) per la gestione del segnale di clock;
- Blocchi di I/O che controllano il flusso di dati tra la logica interna e i pin esterni del dispositivo e che possono essere configurati per supportare diversi tipi di standard.

Anche se tutti i dispositivi della famiglia Spartan-3 presentano queste caratteristiche, ogni sottofamiglia presenta una diversa quantità di risorse. Nella tabella sono visualizzate le risorse di ogni sottofamiglia ed è inoltre evidenziata quella utilizzata per l'implementazione del progetto (XC3S200).

Device	System Gates	Equivalent Logic Cells[1]	CLB Array (One CLB = Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50[2]	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200[2]	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400[2]	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000[2]	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	633	300
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	633	300

Figura 1.5: Elenco dei dispositivi della famiglia Spartan-3 e sommario delle loro risorse.

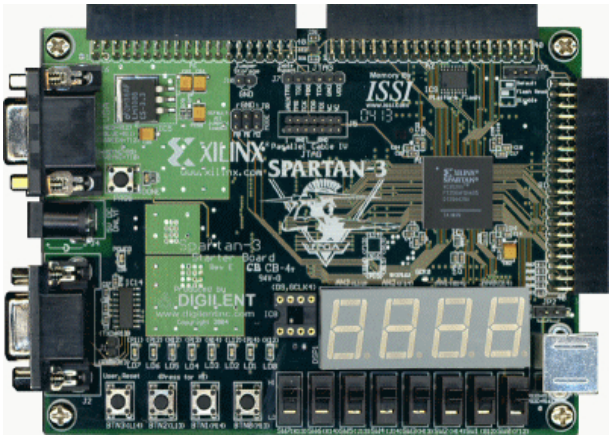


Figura 1.6: Scheda Digilent S3SKB

1.2 S3SKB Board

Per il progetto è stata utilizzata la scheda Digilent S3SKB (Spartan-3 Starter Kit Board), una piattaforma di sviluppo a basso costo e di facile utilizzo per l'implementazione di progetti su dispositivi Spartan-3. Essa monta un FPGA Xiliinx XC3S200 in un package FT256 oltre ad una serie di componenti preassemblati e collegati all'FPGA [2]:

- Memoria Flash Xilinx XCF02S da 2Mbit;
- Due moduli di SRAM asincroni 256k x 16 (ISSI IS61LV25616AL-10T);
- Porta VGA 8 colori;
- Porta seriale RS-232 con traslatore di livello;
- Porta PS/2;
- 4 display LED a 7 segmenti;
- 8 switch;
- 8 LED;
- 4 pulsanti;
- Oscillatore al quarzo da 50 MHz come sorgente di clock;
- 3 connettori di espansione da 40 pin;
- Porta JTAG per la programmazione.

Nel progetto, oltre ovviamente all'FPGA, verranno utilizzati i LED e i display per visualizzare lo stato del sistema, gli switch come ingressi del sistema, la porta RS-232 per la comunicazione con il PC e i connettori di espansione per l'uscita del sistema.
La programmazione è effettuata tramite la l'interfaccia JTAG ed è risultato utile salvare i dati di configurazione nella memoria flash da 2 Mbit (Xilinx XCF02S) onde evitare di riconfigurare il sistema tramite JTAG ad ogni accensione della board.

Capitolo 2

Sintesi Digitale Diretta

2.1 Introduzione

La sintesi digitale diretta (DDS, *Direct Digital Synthesis*) è una tecnica che utilizza dei blocchi di elaborazione digitale per generare un segnale analogico sintonizzabile in frequenza e fase a partire da un segnale di clock di riferimento a frequenza fissa (solitamente generato da un oscillatore al quarzo). In pratica, l'architettura del sintetizzatore digitale permette di dividere la frequenza del clock di riferimento per un fattore di scala rappresentato da una parola binaria programmabile detta FTW (*Frequency Tuning Word*). Tale parola è tipicamente formata da un numero di bit che varia da 24 a 48 e la sua lunghezza determina la risoluzione in frequenza del sintetizzatore.

I dispositivi che utilizzano la tecnica DDS stanno diventando velocemente una alternativa alle soluzioni di sintesi analogica di frequenza, grazie alla loro alta capacità di integrazione (l'architettura può essere integrata in package molto piccoli), alte prestazioni e costi competitivi. Inoltre il consumo di potenza è molto basso e l'abilità di generare forme d'onda con un'alta risoluzione e accuratezza, oltre alla capacità di essere programmate (e riprogrammate) digitalmente, ne fanno una soluzione allettante in applicazioni industriali e di comunicazione. La possibilità di integrare su un singolo chip convertitori digitale-analogico molto veloci e architetture DDS (formando quello che viene chiamato *Complete DDS Solution*), permette a questa tecnologia di puntare ad un largo gruppo di applicazioni e fornisce, in molti casi, un'alternativa attraente ai sintetizzatori analogici basati su PLL (*Phase Locked Loop*). Per molte applicazioni, la soluzione DDS presenta molti vantaggi rispetto ad un sintetizzatore che impiega un circuito PLL:

- Una risoluzione in frequenza dell'ordine del micro-Hertz e una accuratezza nella fase al di sotto del grado, con un completo controllo digitale.

- Un'elevata velocità nella variazione della frequenza (o fase) di uscita in quanto il sistema non richiede nessun tempo di assestamento nel cambiare la frequenza (o fase), infatti si va sull'ordine dei nanosecondi o poche decine di microsecondi.

- I cambiamenti di frequenza sono accompagnati da una variazione continua della fase, in questo modo la nuova frequenza generata riprende esattamente dall'ultimo valore di fase della frequenza precedente.

- L'architettura digitale della DDS elimina la necessità di una sintonizzazione manuale del sistema e di aggiustamenti dovuti all'invecchiamento dei componenti e alla deriva termica[1], caratteristiche dei sintetizzatori analogici.

- L'interfaccia digitale di controllo dell'architettura DDS favorisce un ambiente in cui i sistemi possono essere controllati da remoto e ottimizzati sotto il controllo di un processore.

Tuttavia, come si vedrà più avanti, un sistema DDS presenta anche degli svantaggi:

- La massima frequenza di uscita è minore della frequenza del clock di riferimento

- La generazione digitale di una forma d'onda sinusoidale produce distorsione, l'onda presenta una spettro non puro.

- La purezza spettrale è determinata principalmente dal DAC (Ad alte velocità le proprietà di linearità di un DAC non sono buone).

Poichè il segnale generato è in forma digitale, risulta semplice includere differenti funzionalità di modulazione utilizzando tecniche digitali di elaborazione del segnale. In particolare la variazione veloce e precisa della frequenza e della fase rende il sistema adatto all'implementazione di modulazioni di fase (PSK, *Phase Shift Keying*) e di frequenza (FSK, *Frequency Shift Keying*).

Attualmente le applicazioni che utilizzano la tecnica DDS per generare forme d'onda ricadono nei campi delle comunicazioni, dell'industria e della biomedicina (o bioingegneria). I sistemi di comunicazione o radar richiedono delle sorgenti di frequenza veloci per applicazioni di modulazione e codifica. Nel campo dell'analisi dei segnali, molti sistemi industriali e biomedicali utilizzano la DDS per generare forme d'onda programmabili a frequenza e fase facilmente regolabili, senza il bisogno di cambiare i componenti esterni, come

[1] La deriva termica è il fenomeno che introduce un errore nella misura di un segnale. Questo errore è più o meno grande quanto più o meno i dispositivi elettronici sono sensibili agli effetti della temperatura. I circuiti digitali utilizzati per implementare funzioni di signal-processing non soffrono di tale effetto.

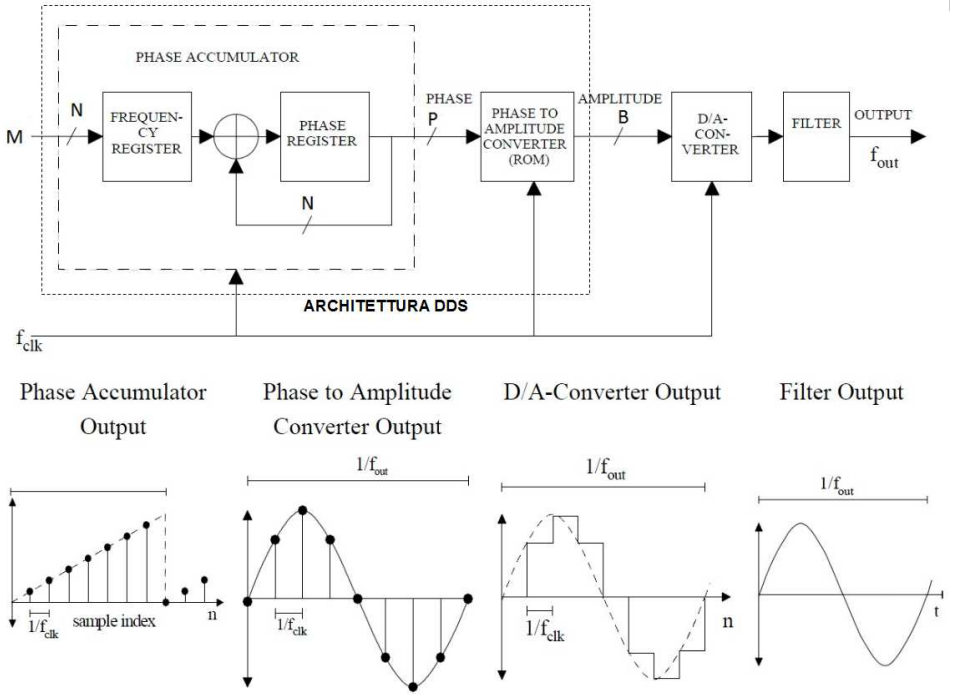


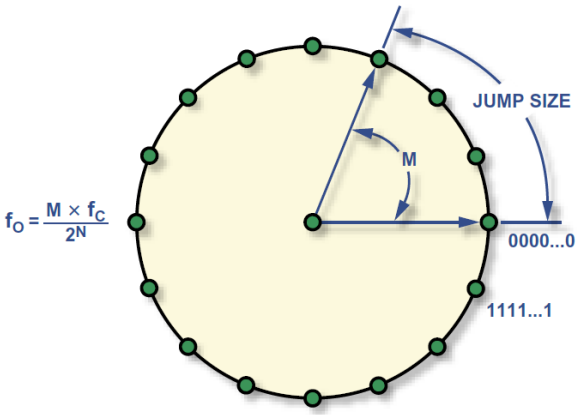
Figura 2.1: Architettura completa di un sistema DDS. Sono illustrati anche i segnali in uscita dai vari blocchi.

solitamente accade per i generatori analogici tradizionali. Alcune applicazioni utilizzano tale tecnica come sorgente di frequenza configurabile per la misura di impedenza (ad esempio l'impedenza di un sensore), per generare segnale modulati per microattuatori o per esaminare l'attenuazione in cavi telefonici o LAN.

2.2 Architettura di base

Lo scopo del sistema è generare un segnale periodico, a tempo discreto ad una determina frequenza f_{out}. Solitamente la forma d'onda è un seno ma il sistema può essere utilizzato per generare una qualsiasi forma d'onda periodica.
Un'architettura DDS è composta da due componenti principali: un accumulatore di fase (*Phase accomulator*) e un covertitore fase-ampiezza (*Phase to amplitude converter*) (Figura 2.1).
L'accumulatore di fase è sostanzialmente un contatore ad N bit con incremento variabile. Ad ogni fronte di salita del clock, il valore dell'accumulatore, memorizzato in un registro di fase (*phase register*), viene incrementato di una quantità intera (M) definita da una parola binaria memorizzata in un registro di frequenza (*frequency register*). Il convertitore fase-ampiezza è

costituito solitamente da una PROM (o LUT, *Look Up Table*) in cui sono contenuti i valori di ampiezza dell'onda sinusoidale corrispondenti ad un periodo completo della sinusoide.
La parola in uscita all'accumulatore rappresenta un determinato valore di fase dell'onda sinusoidale, inviato al convertitore fase-ampiezza il quale da in uscita il valore codificato di ampiezza corrispondente. Per avere un segnale analogico è necessario aggiungere un convertitore digitale-analogico (DAC), formando quello che viene chiamato *Complete DDS Solution*, di modo da convertire il valore digitale in un corrispondente valore analogico di tensione o corrente. Infine viene posto un filtro che elimina le componenti ad alta frequenza fornendo in uscita un'onda sinusoidale pura.
La frequenza di uscita dipende da due variabili: la frequenza del clock di riferimento e la parola binaria programmata nel frequency register (FTW, *Frequency Tuning Word*).



n	NUMBER OF POINTS
8	256
12	4096
16	65535
20	1048576
24	16777216
28	268435456
32	4294967296
48	281474976710656

Figura 2.2: Funzionamento di un sistema DDS illustrato sulla ruota di fase

Per comprendere il funzionamento possiamo visualizzare l'oscillazione della sinusoide come un vettore che ruota attorno ad un cerchio di fase. I segnali sinusoidali a tempo continuo presentano una fase (o angolo) che varia linearmente da 0 a 2π. Ogni punto sul cerchio di fase corrisponde all'equivalente punto su un periodo della sinusoide. Il vettore, ruotando a velocità costante, descrive in una rivoluzione completa un periodo completo della sinusoide d'uscita. L'implementazione digitale non è diversa. L'accumulatore di fase fornisce un numero discreto di valori di fase equispaziati che accompagnano la rotazione lineare del vettore sulla "routa di fase" (*phase*

wheel). I valori binari corrispondono ai punti di un periodo della sinusoide e il loro numero è determinato dalla dimensione del contatore (N). L'uscita dell'accumulatore di fase però è lineare e non può essere direttamente utilizzata per generare una sinusoide o qualsiasi altra forma d'onda periodica, eccetto una rampa. Perciò è necessario utilizzare un convertitore fase-ampiezza per mappare il valore lineare della fase in un corrispondente valore di ampiezza del segnale.

Ad ogni impulso di clock l'accumulatore di fase compie un salto di fase determinato dalla *tuning word* (M) che impone quanti punti saltare nella ruota delle fasi. Più grande è il salto, più velocemente l'accumulatore và in overflow completando una rivoluzione, equivalente ad un periodo della sinusoide. La velocità di rivoluzione del vettore, o equivalentemente di overflow del contatore, è direttamente legata con la frequenza della forma d'onda in uscita al sistema. Questo ci consente di ricavare una relazione che a partire dalla frequenza di uscita ci permette di calcolare il valore del salto di fase necessario, quindi il valore della *tuning word* da impostare nel *frequency register*.

Matematicamente una forma d'onda sinusoidale è espressa come

$$x(t) = \sin(\omega t)$$

La fase ωt dipende linearmente dal tempo e la pulsazione (o velocità angolare) ω dipende linearmente della frequenza f dalla relazione

$$\omega = 2\pi f$$

Per un dato intervallo di tempo di riferimento δt la variazione di fase è pari a

$$\Delta phase = \omega \delta t,$$

da cui si ricava

$$\omega = \frac{\Delta phase}{\delta t} = 2\pi f \quad \Rightarrow \quad f = \frac{\Delta phase}{2\pi \delta t}$$

Nel caso della sintesi DDS l'intervallo di tempo di riferimento è il periodo del clock di riferimento

$$\delta t = T_{clk} = 1/f_{clk}$$

Inoltre, la variazione di fase non è altro che la *tuning word* M e un periodo intero della sinusoide è rappresentato dal numero di punti nella ruota di fase, cioè $2\pi = 2^N$. Per cui si ottiene :

$$f_{out} = \frac{M \cdot f_{clk}}{2^N}$$

Questa relazione è l'equazione base di sintonizzazione di un sistema DDS, dove:

- f_{out} = frequenza di uscita
- M = valore della parola di sintonizzazione (*frequency tuning word*)
- f_{clk} = frequenza del clock di riferimento
- N = numero di bit dell'accumulatore di fase

Un cambiamento del valore di M nell'architettura DDS provoca un immediato cambiamento della frequenza di uscita mantenendo la continuità della fase.
Imponendo $M = 1$ è possibile ricavare la risoluzione in frequenza del sintetizzatore, cioè il più piccolo salto di frequenza che il DDS può compiere. Tale valore coincide anche con la minima frequenza sintonizzabile, dato che si impone il minimo salto di fase :

$$f_{out_{min}} = \frac{f_{clk}}{2^N}$$

Da questa equazione si può notare l'accuratezza raggiungibile da un sistema DDS. Ad esempio un sistema con risoluzione $N = 48$ e frequenza di riferimento $f_{clk} = 300$ MHz, presenta una risoluzione in frequenza pari a 1μHz.

Se la fase viene codificata con N bit, ad ogni valore deve essere associato il corrispettivo di ampiezza del segnale, perciò la PROM dovrà avere 2^N indirizzi. Risulta evidente che utilizzare un accumulatore con una risoluzione che varia dai 24 ai 48 bit, impone di dover utilizzare una ROM di dimensioni troppo grandi per una implementazione realistica ed efficiente. Quindi solitamente si effettua un "troncamento di fase", ossia si utilizzano solo i P bit più significativi (con $P \leq N$) come informazione per la fase, riducendo le dimensioni della ROM a 2^P indirizzi. In questo modo si mantiene l'elevata risoluzione in frequenza ma viene aggiunto un errore di fase (*phase noise*) generalmente accettabile.

Bisogna inoltre ricordare che questo sistema è digitale e il segnale in uscita dal convertitore fase-ampiezza è a tempo discreto, quindi se la frequenza di uscita aumenta, il numero di campioni in un periodo diminuisce. Il teorema del campionamento richiede che ci siano almeno due campioni per periodo per ricostruire esattamente il segnale, cioè deve essere:

$$f_c \geq 2 f_{out}$$

Poichè nel nostro caso la frequenza di campionamento è la frequenza del segnale di riferimento f_{clk}, la massima frequenza di uscita teoricamente sintetizzabile in un sistema DDS è pari a

$$f_{out_{max}} = \frac{f_{clk}}{2}$$

Nella pratica la frequenza massima è circa il 40% della frequenza di clock, di modo da migliorare la qualità del segnale ricostruito e permettere il filtraggio del segnale in uscita al DAC.

2.3 Studio dello spettro del segnale di uscita

Per capire come è fatto lo spettro del segnale in uscita, bisogna far ricorso alla teoria del campionamento. La teoria di Nyquist indica che sono necessari almeno due campioni per ciclo per ricostruire la forma d'onda desiderata in uscita. Se consideriamo che il segnale in uscita al convertitore fase-ampiezza è un segnale discreto, campionato alla frequenza f_{clk}, allora il singolo spettro della forma d'onda voluta viene replicato e centrato in multipli della frequenza di campionamento. Inoltre, poichè il DAC utilizza un metodo di tipo Sample & Hold, tali repliche vengono poi pesate per la funzione $\sin(x)/x$ che si annulla per valori interi multipli della frequenza di campionamento. Lo spettro del segnale all'uscita del DAC è visibile in figura 2.3.

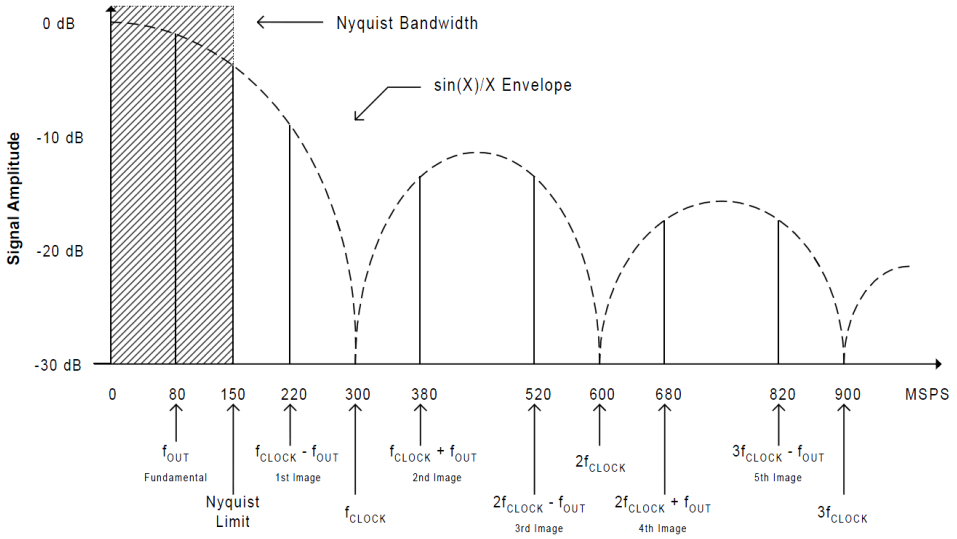


Figura 2.3: Spettro di uscita del segnale campionato

Le immagini si trovano alle frequenze $nf_{clk} \pm f_{out}$ con $n = 0, 1, ...$etc. L'ampiezza della forma d'onda in uscita normalizzata è pari a

$$A(f_o) = \frac{\sin(\frac{\pi f_o}{f_{clk}})}{\frac{\pi f_o}{f_{clk}}}$$

Il filtro di antialiasing, oltre ad eliminare le immagini del segnale, deve essere progettato in modo da compensare l'effetto di rolloff della funzione $\sin(x)/x$. Per facilitare la progettazione e migliorare la purezza del segnale è ragionevole limitare la frequenza di uscita al 40% della frequenza di campionamento, in modo da realizzare filtri con costi competitivi.

In aggiunta al problema delle immagini, altri fattori degradano la purezza spettrale del segnale. Ulteriori componenti spettrali indesiderate (*spurie*) sono dovute principalmente ad effetti di quantizzazione, in particolare:

- Effetti dovuti alla risoluzione del convertitore D/A
- Effetti dovuti al troncamento di fase

Queste anomalie non sono eliminabili tramite il filtro poichè ricadono nella banda di interesse del segnale, tuttavia hanno un'ampiezza che è generalmente molto minore rispetto alla fondamentale e alle sue immagini.

2.3.1 Effetti della risoluzione del convertitore D/A

La risoluzione di un DAC (*Digital to Anologic Converter*) è specificata dal numero dei suoi bits (B-Bits) in ingresso. Guardando la ricostruzione di un'onda sinusoidale (Figura 2.4) si capisce facilmente che importanza abbia la risoluzione di un DAC.

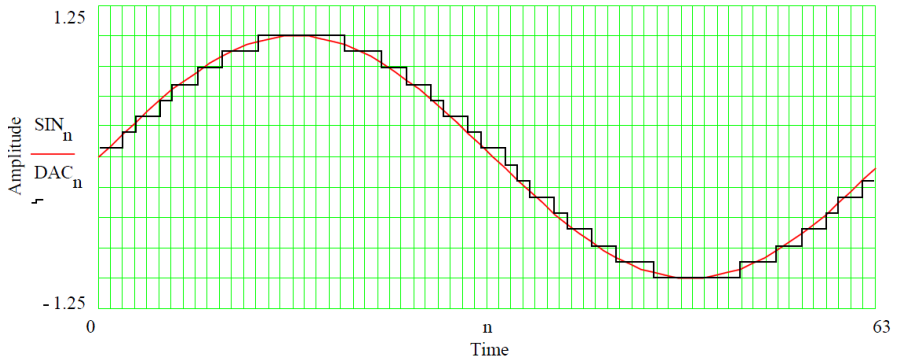


Figura 2.4: Effetto della risoluzione del DAC

Come si vede dalla figura, un DAC a 4 bit di risoluzione è utilizzato per ricostruire una forma d'onda sinusoidale perfetta. Le linee verticali sono gli istanti di tempo durante i quali l'uscita del DAC viene aggiornata. Le distanze orizzontali tra le righe verticali rappresentano il periodo di campionamento. La distanza verticale tra due righe orizzontali è proprio l'errore introdotto dal DAC e dipende dalla risoluzione di quest'ultimo. Questo errore è detto *errore di quantizzazione* e dà origine ad un effetto noto come *distorsione da quantizzazione*.
Per comprendere la distorsione da quantizzazione dobbiamo guardare le rapide variazioni del DAC, queste brusche variazioni implicano la presenza di componenti ad alta frequenza sulla fondamentale. Sono queste frequenze che costituiscono la distorsione dovuta all'errore di quantizzazione per cui nel dominio della frequenza appaiono come spurie discrete interne alla banda di Nyquist nello spettro di uscita del DAC.

È evidente che aumentando la risoluzione l'errore di quantizzazione diminuisce, quindi diminuisce la distorsione di quantizzazione e le componenti spurie nello spettro all'uscita del DAC diminuiscono (Figura 2.5).

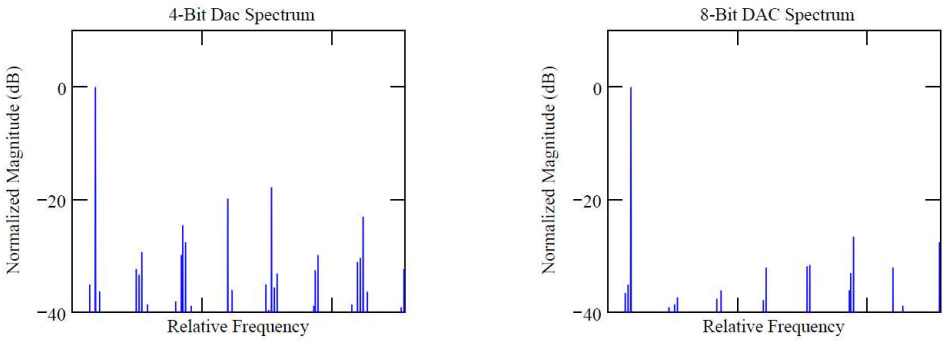


Figura 2.5: Spettro di uscita di un DAC da 4 bits contro uno da 8 bits.

La relazione tra la risoluzione del DAC e l'errore di quantizzazione è quantificabile tramite il parametro SQR (*Signal power to Quantization noise power Ratio*) definito come:

$$SQR_{dB} = 1.76 + 6.02B$$

Dove B rappresenta il numero dei bit del DAC.
Per esempio, un DAC ad 8 bit fornisce un SQR pari a 49.92 dB. Da notare che l'equazione specifica solo il rapporto delle potenze di segnale e di rumore di quantizzazione e non fornisce nessuna informazione sulla distribuzione delle spurie o sul massimo livello di spurie. Come si nota in figura 2.5, avere più bit a disposizione comporta un netto miglioramento sulla qualità dello spettro di uscita. I livelli delle spurie dovuto all'errore di quantizzazione sono più bassi se si aumenta il numero di bit. Il valore del SQR visto prima è valido solo se il DAC opera in fondo scala.

Un effetto che produce il miglioramento di tale valore è dovuto al sovra-campionamento. Ricordando che Nyquist richiede che la banda del segnale campionato deve essere al massimo 1/2 della frequenza di campionamento, in questo caso la banda del segnale campionato viene intenzionalmente ridotta ad una frazione della banda di Nyquist richiesta.

La quantità di potenza di rumore di quantizzazione dipende dalla risoluzione del DAC. Inoltre è una quantità fissa e proporzionale all'area più scura nella figura 2.6. In entrambi i casi (campionando a Nyquist o sovra-campionando), la quantità di potenza di rumore di quantizzazione è la stessa (le aree sono uguali). Poichè la potenza di rumore è la stessa in entrambi i casi e le aree dei due rettangoli di rumore sono proporzionali alla potenza di rumore, allora l'altezza del rettangolo di rumore nel caso sovra-campionato deve essere minore del caso di campionamento alla Nyquist in modo tale che le aree coincidano. Questo fa si che nella banda di interesse, per una data quantità di potenza del segnale, il rapporto delle potenze segnale-rumore sia più grande nel caso di sovra-campionamento.

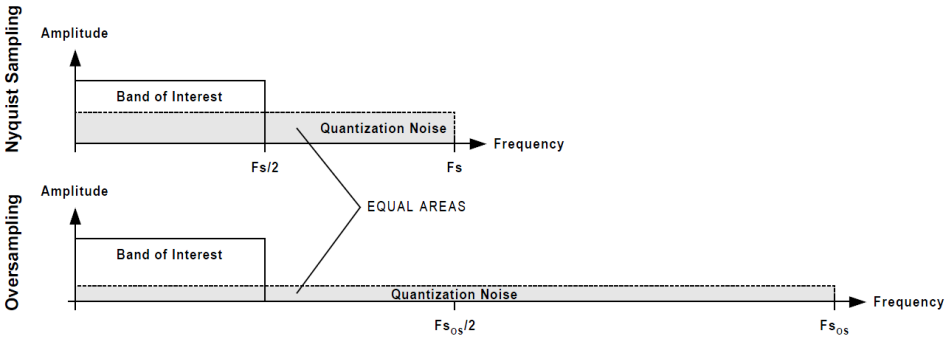


Figura 2.6: Effetto del sovra-campionamento sul SQR

L'effetto del sovra-campionamento è quantificabile ed è dato da:

$$C = 10\log(F_{S_{OS}}/F_S)$$

dove F_S è la frequenza di campionamento alla Nyquist mentre $F_{S_{OS}}$ è la frequenza di sovra-campionamento. Il nuovo valore dell'SQR sarà

$$SQR_{dB} = 1.76 + 6.02B + 10\log(F_{S_{OS}}/F_S)$$

L'effetto di sovra-campionamento può compensare la diminuzione di SQR che si avrebbe se il DAC opera al di sotto del fondo scala.

2.3.2 Effetti del troncamento di fase

Come detto in precedenza,l'impossibilità di realizzare una LUT di grandi dimensioni rende necessario il troncamento dell'informazione di fase in uscita dall'accumulatore. Ciò si ripercuote nella presenza di spurie all'interno dello spettro d'uscita. Per comprendere bene tale fenomeno è necessario ritornare a fare riferimento alla ruota di fase digitale.

Supponiamo di avere un DDS che utilizza un accumulatore ad 8-bit di cui vengono utilizzati solo i 5 più significativi per mappare la LUT. La ruota di fase è visibile in figura 2.7. Con un accumulatore ad 8-bit la risoluzione di fase è 1/256 di un giro completo ovvero 1.41° $(360/2^8)$. Nella figura tale risoluzione è identificata dai punti del cerchio esterno. Se si utilizzano solo i primi 5 bit più significativi la risoluzione è identificata dai punti del cerchio interno dunque è pari a 1/32 di giro cioè $(360/2^5)$.
Assumiamo che la *tuning word* valga 6. Ciò significa che ad ogni colpo di clock l'accumulatore si incrementa di 6 alla volta. Al primo colpo di clock non si riesce a raggiungere il primo punto del cerchio interno, formano una discrepanza tra la fase determinata dall'accumulatore di fase (cerchio esterno) e quella espressa con la parola troncata a 5 bit (cerchio interno). Questa discrepanza porta ad un errore di fase pari a 8.46°$(6x1.41°)$, come indicato dall'arco E1. Al secondo colpo di clock la fase dell'accumulatore si

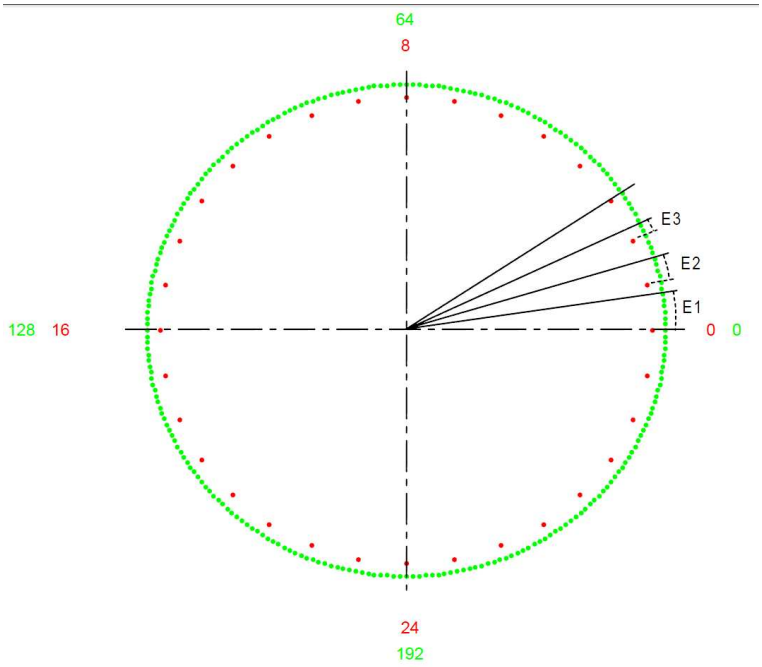


Figura 2.7: Visualizzazione dell'errore di fase nella ruota di fase

incrementa di 6 ricadendo tra il primo ed il secondo punto del cerchio interno. Ancora una volta c'è una differenza tra la fase dell'accumulatore e la reale fase (risoluzione 5-bit). L'errore vale $5.64°(4x1.41°)$ come mostrato dall'arco E2. Al terzo passo l'errore vale 2.82°. Al quarto le 2 fasi coincideranno senza errore di fase. A questo punto tale pattern si ripete.
Chiaramente, l'errore di fase si ripercuote sull'errore di ampiezza durante la conversione nella LUT. Questo errore è periodico in quanto, senza tener conto della *tuning word* scelta, dopo un numero sufficiente di giri sulla ruota di fase il valore troncato e quello reale coincideranno. Essendo periodici nel dominio del tempo, si presentano come spurie spettrali nel dominio della frequenza conosciute come *Phase Truncation Spurs*.

Si dimostra che ampiezza e distribuzione delle spurie dovute al troncamento dipendono da tre fattori:

1. Precisione dell'accumulatore (**N** bits)

2. Precisione della parola di fase (**P** bits), cioè il numero di bit di fase dopo il troncamento

3. Tuning word (**M**)

Ampiezza delle spurie

Alcune tuning words non determinano spurie dovute al troncamento di fase,mentre le restanti producono spurie del massimo livello possibile. Se la quantità N-P è maggiore o uguale a 4 allora il livello massimo di ampiezza delle spurie è approssimato a $-6.02P$ dBc (cioè $6.02P$ dB al di sotto del livello della fondamentale a frequenza f_{out}). Per cui sperimentalmente gli errori nello spettro risultano accettabili. Se ho un DDS a 32 NBits che utilizza 12 PBit il livello delle spurie è circa -72 dBc rispetto al valore della fondamentale.

Le tuning words che portano ad un livello massimo di spurie sono quelle che soddisfano la seguente equazione:

$$GCD(M, 2^{(N-P)}) = 2^{(N-P-1)}$$

dove $GCD(X, Y)$ è il massimo comune divisore tra X e Y. Affinché la relazione sia soddisfatta le tuning word devono avere la forma mostrata in figura 2.8.

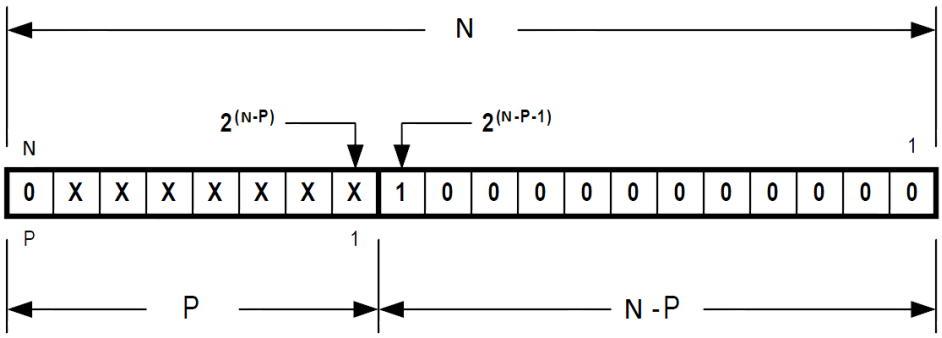


Figura 2.8: Tuning word che produce il massimo livello di spurie.

I primi P bit sono quelli utilizzati per l'informazione di fase (*Phase Word*). I restanti N-P bits vengono quindi ignorati. La FTW è composta da N-1 bits (questo perché il primo bit, ossia il *Most Significative Bit* MSB, deve essere 0 per poter rispettare la condizione di Nyquist e quindi risolvere il problema di aliasing). Come mostrato in figura qualunque tuning word che presenta un bit posto a 1 nella posizione $2^{(N-P-1)}$ e zero in tutte le restanti posizioni meno significative produrà il caso peggiore di livello di spurie di troncamento di fase (-6.02 dBc).

Dall'altro lato, le tuning words che non producono alcuna spuria dovuta al troncamento di fase soddisfano la seguente equazione:

$$GCD(M, 2^{(N-P)}) = 2^{(N-P)}$$

Come mostrato in figura 2.9 queste parole presentano un bit ad 1 nella posizione $2^{(N-P)}$ e 0 nei restanti bit che vengono troncati, ma essendo zeri non

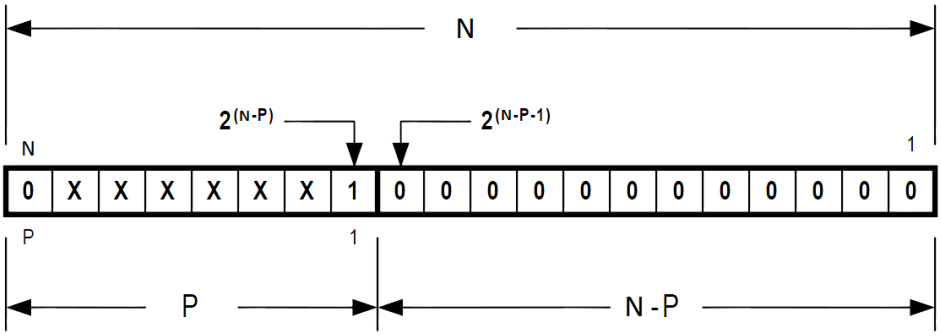


Figura 2.9: Tuning word che non produce spurie.

portano informazione aggiuntiva per cui non si ha perdita di dati.
Tutte le altre tuning words che non rientrano nei due casi precedenti produrranno un livello di spurie tra i due estremi (per l'analisi della distribuzione delle spurie si veda [7]).

2.4 Spurious Free Dynamic Range (SFDR)

Il rapporto (misurato in dB) tra l'ampiezza della componente fondamentale (f_{out}) e l'ampiezza più grande delle componenti spurie presenti nello spettro è detto *Spurious Free Dynamic Range* (SFDR) ed è un parametro molto importante per un sistema DDS. Una formula semplificata che stima il massimo valore delle spurie quando il livello della fondamentale è 0 dB è[9]:

$$S_{max} = SFDR = -6.02P + 3.92 \quad dBc$$

Ricordiamo però che il convertitore D/A aggiunge una componente di rumore che è quantificabile da un rapporto segnale-rumore approssimato a :

$$SNR = -6.02B - 1.76 \quad dBc$$

Tipicamente in un sistema DDS gli errori presenti all'uscita sono determinati principalmente dalle prestazioni del convertitore D/A. Se si impone che:

$$P = B + 1$$

allora si ha che $S_{max} < SNR$. Ciò significa che l'errore dovuto al troncamento della fase è minore di quello causato dalla risoluzione del DAC.

Per incrementare il valore dell'SFDR il modo più semplice è incrementare la lunghezza della parola di fase P (e B di conseguenza). L'unico svantaggio è che la quantità di memoria utilizzata è pari a 2^Px B bit, quindi per alti valori di P i requisiti di memoria risultano eccessivi per lavorare ad alte frequenze o per implementazioni embedded.

Per raggirare tale inconveniente, si utilizzano soluzioni di compressione dell'onda sinusoidale, riducendo così il consumo di memoria. Una tecnica consiste nel sfruttare la simmetria del seno e memorizzare solamente un quarto della forma d'onda. Questo approccio utilizza una ROM con 2^{P-2} indirizzi, con l'aggiunta di altre componenti hardware per la ricostruzione dell'onda. In alternativa vi sono tecniche algoritmiche che approssimano la funzione. Queste tecniche riducono di molto le dimensioni della ROM, tuttavia producono degli errori che incrementano il livello di spurie nello spettro del segnale.

Capitolo 3

DDS Core

Il generatore di forme d'onda realizzato è in grado di creare tre tipi di forme d'onda: onda sinusoidale, onda triangolare e onda quadra. Inoltre è possibile programmare la frequenza di uscita e la selezione dell'onda tramite comunicazione seriale. In questo capitolo verrà esaminato il nucleo del progetto, costituito da un sintetizzatore digitale che utilizza la tecnica di sintesi digitale diretta (DDS) per la generazione delle singole forme d'onda.

Come è stato spiegato nel capitolo precedente, un sistema DDS è formato da due blocchi principali: un accumulatore di fase (*phase accumulator*) e un convertitore fase-ampiezza (*phase to amplitude converter*). Il sistema realizzato presenta le seguenti caratteristiche:

- N = 28 bits, precisione dell'accumulatore;
- P = 12 bits, precisione della parola di fase;
- B = 10 bits, risoluzione in ampiezza;
- f_{clk} = 50 MHz, frequenza del clock di riferimento.

Nei prossimi paragrafi verranno descritti i singoli blocchi implementati e il loro funzionamento.

3.1 Accumulatore di fase

L'accumulatore contiene al suo interno due registri da N bit: `ftw_reg`, che rappresenta il registro di frequenza e `phase_reg`, che rappresenta il registro di fase e quindi l'uscita del blocco (`phase_out`)(vedi Figura 3.1). Il parametro N è generico in modo che quando viene istanziato il componente è possibile impostare il numero di bit dei registri. In ingresso, oltre al segnale di clock `clk`, è presente un segnale di controllo `rd_tick` e un segnale di `reset`. Se il segnale di reset è alto entrambi i registri vengono azzerati in modo asincrono.

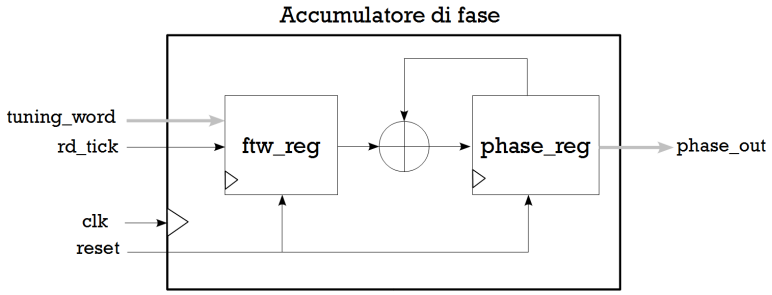


Figura 3.1: Schema a blocchi dell'accumulatore implementato.

Ad ogni fronte di salita del segnale di clock, il valore contenuto nel registro di fase viene incrementato del valore contenuto nel registro di frequenza. Nello stesso istante, se il segnale di controllo è alto significa che all'ingresso `tuning_word` dell'accumulatore è presente una nuova parola di sintonizzazione che viene caricata nel registro di frequenza. In questo modo al successivo fronte di salita del clock si ha una variazione del salto di fase e quindi la variazione della frequenza di uscita.
Il segnale di controllo viene generato da un altro blocco circuitale (word_ control) che verrà descritto nel prossimo capitolo. Questo blocco ha il compito di gestire gli ingressi del DDS core in base alla parola ricevuta dalla porta seriale.
Nel progetto si è utilizzato un accumulatore da 28 bit che unito ad un segnale di clock da 50 MHz permette di ottenere una risoluzione in frequenza di 0.18 Hz e una frequenza massima di 20 MHz per l'onda sinusoidale. Se si utilizza un oscillatore esterno da 25 MHz si può ottenere una risoluzione di circa 0.1 Hz e una massima frequenza di 10 MHz.
Poichè viene effettato un troncamento della fase, verranno utilizzati solo i P bits più significati degli N utilizzati dall'accumulatore.

```
process (clk, reset)
begin
  if reset = '1' then
     phase_reg <= (others => '0');
     ftw_reg <= (others => '0');
  elsif rising_edge(clk) then
     phase_reg <= phase_reg + ftw_reg;
     if rd_tick = '1' then
        ftw_reg <= tuning_word;
     end if;
  end if;
end process;

--output
phase_out <= phase_reg;
```

Listing 3.1: Codice VHDL dell'accumulatore di fase.

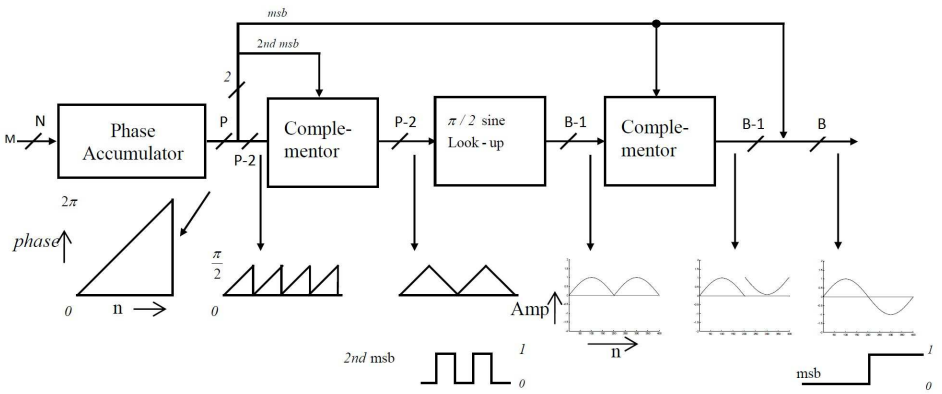


Figura 3.2: Schema a blocchi relativo alla tecnica di compressione della ROM.

3.2 Convertitore fase-ampiezza

La risoluzione dei valori inseriti nella ROM influisce sulla purezza spettrale del sistema DDS, perciò è desiderabile incrementare la risoluzione della ROM. Tuttavia, aumentare le dimensioni della ROM significa avere una maggiore occupazione di area oltre ad un maggiore consumo di potenza e una bassa velocità.
Per ovviare a questo problema, il sistema realizzato sfrutta la proprietà di simmetria dell'onda sinusoidale conservando solamente i valori relativi al primo quadrante $(0 - \pi/2)$, riducendo così la dimensione della ROM di 4 volte. Questa tecnica non introduce errori, l'unico prezzo da pagare è l'introduzione di componenti hardware per ricostruire l'intero periodo dell'onda.

Lo schema a blocchi è visibile in figura 3.2. I due bit più significativi della parola di fase sono utilizzati per decodificare il quadrante dell'onda, mentre i restanti $P-2$ bits sono utilizzati per indirizzare i valori della ROM. Il bit più significativo (MSB) determina il segno, mentre il secondo bit più significativo determina se l'ampiezza è crescente o decrescente. Per avere una corretta simmetria è necessario aggiungere a tutti i campioni del seno un offset di fase pari a 1/2 LSB, ossia $\pi/2^P$ (vedi figura 3.3).
Ne segue che i $P-2$ bits non vengono modificati per il primo e terzo quadrante, mentre devono essere complementati per il secondo e quarto quadrante di modo che la rampa sia invertita e la ROM venga letta dal basso verso l'alto dando in uscita l'andamento descrescente dell'onda. All'uscita della ROM si ottiene una versione raddrizzata dell'onda sinusoidale. Tale onda ha un ampiezza che è metà dell'onda sinusoidale, quindi bastano $B-1$ bits per codificare i valori di ampiezza, riducendo ulteriormente l'occupazione di memoria. I valori di ampiezza corrispondenti al terzo e quarto quadrante (MSB = 1) vengono complementati di modo da ribaltare la semi-onda negativa e infine l'MSB viene concatenato alle parole in uscita determinando il

segno delle due semi-onde.
Il sistema realizzato considera di utilizzare un DAC unipolare, quindi l'MSB concatenato ai valori in uscita dalla ROM deve essere negato.

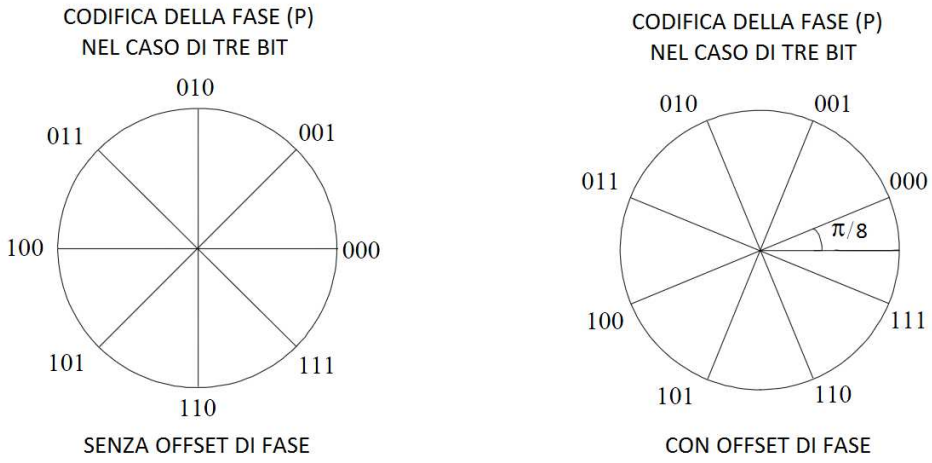


Figura 3.3: Codifica della fase se si aggiunge un offset pari ad 1/2 LSB.

3.2.1 ROM

Per la memorizzazione dei valori della sinusoide si è scelto di utilizzare i blocchi di RAM preinstallati all'interno dell'FPGA, i quali possono essere configurati come ROM. Questo perchè per quantità di dati consistenti, i blocchi di RAM sono più efficienti rispetto all'utilizzo di RAM distribuita (cioè le LUT).
Il dispositivo XC3S200 della famiglia Spartan-3 dispone di 12 blocchi di RAM da 18 Kbit sincrone. Per facilitare l'implementazione del blocco circuitale è stato utilizzato il software *CORE Generator* della Xilinx per creare il modulo IP[1]. Il blocco di memoria è stato configurato come una memoria ROM *Single Port* sincrona con $P - 2 = 10$ bits di indirizzamento e $B - 1 = 9$ bits di risoluzione. Inoltre è stato aggiunto un ingresso di enable, che verrà utilizzato per inibire il funzionamento della ROM quando non è necessaria. Con uno *script* in Matlab sono stati calcolati i campioni dell'onda sinusoidale ed è stato creato il file con estensione `.coe` che il *Block Memory Generator* ha utilizzato per inizializzare la memoria. La ROM presenta una latenza pari ad un ciclo di clock, cioè il dato viene presentato in uscita al successivo fronte di salita del clock. Questo non porta problemi nell'implementazione, si ha solo una traslazione del segnale di un periodo di clock.

[1] Un modulo IP (*Intellectual Property Module*) nel caso più generale, è un blocco circuitale parzialmente o completamente pre-progettato, e disponibile per essere istanziato all'interno di un circuito più complesso.

Le dimensioni della ROM sono quindi $2^{P-2} \cdot (B-1) = 9216$ bits, cioè esattamente la metà di un blocco da 18 Kbit. Se non si fosse usata la tecnica di compressione si sarebbero dovuti utilizzare 3 blocchi RAM da 18 Kbit.

3.2.2 Onda triangolare e onda quadra

Grazie alla tecnica utilizzata, per generare l'onda triangolare basta semplicemente aggirare la ROM. Per questo è stato realizzato un multiplexer che, a seconda del valore del segnale di controllo `sin_tri`, seleziona i 9 bits più significativi all'ingresso della ROM (onda triangolare) oppure i 9 bits di uscita della ROM (onda sinusoidale). Quando viene selezionata l'onda triangolare (`sin_tri` = 1) viene inibito il funzionamento della ROM portando a zero il suo ingresso di enable.
Il segnale triangolare ha uno spettro che contiene delle armoniche a frequenze dispari della fondamentale, con un ampiezza che è proporzionale al quadrato dell'inverso del numero dell'armonica. Ne segue che, mentre l'onda sinusoidale può raggiungere una frequenza di circa 20 MHz, l'onda triangolare può raggiungere una frequenza massima di circa 5 MHz, per avere una buona ricostruzione del segnale.

Selezionando il bit più significativo dell'accumulatore di fase (MSB) è possibile generare un'onda quadra con duty cycle 50% che può essere utilizzata come sorgente di clock per circuiti esterni. L'MSB dell'accumulatore di fase varia ogni π radianti, poichè l'accumulatore rappresenta 2π radianti. In realtà tale onda non ha un duty cycle stabile, a meno che la frequenza di uscita non sia un sottomultiplo delle frequenza di clock. L'errore di fase (*phase jitter*) tra il segnale ideale e quello in uscita cresce fino a raggiungere il massimo di un periodo di clock, poi torna a zero e ricomincia a crescere[2]. L'errore è accettabile per frequenze non troppo elevate.
Per avere in uscita l'onda quadra è stato inserito un multiplexer all'uscita dell'accumulatore che, a seconda del segnale di controllo `sq_wave`, seleziona l'MSB o pone l'uscita a massa. L'onda quadra generata è analogica quindi viene mandata direttamente in uscita all'FPGA su un piedino separato a quelli impiegati per il convertitore D/A.

3.3 Simulazione

Il software di sviluppo ISE della Xilinx include lo strumento ISim per effettuare simulazioni e verificare il corretto funzionamento dei blocchi circuitali progettati.

In figura 3.4 è possibile vedere in dettaglio come avviene il cambio di frequenza. Il segnale di ingresso rd_tick avvisa che c'è una nuova tuning_word in ingresso. Si può notare come al successivo fronte di salita del clock la parola viene memorizzata nel frequency register (ftw_reg). L'uscita dell'accumulatore (phase_out) varia al successivo fronte di salita del clock, facendo

variare la frequenza. Si può notare anche la latenza di un ciclo di clock della ROM. Nel caso in figura si passa da una frequenza di 1 KHz (M = 5369) ad una frequenza di 1 MHz (M = 5368709).

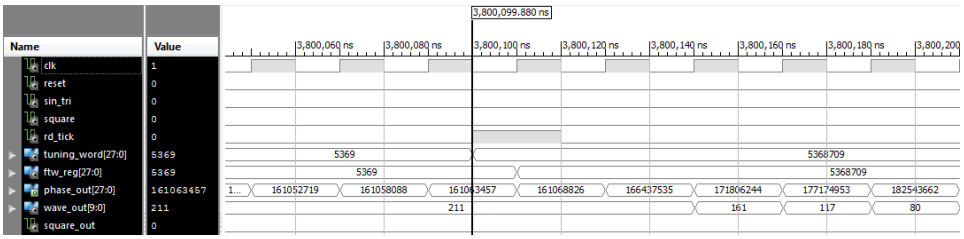


Figura 3.4: Simulazione del DDS core.

Per verificare la corretta generazione delle forme d'onda, i dati di uscita sono stati esportati in un file di testo e utilizzando MATLAB è stato possibile visualizzare le forme d'onda generate. In figura si può vedere il risultato. L'onda sinusoidale passa dalla frequenza di 5 KHz ad 1 KHz, successivamente si è portato ad '1' il valore del segnale sin_tri generando l'onda triangolare. Il segnale sq_wave è sempre attivo fornendo all'uscita square_out anche l'onda quadra. Si può notare la variazione praticamente immediata della frequenza pur mantenendo la continuità nella fase.

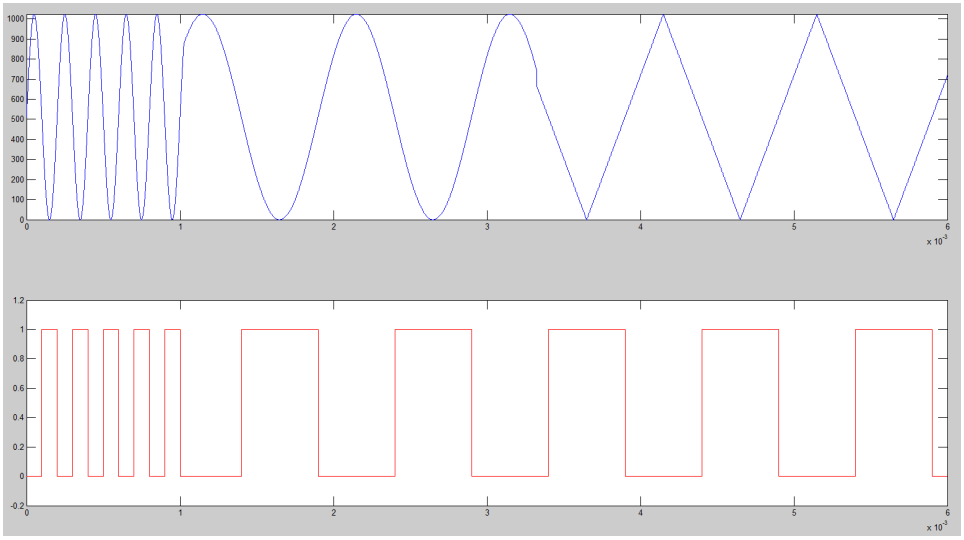


Figura 3.5: Forme d'onda in uscita al DDS core.

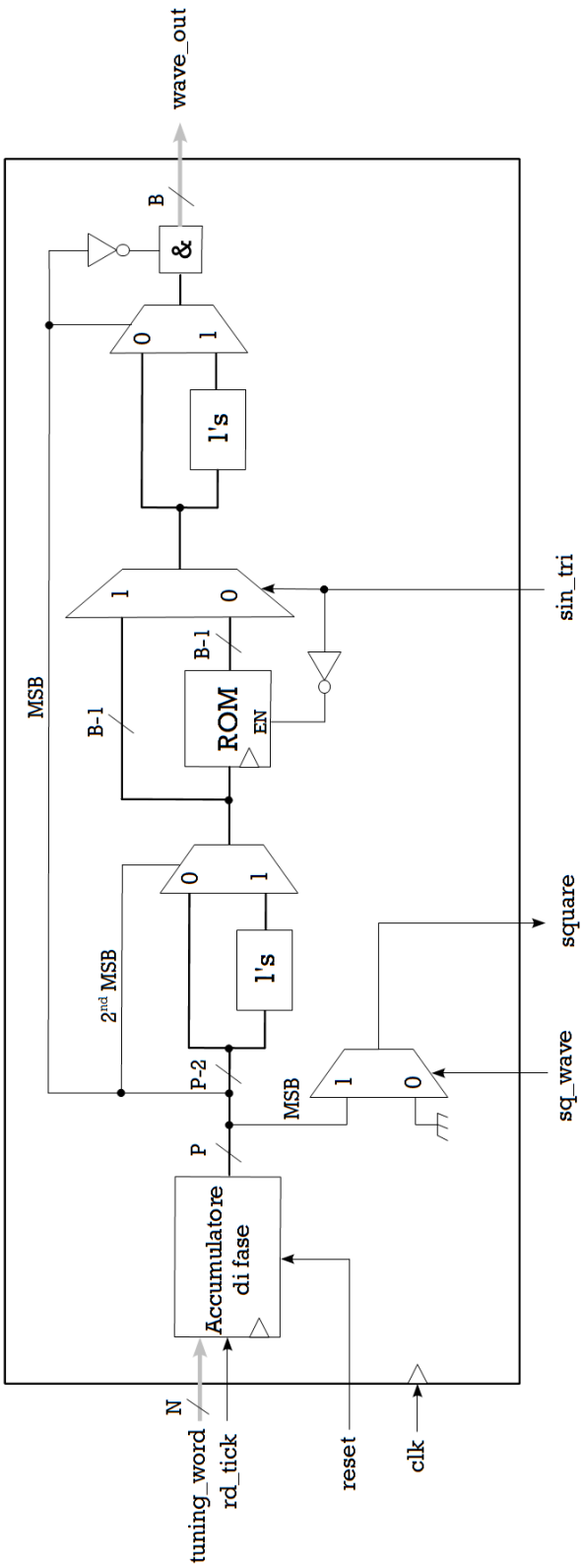


Figura 3.6: Schema a blocchi del core DDS implementato.

Capitolo 4

Ricezione e sintonizzazione

In questo capitolo ci occuperemo di come viene gestita la comunicazione seriale e come viene programmato il sistema DDS.
Un primo blocco circuitale è costituito da un ricevitore UART (*Universal asynchronous receiver and trasmitter*) il quale si occupa della ricezione dei dati dalla linea seriale. L'hardware presente nella board si occupa della gestione dello standard RS-232, quindi ci si è concentrati solo sul progetto del blocco circuitale.
La linea seriale è a livello logico alto quando non vi è comunicazione. La trasmissione comincia con un bit di start, che è '0', seguito dai bit che costituiscono il dato, un bit di parità opzionale e infine un o più bit di stop che valgono '1'. Come si nota in figura, il primo bit trasmesso è il bit meno significativo.

Figura 4.1: Trasmissione seriale di un dato da 8 bits.

Poichè il tipo di comunicazione è asincrona è necessario generare un segnale di clock locale impostato secondo una serie di parametri di trasmissione, quali il *baud rate*, che indica la velocità di trasmissione (cioè il numero di bit al secondo), il numero di bit del dato e di stop e l'eventuale bit di parità. Il numero di bit del dato può essere 6, 7 o 8, mentre il numero di bit di stop 1, 1.5 o 2. Nel progetto il blocco di ricezione è stato impostato con un *baud rate* di 115200 bps, 8 bits di dato, 1 bit di stop e nessun bit di parità.
Poichè nel sistema DDS realizzato la *tuning word* è composta da 28 bit, per programmare il sistema si è pensato di utilizzare una parola da 32 bit in cui i 4 bit più significativi sono utilizzati per impostare il sistema, mentre i restanti 28 bit rappresentano la parola di sintonizzazione (codifica dell'intero

M). Un secondo blocco circuitale avrà quindi il compito di traferire le 4 parole da 8 bit, ricevute dalla seriale, in un registro da 32 bit per poi impostare gli ingressi del sistema DDS.

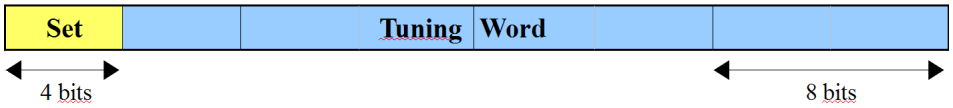


Figura 4.2: Formato della parola di programmazione.

4.1 Ricevitore UART

Per recuperare i bits del dato è stato utilizzato uno schema di sovra-campionamento per determinare i punti medi dei bits e in questi punti riceverli correttamente[5]. Solitamente si utilizza una frequenza di campionamento che è 16 volte il baud rate, ciò significa che ogni bit è campionato 16 volte. Considerando di avere n bit di dato e m bit di stop, il funzionamento può essere riassunto come segue:

1. Un contatore inizia il campionamento quando il segnale va a '0' (inizio della comunicazione);

2. Quando il contatore arriva a 7 allora si è raggiunto il punto medio del bit di start. Si azzera il contatore e si ricomincia il conteggio;

3. Quando il contatore arriva a 15 si raggiunge il punto medio del primo bit del dato. Si inserisce il bit in un registro e si fa ripartire il contatore;

4. Si ripete il punto 3 $n-1$ volte, per salvare i restanti bit del dato;

5. Si ripete il punto 3 m volte per i bit di stop.

Un diagramma a blocchi del ricevitore è visibile in figura 4.3. Esso è composto da tre componenti principali:

- Baud rate generator : si occupa di generare il segnale di campionamento, in base alla velocità di trasmissione (baud rate);

- Ricevitore UART : si occupa di recuperare i dati in base allo schema di sovra-campionamento;

- Buffer FIFO : fornisce una spazio di memorizzazione per la comunicazione tra il ricevitore e il sistema principale che lo utilizza.

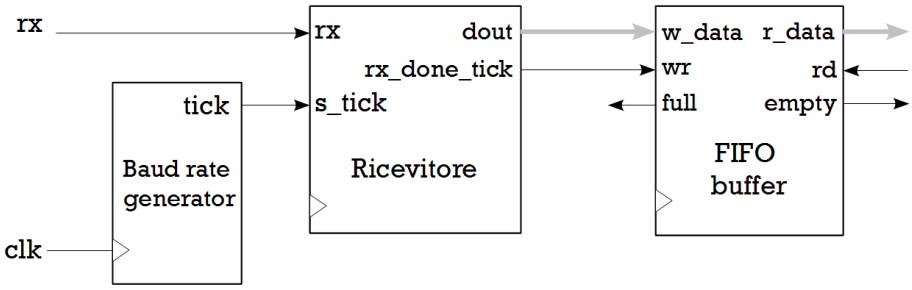


Figura 4.3: Ricevitore UART.

4.1.1 Baud rate generator

Il baud rate generator fornisce un segnale di campionamento la cui frequenza è 16 volte il baud rate. Per evitare di creare un nuovo segnale di clock e mantenere tutto il sistema nel dominio di sincronia del segnale di clock principale (50 MHz), il segnale di campionamento è un segnale impulsivo che genera degli impulsi (*ticks*), della durata di un periodo di clock, alla frequenza di campionamento.

Per un baud rate di 115200, la frequenza di campionamento è $115200 \cdot 16 = 1843200$ impulsi al secondo. Il numero di cicli di clock necessari per ottenere tale frequenza è pari a $\frac{50 \cdot 10^6}{1843200}$, che arrotondato all'intero più vicino da come risultato 27.
Il baud rate generator è formato quindi da un contatore modulo 27 il quale genera un impulso ogni 27 cicli di clock. Il contatore è stato realizzato con due parametri generici che specificano il limite del conteggio e il numero di bit necessari (in questo caso 27 e 5).

4.1.2 Ricevitore UART

Il ricevitore segue la procedura di sovra-campionamento illustrata precedentemente. Il blocco circuitale è stato modellizzato come una macchina a stati finiti FSM (*finite state machine*). Per rendere il codice riutilizzabile si sono utilizzate due costanti, `DBIT` che indica il numero di bits del dato e `SB_TICK` che indica il numero di impulsi necessari per i bit di stop, cioè 16, 24 o 32 per 1, 1.5 o 2 bits di stop. Nel progetto valgono rispettivamente 8 e 16.

In una FSM ad ogni fronte di salita del clock lo stato futuro del sistema, determinato in base al valore degli ingressi e allo stato presente, diventa il nuovo stato del sistema. Quindi, ogni dato definito all'interno del sistema (incluso lo stato) dovrà avere una versione presente (suffiso `_reg`) e una versione futura (suffisso `_next`). A livello di codice si avrà un processo sensibile ai fronti di salita del clock, il cui compito è quello di aggiornare i dati del

sistema. Un secondo processo è invece sensibile agli ingressi e ai dati presenti. La sua funzione è quella di definire tutti i possibili stati del sistema e di determinare lo stato futuro. Le uscite del sistema dipendono generalmente dallo stato attuale del sistema e dagli ingressi. Bisogna prestare particolare attenzione al fatto che tutte le uscite modicate da un certo stato verranno in realtà aggiornate durante l'esecuzione dello stato successivo.

All'ingresso `s_tick` arriva il segnale di campionamento generato dal baud rate generator. Inoltre, ci sono due contatori interni rappresentati dai registri `s` ed `n`. Il registro `s` tiene traccia del numero di impulsi di campionamento mentre `n` tiene il conto del numero di bits del dato ricevuto. I bits del dato ricevuto vengono inseriti riordinati in un registro `b` disponibile in uscita. In uscita al ricevitore vi è inoltre un segnale di stato `rx_done_tick` che và ad '1' per un ciclo di clock quando il dato è pronto all'uscita `dout` del registro.

La macchina è composta da quattro stati che rappresentano la procedura di sovra-campionamento:

- `idle` : la macchina rimane in questo stato fino a che l'ingresso `rx` è a '1'. Quando `rx` passa allo stato '0' ed inizia la comunicazione, ci si porta nello stato `start` e si azzera il registro `s`;
- `start` : ad ogni impulso di campionamento si incrementa il registro `s`; quando `s` arriva a 7 si azzerano i registri `s` ed `n` e ci si porta nello stato `data`;
- `data` : si incrementa `s` fino a che non arriva a 15, a quel punto si può leggere il bit del dato e inserirlo nel registro `b`. Il registro `n` viene incrementato mentre `s` viene resettato. Quando `n` raggiunge DBIT-1 l'ultimo bit del dato è stato inserito nel registro e ci si porta nello stato di `stop`;
- `stop` : si incrementa il registro `s` fino a che non si raggiunge il valore SB_TICK-1, a questo punto ci si riporta nello stato `idle` e si attiva il segnale di stato `rx_done_tick` per avvisare che il dato è pronto all'uscita del ricevitore.

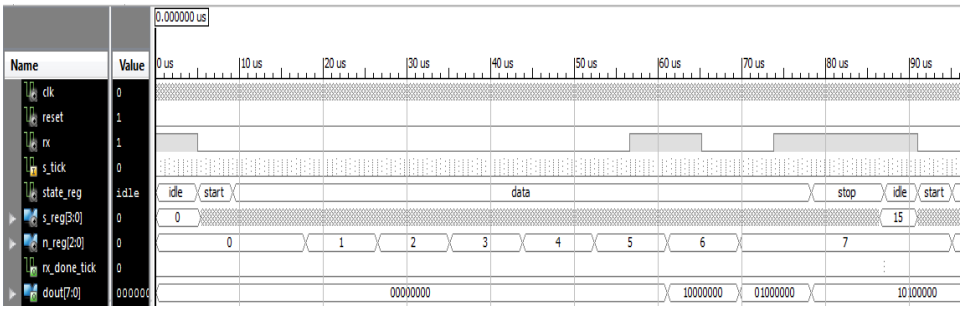


Figura 4.4: Simulazione del blocco ricevitore.

In figura 4.4 è riportata una simulazione, effettuata tramite lo strumento ISim, della ricezione del dato B0 (10100000) dalla linea seriale. Poichè il segnale si clock e il registro `s` variano molto velocemente, in figura non è possibile vedere i valori assunti.

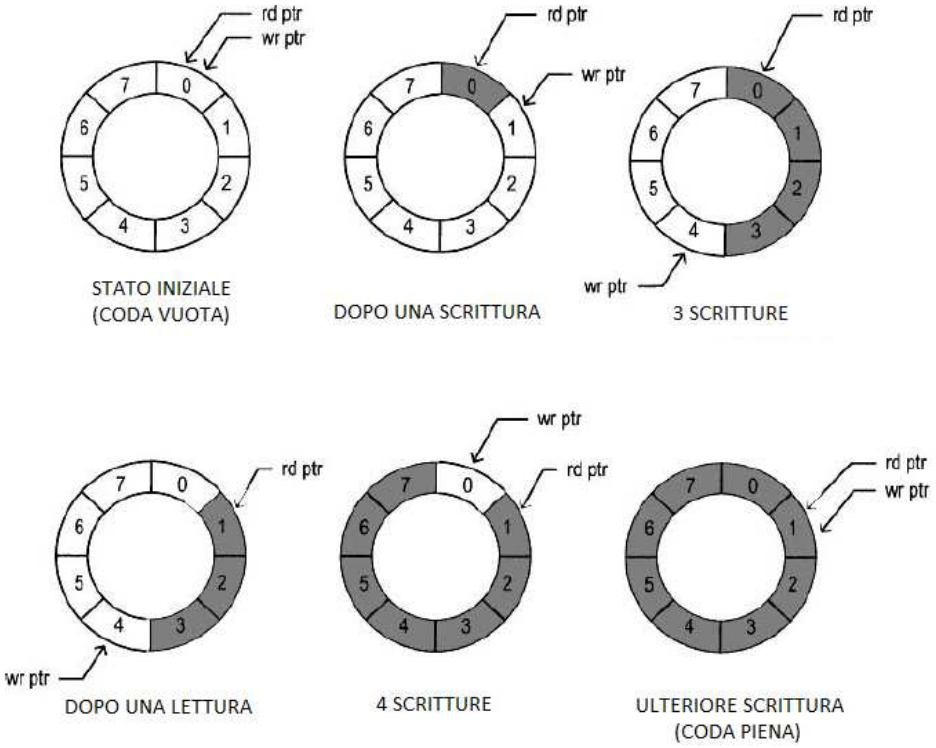


Figura 4.5: Buffer FIFO basato su una coda a struttura circolare.

4.1.3 FIFO buffer

Se il sistema remoto inizia una nuova trasmissione prima che il sistema principale acquisisca il dato precedente, questo può essere sovrascritto portando ad un errore conosciuto come *data overrun*. Per evitare tale inconveniente viene aggiunto un buffer di tipo FIFO (*First In First Out*). A seconda delle necessità del sistema principale è possibile impostare la quantità di memoria della FIFO attraverso due costanti `B` e `W`, che indicano rispettivamente il numero di bits del dato e il numero di bits di indirizzamento.

Il blocco circuitale ha due segnali di controllo, `wr` e `rd`, per le operazione di scrittura e lettura. Il segnale `rx_ready_tick` è connesso al senale `wr` della FIFO. Quando viene ricevuto un nuovo dato, `wr` viene attivato per un ciclo di clock e il dato viene inserito nella FIFO. Il dato in testa alla coda è disponibile all'uscita del blocco. Dopo aver prelevato il dato, il sistema

principale attiva il segnale `rd` per un ciclo di clock, così da rimuovere il primo dato e presentare il dato successivo in uscita alla FIFO.

La memoria è costituito da un array bidimensionale che contiene 2^W registri da B bits. I registri sono organizzati come una coda circolare (figura 4.5) con due puntatori. Il puntatore di lettura `r_ptr` indirizza alla testa della coda, mentra il puntatore di scrittura `w_ptr` indirizza alla coda. I puntatori avanzano di una posizione per ogni operazione di lettura e scrittura.

Il buffer FIFO presenta anche due segnali di stato, `full` e `empty`, che indicano rispettivamente che la memoria è piena (cioè non si può scrivere) e vuota (cioè non si può leggere). Una delle due condizioni avviene quando i puntatori di lettura e scrittura indirizzano lo stesso registro. Per distinguere le due condizioni si sono utilizzati due flip-flop per tenere traccia degli stati empty e full. Inizialmente questi sono impostati a '1' e '0' e vengono modificati ad ogni ciclo di clock in base ai valori che assumono i segnali `wr` e `rd`.

Il codice è diviso in due parti. Una parte si occupa della scrittura e lettura del dato a seconda dei valori assunti dai puntatori. La scrittura avviene solamente se il segnale `wr` è attivo e la coda non è vuota.

```
process(clk, reset)
  begin
    if (reset = '1') then
      array_reg <= (others =>(others => '0'));
    elsif (clk'event and clk = '1') then
      if wr_en = '1' then
        array_reg(to_integer(unsigned(w_ptr_reg))) <= w_data;
      end if;
    end if;
  end process;

  --read port
  r_data <= array_reg(to_integer(unsigned(r_ptr_reg)));

  --scrittura concessa solo quando la coda FIFO non e' piena
  wr_en <= wr and (not full_reg);
```

Listing 4.1: Codice VHDL del buffer FIFO relativo alla parte di scrittura e lettura.

Una seconda parte si occupa del controllo dei puntatori e dei segnali di stato. Ad ogni fronte di salita del clock tali segnali vengono aggiornati a seconda dei valori assunti dai segnali `wr` e `rd`. Quindi, per questi segnali sono state definite una versione presente (suffisso `_reg`) e una versione futura (suffisso `_next`). Per i puntatori è stata definita anche una versione successiva che calcola il valore successivo del puntatore a seconda del valore presente.

```
  --successivi valori dei puntatori
  w_ptr_succ <= std_logic_vector(unsigned(w_ptr_reg)+1);
  r_ptr_succ <= std_logic_vector(unsigned(r_ptr_reg)+1);
```

```
--parte coombinatoria (next-state logic)
wr_op <= wr & rd;
process (w_ptr_reg, w_ptr_succ, r_ptr_reg, r_ptr_succ, wr_op,
         empty_reg, full_reg)
begin
  w_ptr_next <= w_ptr_reg;
  r_ptr_next <= r_ptr_reg;
  full_next <= full_reg;
  empty_next <= empty_reg;
  case wr_op is
    when "00" => --nessuna operazione
    when "01" => --lettura
      if (empty_reg /= '1') then    --se non vuota
        r_ptr_next <= r_ptr_succ;
        full_next <= '0';
        if (r_ptr_succ = w_ptr_reg) then
          empty_next <= '1';
        end if;
      end if;
    when "10" => --scrittura
    if (full_reg /= '1') then      --se non piena
        w_ptr_next <= w_ptr_succ;
        empty_next <= '0';
        if (w_ptr_succ = r_ptr_reg) then
          full_next <= '1';
        end if;
    end if;
    when others => --scrittura/lettura
      w_ptr_next <= w_ptr_succ;
      r_ptr_next <= r_ptr_succ;
  end case;
end process;

--output
full <= full_reg;
empty <= empty_reg;
```

Listing 4.2: Codice VHDL del buffer FIFO relativo alla parte di controllo.

Data la velocità del sistema principale rispetto al sistema remoto, nel progetto si sarebbe potuto fare a meno di un buffer FIFO. Tuttavia per motivi di portabilità del codice si è voluto implementare una struttura generale che può essere impostata a seconda delle esigenze. Nel progetto il buffer FIFO ha solo 2 registri da 8 bits ciascuno. Inoltre la struttura implementata è semplice e meno efficiente rispetto alle soluzioni fornite da Xilinx, le cui implementazioni sono più ottimizzate. La scelta fatta è per scopo di apprendimento personale.

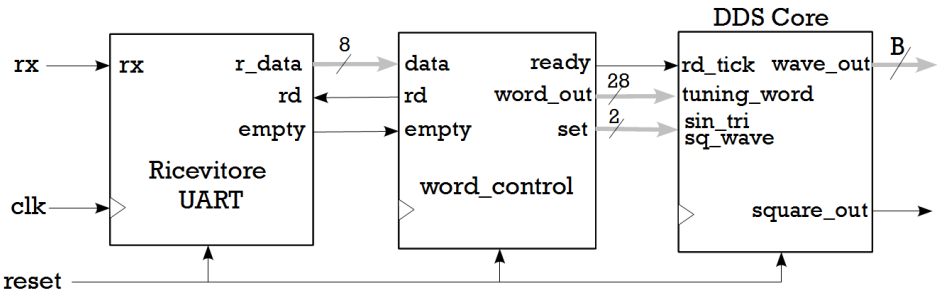


Figura 4.6: Schema a blocchi del sistema completo.

4.2 Sintonizzazione

Per sintonizzare il sistema DDS è stato realizzato un blocco circuitale (`word_control`) che ha il compito di recuperare i dati inviata dalla seriale, ricostruire la parola trasmessa e impostare gli ingressi del DDS core.
Il circuito presenta due ingressi, `data` e `empty`, collegati rispettivamente alle uscite r_data e empty del buffer FIFO. Inizialmente la coda FIFO è vuota, quindi il segnale empty è alto. Quando un dato viene scritto nel buffer il segnale empty passa a livello logico '0' e il dato è disponibile in ingresso. A questo punto il dato viene inserito in un registro interno da 32 bit (`reg`) con una operazione di shift. Il segnale di uscita `rd`, collegato all'ingresso di read della FIFO, viene attivato per avvisare che il dato è stato letto e può essere eliminato dalla coda.
Per tenere conto della quantità di dati inseriti si è utilizzato un contatore modulo 4, che viene incrementato ad ogni dato inserito. Quando il contatore ritorna a zero significa che l'intera parola è stata ricevuta. In questo istante si avvisa il core DDS che una nuova tuning word è disponibile abilitando il segnale di uscita `ready` per un ciclo di clock.
Il sistema DDS realizzato presenta solo due segnali di configurazione che servono ad impostare quale forma d'onda si vuole in uscita. Perciò i 4 bits più significativi (`set`) sono eccessivi. Tuttavia si è pensato che, in caso di modifiche future che aggiungano funzionalità al sistema DDS, si possono gestire più configurazioni, fino ad un massimo di 16. Nel progetto qui sviluppato sono stati utilizzati solo i 2 bits meno significativi, collegati direttamente ai segnali di ingresso del sistema DDS. Le possibili configurazioni sono visibili in tabella.

Solo quando tutti i dati sono stati ricevuti (cioè il contatore torna a zero) il sistema DDS viene configurato. In questo modo la forma d'onda e la frequenza vengono cambiate contemporaneamente. Fino a quel momento i segnali di configurazione (`set`) restano quelli impostati precedentemente.

Forma d'onda	set	`sq_wave`	`sin_tri`
sinusoidale	0000	0	0
triangolare	0001	0	1
quadra e sinusoidale	0010	1	0
quadra e triangolare	0011	1	1

Tabella 4.1: Configurazioni possibili

```
process(clk, reset)
begin
  if reset = '1' then
    reg <= (others => '0');
    set_int <= (others => '0');
    rd_int <= '0';
    ready <= '0';
    count <= "00";
  elsif clk'event and clk = '1' then
    rd_int <= '0';
    ready <= '0';
    if empty = '0' and rd_int = '0' then
      reg <= reg((PBIT+CBIT-DBIT)-1 downto 0) & data;
      rd_int <= '1';
      count <= count + 1;
      if count = "11" then
        ready <= '1';
        set_int <= reg((PBIT+CBIT-DBIT)-1 downto (PBIT-DBIT));
      end if;
    end if;
  end if;
end process;

--output
rd <= rd_int;
word_out <= reg(PBIT-1 downto 0);
set <= set_int;
```

Listing 4.3: Codice VHDL del blocco circuitale wordc_control. PBIT, CBIT, DBIT sono delle costanti e corrispondono rispettivamente a 28 bits della tuning_word, 4 bits per il controllo e 8 bits per il dato.

In figura 4.7 è riportata una simulazione in cui si può vedere in dettaglio come avviene l'inserimento di un dato all'interno del registro e come variano i segnali di uscita. Nel caso in figura il dato inserito è l'ultimo, il contatore quindi torna a zero e si cambiano i segnali di configurazione. In questo caso si passa da una precedente parola `A00014F9`, alla nuova parola di programmazione `3051EB85`. Solo quando l'intera parola è stata ricevuta si cambia l'uscita `set` e si avvisa il sistema che una nuova `tuning_word` è disponibile.

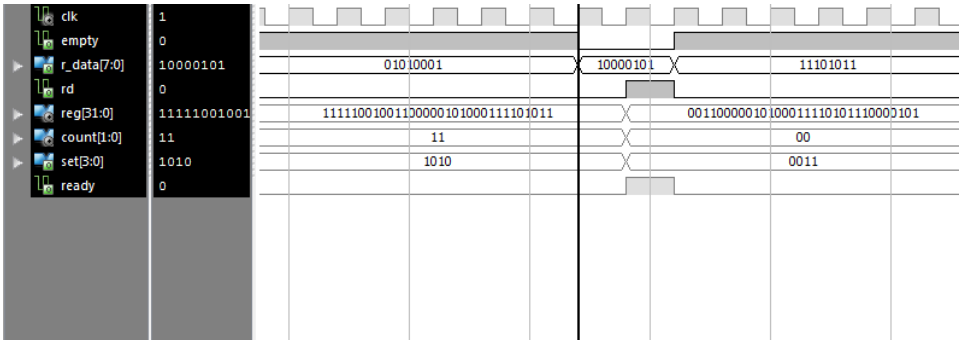


Figura 4.7: Simulazione del blocco word_control.

Capitolo 5

Conclusioni

5.1 Implementazione

L'intero progetto è stato implementato utilizzando il dispositivo FPGA della S3SKB Board. Sono stati utilizzati 3 LED per segnalare i tipi di forme d'onda in uscita (onda sinusoidale, triangolare e quadra). I display a sette segmenti sono stati utilizzati per visualizzare la parola all'uscita del blocco word_control di modo da verificare la corretta ricezione. Poichè la parola è da 32 bit e i display sono solo 4, si è inserito un multiplexer da 16 bits comandato dallo switch SW7, di modo da controllare l'intera parola. I segnali di uscita per il DAC e il segnale d'uscita dell'onda quadra sono stati connessi al connettore A1, mentre il segnale di reset è stato collegato allo switch SW0.

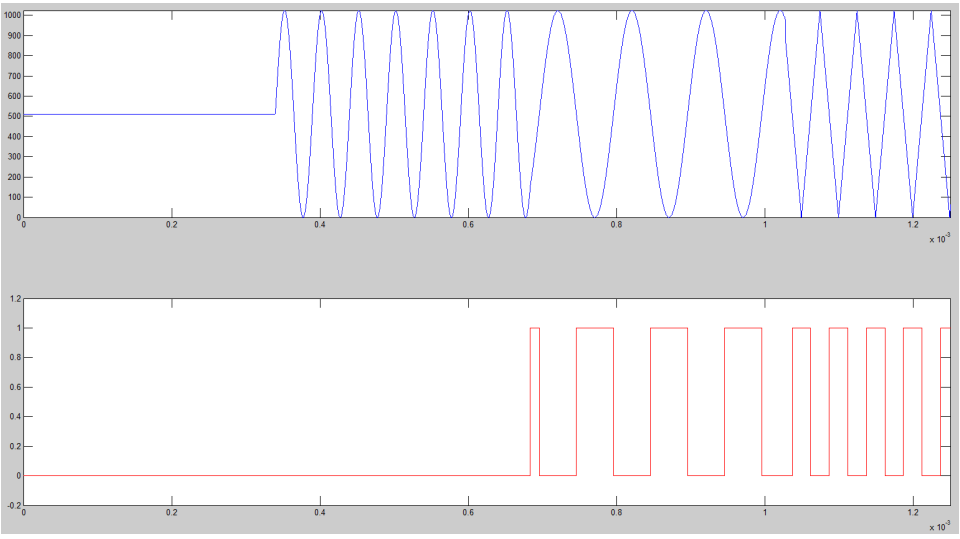


Figura 5.1: Simulazione delle uscite del sistema.

Con uno script MATLAB, a seconda della frequenza e della forma d'onda che si vuole impostare, viene calcolata la parola di programmazione corrispondente e inviata tramite comunicazione seriale all'FPGA. Si è verificato che la ricezione funziona correttamente e il sistema viene impostato in modo corretto.
Poichè il dimensionamento del convertitore D/A e del filtro esulano dagli scopi di questa tesi, per verificare le uscite del sistema si è utilizzato il simulatore ISim e i risultati sono poi stati elaborati con MATLAB per visualizzare il comportamento delle uscite. In figura 5.1 si può vedere il risultato della simulazione. Inizialmente viene impostata l'onda sinusoidale alla frequenza di 20 KHz (0001A36E), poi viene attivata anche l'onda quadra cambiando la frequenza a 10 KHz (2000D1B7) e successivamente si attiva l'onda triangolare e l'onda quadra tornando alla frequenza di 20 KHz (3001A36E).

In figura 5.2 sono riportati i dati di sintesi del progetto. L'analisi dei risultati ottenuti è utile alla comprensione della quantità di risorse utilizzate. Risulta evidente dai dati riportati che la quantità di risorse utilizzate è poca. Con lo strumento XPower Analyzer, incluso nel software di sviluppo ISE, si è potuto stimare il consumo di potenza a 68 mW[1].

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	159	3,840	4%
Number of 4 input LUTs	153	3,840	3%
Number of occupied Slices	132	1,920	6%
Number of Slices containing only related logic	132	132	100%
Number of Slices containing unrelated logic	0	132	0%
Total Number of 4 input LUTs	177	3,840	4%
Number used as logic	153		
Number used as a route-thru	24		
Number of bonded IOBs	30	173	17%
Number of RAMB16s	1	12	8%
Number of BUFGMUXs	1	8	12%
Average Fanout of Non-Clock Nets	2.95		

Figura 5.2: Risorse utilizzate nell'implementazione del progetto.

In figura 5.3 si possono vedere i risultati della sintesi del DDS core. Si può notare che il sistema DDS occupa un'area molto ridotta (uno degli obiettivi che si volevano raggiungere) e può essere istanziato in un progetto più complesso senza l'utilizzo di grandi risorse.

[1]Gran parte dei consumi sono dovuti ai blocchi di I/O e alle correnti di leakage.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	31	1920	1%
Number of Slice Flip Flops	56	3840	1%
Number of 4 input LUTs	50	3840	1%
Number of BRAMs	1	12	8%
Number of GCLKs	1	8	12%

Figura 5.3: Risorse utilizzate dal core DDS.

5.2 Sviluppi futuri

Il sistema DDS realizzato presenta un'architettura basilare ed è possibile utilizzarlo solo come generatore di forme d'onda, per esempio per il test di circuiti esterni. Inoltre è possibile modificare sola la frequenza della forma d'onda.

Una eventuale modifica è quella di aggiungere la possibilà di variare anche la fase dell'onda, semplicemente aggiungendo all'uscita dell'accumulatore una parola da P bits programmabile che codifica il salto di fase[2].
Inoltre, grazie alla capacità di rapida variazione della frequenza e della fase di uscita, un sistema DDS è predisposto per effettuare modulazioni di frequenza (FSK) e di fase (PSK). Una ulteriore modifica quindi sarebbe quella di fornire capacità di modulazione al sistema. In figura 5.4 è mostrato uno schema a blocchi per tale miglioria. In questo caso, a seconda dei valori assunti dai 4 bits più significativi della parola di programmazione, si possono selezionare e programmare i singoli registri e configurare il sistema a seconda della funzione che si vuole far svolgere.

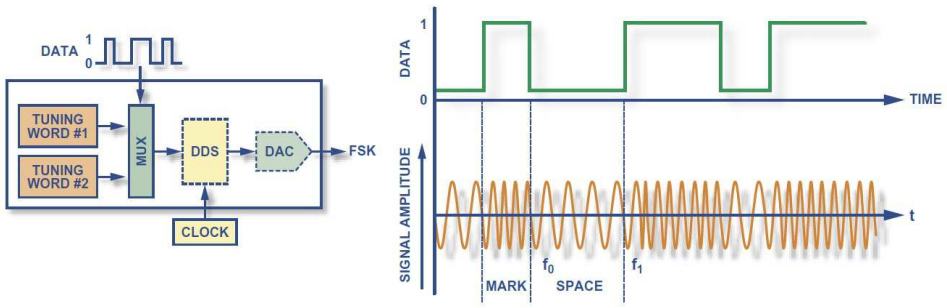


Figura 5.4: Schema per la modulazione FSK.

[2]La risoluzione della fase è $2\pi/2^P$ e il valore dell'incremento si può calcolare dalla formula $\varphi = \Delta\varphi \cdot 2\pi/2^P$.

Bibliografia

[1] Clive "Max" Maxfield, *The Design Warrior's Guide to FPGAs*. Mentor Graphics Corporation and Xilinx, Inc., 2004.

[2] *Spartan-3 Starter Kit Board Reference Manual*. Xilinx, 2005. https://www.digilentinc.com/Data/Products/S3BOARD/S3BOARD_RM.pdf

[3] *Spartan-3 FPGA Family Data Sheet*. Xilinx, 2008. http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf

[4] *Spartan-3 Generation Configuration User Guide*. Xilinx, 2009. http://www.xilinx.com/support/documentation/user_guides/ug332.pdf

[5] Pong P. Chu, *FPGA prototyping by VHDL Examples, Xilinx Spartan-3 Version*. John Wiley & Sons, Inc., 2008.

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

[7] *AD9833: Low Power, 12.65 mW, 2.3 V to +5.5 V, Programmable Waveform Generator DataSheet* , Analog Devices, Inc (2012) http://www.analog.com/static/imported-files/data_sheets/AD9833.pdf

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

[9] Jouko Vankka, *Direct Digital Synthesizers: Theory, Design and Applications*, November 2000.

[10] Lionel Cordesses, *Direct Digital Synthesis: A Tool for Periodic Wave Generation (Part 1)*. IEEE Signal Processing Magazine, Luglio 2004.

[11] Daniele Vogrig, Dispensa del corso di: laboratorio di elettronica digitale. Thechnical Report, Università degli studi di Padova, Facoltà di Ingegneria, 2009.

Ringraziamenti

In primo luogo desidero ringraziare il Prof. Daniele Vogrig, relatore della mia tesi, per la sua professionalità e l'aiuto fornitomi durante la stesura del lavoro.

Un particolare ringraziamento va alla mia famiglia, gli amici e tutti quelli che mi hanno sostenuto in questi mesi e accompagnato nella realizzazione di questa tesi.