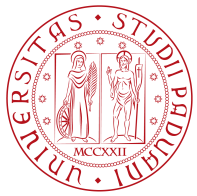


Università degli Studi di Padova

Dipartimento di ingegneria dell'informazione

Corso di laurea in Ingegneria dell'Informazione

Implementazione di un sintetizzatore musicale su scheda Nexys4 DDR

Anno Accademico 2018/19
15 Luglio 2019

Relatore
Daniele Vogrig
Università di Padova

Laureando
Enrico Lumetti

Abstract

Il successo dei sintetizzatori in ambito musicale dagli anni 70 in poi è è stato determinante per l'industria musicale. Grazie alla loro versatilità i sintetizzatori permettono di riprodurre svariati suoni, spesso programmabili dal musicista, e ampliano enormemente le possibilità espressive. Con l'avvento dell'elettronica digitale quasi tutti i sintetizzatori in commercio hanno cominciato a fare uso di tecniche di sintesi digitale e dei microprocessori. In questa tesi ci si propone di realizzare un semplice sintetizzatore digitale polifonico e con sintesi a wavetable, comandabile attraverso il protocollo MIDI. Per la realizzazione del sintetizzatore si farà uso di una scheda Nexys4 DDR, contenente una FPGA Xilinx Artix 7, un dispositivo programmabile molto diverso da un convenzionale processore e programmato attraverso il linguaggio di descrizione dell'hardware VHDL.

Indice

Capitolo 1

Introduzione

Lo scopo di questa tesi è di descrivere la realizzazione di un sintetizzatore musicale implementato su una scheda Nexys 4 DDR della Digilent. La particolarità di quest'ultima è la presenza, al suo interno, di una FPGA, un circuito programmabile molto diverso da un convenzionale microprocessore o microcontrollore. Una FPGA, infatti, si programma con linguaggi di descrizione dell'hardware (HDL) che descrivono la struttura e il funzionamento di un circuito digitale. Il sintetizzatore presenta le seguenti caratteristiche:

- polifonia
- sintesi basata su wavetable (forma d'onda campionata)
- supporto del protocollo MIDI
- audio monofonico

Oltre all'architettura realizzata, vengono discusse in questo lavoro la teoria dietro alla sintesi digitale del segnale sonoro e della PWM utilizzata per pilotare l'uscita audio. Infine si dà una caratterizzazione delle performance del sintetizzatore in termini di signal-to-noise ratio.

1.1 Descrizione di un sintetizzatore musicale

Un sintetizzatore musicale permette la riproduzione di suoni di carattere, intensità e frequenza controllabili dal musicista. L'utilizzo dei sintetizzatori come strumento, sia in studio che dal vivo, si afferma negli anni settanta grazie all'invenzione del *Minimoog*, meno costoso e voluminoso dei sintetizzatori a muro degli anni sessanta. Il *Minimoog* era un sintetizzatore analogico e monofonico, cioè capace di riprodurre solamente una nota alla volta, al contrario dei moderni sintetizzatori detti *polifonici*. Al giorno d'oggi i sintetizzatori sono per la maggior parte implementati in maniera digitale e permettono l'utilizzo di una varietà di tecniche di sintesi del suono. In questa tesi ci si occuperà di un semplice sintetizzatore polifonico, capace di riprodurre forme d'onda campionate e controllabile attraverso il protocollo

MIDI (Musical Instrument Digital Interface), ampiamente diffuso nell'industria musicale. La scheda Nexys 4 DDR utilizzata per il progetto è dotata di una porta seriale usata per comandare il sintetizzatore e di un uscita mini-jack a singolo canale.

1.2 La catena del segnale del sintetizzatore

L'input di un sintetizzatore viene dato dal musicista attraverso uno strumento fisico. Lo strumento invia al sintetizzatore un segnale di controllo che specifica quali note suonare, a che intensità, quali sono i parametri dello strumento, ecc. A tale scopo è universalmente impiegato lo standard MIDI, che specifica sia il protocollo di comunicazione che le caratteristiche elettriche della connessione fisica. Il sintetizzatore si occuperà di riprodurre i suoni desiderati. Due caratteristiche fondamentali del suono riprodotto sono la sua frequenza e il suo timbro. Per variare queste due componenti sono disponibili una varietà di tecniche. In questo progetto ci si avvale della *Direct Digital Frequency Synthesis* per la generazione di una frequenza determinata, mentre si farà uso di una ROM contenente la forma d'onda campionata per ottenere il timbro desiderato. La riproduzione di più note contemporaneamente si ottiene attraverso una fase di *mixing*, per cui le forme d'onda delle note desiderate vengono sovrapposte. Essendo la sintesi di tipo digitale, alla fine della catena è presente un convertitore digitale-analogico, seguito da un connettore fisico.

Il tempo che intercorre tra la sollecitazione dello strumento e l'emissione del suono corrispondente viene chiamato ritardo di propagazione o *delay*. Il feedback uditivo è essenziale ai fini di una performance musicale, per cui il ritardo introdotto dal sintetizzatore dovrebbe essere il più basso possibile, con un limite indicativo di 5 ms. Data la bassa complessità del sintetizzatore realizzato, il tempo di elaborazione del segnale MIDI di ingresso fino alla produzione del suono è trascurabile.

Capitolo 2

Lo standard MIDI

2.1 Il protocollo di comunicazione e i messaggi MIDI

Nel protocollo MIDI [2] le informazioni e gli eventi vengono trasmessi sotto forma di *messaggi*. Ogni messaggio è composto da una sequenza di byte ordinata di cui il primo è detto *status byte* e i successivi vengono detti *data byte*. Per riconoscere gli status byte dai data byte si impone che il bit più significativo degli status byte sia sempre 1, quello dei data byte sia 0 e i rimanenti 7 bit possono rappresentare valori da da 0 a 127. I primi 4 bit dello status byte identificano il tipo di messaggio MIDI, mentre gli ultimi 4 specificano il canale su cui il messaggio ha effetto, per un massimo possibile di 16 canali.

Ai fini del progetto, verranno gestiti solamente i messaggi di **note on** e **note off** sul canale '0', mentre ogni altro tipo di messaggio sarà ignorato. Ogni messaggio di *note on* viene trasmesso nel momento in cui una certa nota deve essere suonata e contiene due data byte: il primo, il **note number**, identifica la nota da riprodurre, mentre il secondo, detto **velocity**, fornisce un'informazione sull'intensità con cui la nota viene suonata. Il suo status byte è del tipo *1001xxxx* dove *xxxx* sta ad indicare il canale del messaggio. Il messaggio note off è analogo e viene trasmesso quando si vuole interrompere

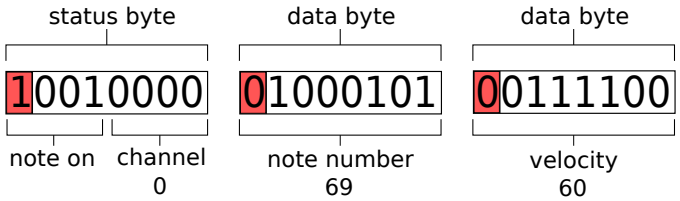


Figura 2.1: Struttura di un messaggio MIDI di tipo *note on*, con note number 69 e velocity 60. In rosso è evidenziato il bit più significativo.

la riproduzione di una certa nota e il suo status byte ha il formato *1000xxxx*. Un altro modo di interrompere la nota è quello di mandare un messaggio *note on* con *velocity* impostata a zero.

2.2 Physical Layer

I messaggi MIDI vengono trasmessi in maniera seriale e vengono ricostruiti da un convertitore seriale/parallelo detto *UART* (Universal Asynchronous Receiver-Transmitter). La comunicazione MIDI segue il protocollo RS-232 e funziona nel seguente modo:

- In assenza di comunicazione, la linea dati viene mantenuta al valore logico alto.
- Per iniziare una trasmissione, il trasmettitore porta la linea dati al valore logico basso per la durata T_{bit}, e questo valore prende il nome di **start bit**.
- I bit da trasmettere vengono inviati in sequenza, in ordine dal bit meno significativo al più significativo; ad ogni bit corrisponde il valore logico alto o basso sulla linea dati per il periodo T_{bit}
- Alla fine della tramissione la linea dati viene portata al valore logico alto per un tempo di almeno T_{bit}, cioè si invia uno **stop bit**.

Il tempo $T_{bit} = \frac{1}{f_{bit}}$ è definito nello standard MIDI imponendo la frequenza di bit a $f_{bit} = 31\,250\,\text{Hz}$, detta anche *baud rate*. La sequenza di *start bit*, bit di informazione e *stop bit* compongo il **data frame** del protocollo MIDI. Si noti che è necessaria a priori la conoscenza del numero di bit da trasmettere perché la comunicazione abbia successo; nello standard MIDI, ad ogni comunicazione viene inviato un byte e quindi 8 bit. Il protocollo RS-232 permette anche l'aggiunta di un secondo stop bit e di un *parity bit* per il controllo degli errori, tuttavia questi non sono usati nella comunicazione MIDI. A differenza del protocollo RS-232 i valori logici alti sono segnalati dallo scorrere di una corrente di 5 mA invece che dalla presenza di un voltaggio positivo. Il valore logico basso corrisponde all'assenza di corrente.

2.3 Il connettore MIDI

La prima specifica MIDI prevedeva un connettore DIN a 5 pin, con un cavo di lunghezza massima di circa 15 m usato per trasmettere la corrente relativa al bit trasmesso. Sebbene ancora supportato, questo tipo di connettore fisico è stato soppiantato con l'avvento del protocollo USB e del relativo connettore, universalmente supportato dai moderni sintetizzatori come mezzo fisico di trasmissione per i messaggi MIDI.

In questo progetto non si fa utilizzo del connettore MIDI, che dovrebbe essere interfacciato con un circuito alla scheda; il testing è stato dunque

condotto solamente attraverso l'utilizzo di un computer che comunicasse con la scheda attraverso la porta USB seriale.

2.4 Il MIDI tuning standard

Ad ogni *note number* del protocollo MIDI viene associata una frequenza determinata dal MIDI tuning standard (MTS). Date due frequenze $f_2 > f_1$ si dice che distano un'*ottava* se vale la relazione $f_2 = 2 \cdot f_1$. Si suddivide ogni ottava in 12 note equidistanziate secondo una progressione geometrica, per cui per ogni nota di frequenza f vale

$$fs^{12} = 2f$$

Da cui si ricava la ragione $s = 2^{\frac{1}{12}}$ della progressione geometrica. Moltiplicando una frequenza per s si ottiene la frequenza della nota successiva. Fissando convenzionalmente la frequenza della nota numero 69 a 440 Hz, si ottiene la corrispondenza tra il note number i e la frequenza f associata:

$$f = 440\,\text{Hz} \cdot s^{i-69}$$

Dal punto di vista musicale, ogni ottava contiene 12 note, e queste si ripetono più acute o più gravi passando all'ottava successiva o precedente. Alla frequenza di 440 Hz si associa convenzionalmente il "la" da concerto, con cui si accordano gli strumenti di un'orchestra.

Capitolo 3

La scheda Nexys4 DDR

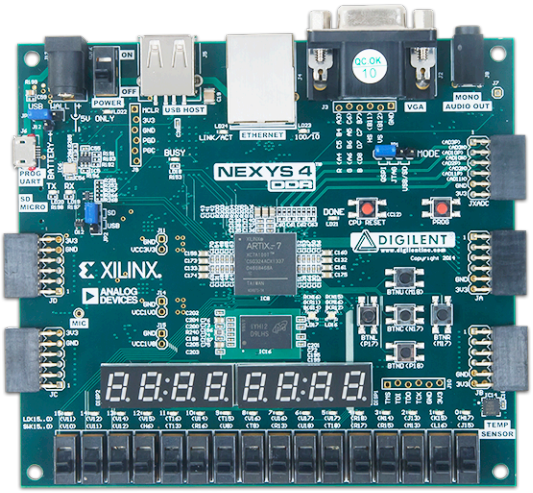


Figura 3.1: Scheda Digilent Nexys4 DDR.

L'hardware utilizzato per il progetto consiste in una scheda Nexys4 DDR. Il nucleo della scheda è costituito da un FPGA Xilinx Artix 7 XC7A100T. Attorno ad esso sono collegati vari connettori, sensori e pulsanti [5].

Per la realizzazione del sintetizzatore si farà uso dei seguenti componenti:

- **Bridge USB-UART** Il bridge USB<->UART permette la lettura dei messaggi MIDI, inviati dal computer attraverso la porta seriale
- **Mono Audio Output** Un connettore audio monofonico di tipo minijack viene usato per emettere il suono.
- **Clock Crystal** Il cristallo che fornisce un clock di 100 MHz alla FPGA e alla scheda.

L'output fornito dal jack audio viene generato attraverso un convertitore analogico/digitale, costituito da un filtro passa-basso a cui viene fornito in

ingresso il segnale da generare in codifica PWM.

Il filtro passabasso è un filtro di Butterworth del quarto ordine, la cui risposta in frequenza è rappresentata nella fig. 3.2

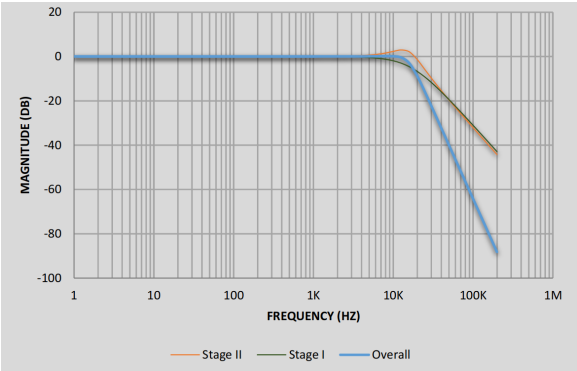


Figura 3.2: Risposta in frequenza del filtro passa-basso prima dell'uscita audio

3.1 FPGA

Una Field-programmable Gate Array (FPGA) è un circuito digitale integrato capace di implementare le più svariate funzioni in modo programmatico: può essere utilizzato per la fase di prototipazione di Application-specific Integrated Circuits (ASIC), oppure per sostituirli nel caso in cui i costi di integrazione di una FPGA nel progetto siano minori dei costi di sviluppo e realizzazione dell'ASIC (che di solito sono più vantaggiosi per volumi di produzione molto grandi). Un FPGA, data la sua natura generale, offre performance inferiori sia in consumi che in prestazioni rispetto ad un ASIC, ma ha come vantaggio la riprogrammabilità (anche una volta integrato nel progetto), che è invece impossibile con un circuito integrato specifico. Un altro uso che si va affermando negli ultimi anni è l'utilizzo dei FPGA come acceleratori software, ad esempio per applicazioni di trading finanziaro e intelligenza artificiale.

3.1.1 Struttura di un FPGA

Una FPGA ha una struttura regolare e contiene:

- **configurable logic blocks**: in breve **CLB**, blocchi logici configurabili che permettono di realizzare funzioni logiche arbitrarie
- **configurable routing**: i blocchi logici sono collegati attraverso delle connessioni a loro volta configurabili, rendendo possibile creare funzioni logiche a un maggior numero di ingressi e più complesse

Oltre a queste due componenti fondamentali, una FPGA contiente altri componenti necessari al suo funzionamento e interfacciamento con circuiti esterni. L'FPGA utilizzato dalla scheda Nexys 4 DDR è uno Xilinx Artix 7 XC7A100T e contiene svariati blocchi [3], tra cui:

- **I/O blocks**: blocchi di input e output, permettono alla FPGA di interfacciarsi con l'esterno
- **DSP blocks**: blocchi che rendono più efficiente in termini di risorse l'implementazione di sommatori e moltiplicatori
- **Memory blocks**: è possibile trovare anche blocchi di memoria all'interno di una FPGA
- **Clock Management Tiles**: oltre a contenere la circuiteria necessaria per la gestione e propagazione del segnale di clock, permettono operazioni avanzate sul segnale di clock come la generazione di clock a frequenze diverse o di clock sfasati
- **Dedicated blocks**: Blocchi contenenti funzionalità avanzate, come ad esempio l'interfaccia PCI Express e blocchi di I/O ad alta velocità

Esso contiene 101440 logic elements (i blocchi elementari di un CLB), e sei clock management tiles (CMT) con phase-locked loop. Di importanza per il progetto trattato sono i 240 DSP slices, data la presenza di molti sommatori all'interno del design, e i 4860 Kbits di Block Ram (BRAM) che vengono impiegati in fase di sintesi per memorizzare sia la forma d'onda campionata che le frequency tuning words impiegate nella digital direct synthesis.

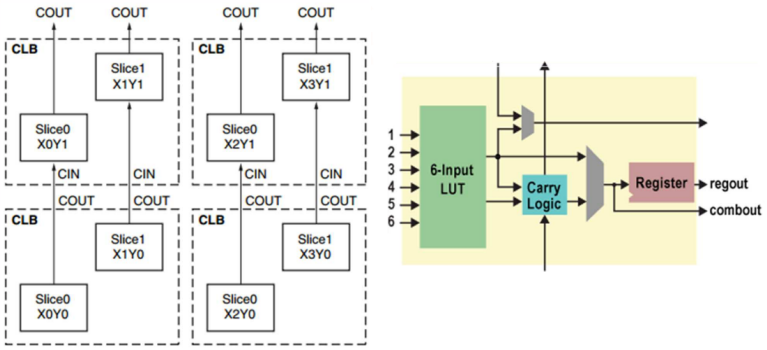


Figura 3.3: Struttura gerarchica di una FPGA: a sinistra l'insieme di più CLB, a destra la struttura di un singolo logic element

3.1.2 Struttura di un CLB

Il CLB è la macrostruttura fondamentale di una FPGA. I circuiti elementari all'interno di un CLB sono i logic elements, che implementano con una look-up table (LUT) una funzione logica a 5 o 6 ingressi. Alcune LUT sono utilizzabili anche come elementi di sola memorizzazione. Ogni logic element

contiene anche un elemento di memoria in cui poter memorizzare il risultato dell'operazione, configurabile sia come flip-flop che come latch. Quattro logic elements, insieme a dei multiplexer e alla catena del carry formano uno logic slice. Due slice combinati formano quindi un CLB nella sua interezza.

3.2 Programmazione di un FPGA

La programmazione di una FPGA avviene a livello fisico attraverso la scrittura di una SRAM che indica come vanno configurati i blocchi logici e le interconnessioni presenti nell'IC.

A livello di sviluppo si utilizzano linguaggi di descrizione dell'hardware (HDL) come VHDL e Verilog, che nascondono in gran parte la complessità dell'FPGA.

Il codice sorgente viene processato da un programma di sintesi che compie le seguenti operazioni:

- **Creazione del netlist**: il codice viene convertito in una netlist composta da componenti ad alto livello (come sommatori, multiplexer, registri ecc..) che tuttavia non combacia con la struttura fisica della FPGA
- **Place and Route**: la netlist viene mappata fisicamente sui componenti della FPGA
- **Bitstream Generation**: Il risultato del Place and Route viene convertito in una serie di bit che vengono poi scritti sulla SRAM della FPGA in fase di programmazione

3.3 Vincoli temporali

In fase di progetto e di sviluppo il design può essere simulato e analizzato ben prima di passare alla programmazione della scheda. Un passo particolarmente importante è l'analisi dei tempi di propagazione: successivamente al place and route, è possibile calcolare il tempo di propagazione del segnale attraverso i circuiti della FPGA. Nel caso in cui siano presenti delle route con un tempo di propagazione troppo lungo, sarà necessario intervenire a livello del design o addirittura manualmente nel posizionamento dei blocchi sulla FPGA. Questo passo viene detto soddisfacimento dei vincoli temporali (timing constraints).

Il periodo di clock utilizzato nel progetto è di $T_{clk} = 10\,\text{ns}$, che è il minore possibile sulla scheda Nexys4 DDR. In un collegamento tra due elementi sequenziali (ad esempio flip-flop) il tempo di propagazione dell'uscita del primo elemento all'entrata del secondo elemento non può quindi superare il periodo di clock. Ordinando tutte le route per tempo di propagazione decrescente e considerando quella avente tempo t_{max} di propagazione maggiore, si definisce il **Worst Negative Slack** (WNS) come:

$$\text{WNS} = T_{clk} - t_{max}$$

Quando questo diventa negativo vuol dire che i timing constraints non sono rispettati.

Capitolo 4

Sintesi digitale del segnale

4.1 Sintesi digitale diretta

La sintesi digitale diretta (Direct Digital Synthesis, Direct Digital Frequency Synthesis, DDS) permette di generare un segnale periodico di frequenza programmabile attraverso un sistema digitale.

Sia $x(\phi)$ un segnale periodico di periodo 2π e tale che $|x(\phi)| \leq 1$. In termini formali, si vuole generare un segnale $s(t) = Ax(2\pi ft)$ periodico di periodo $T = \frac{1}{f}$, di ampiezza A, dove

$$\phi(t) = 2\pi ft \tag{4.1}$$

viene detta **fase** del segnale, ed è inizialmente nulla.

Si noti che le ipotesi fatte sul segnale $x(\phi)$ non sono restrittive in quanto un qualsiasi segnale $y(z)$ periodico di periodio T_y e tale che $|y(z)| \leq y_{max}$ può essere ricondotto ad un segnale dello stesso tipo di x operando la trasformazione

$$x(\phi) = \frac{1}{y_{max}} y\Big(\frac{T_y}{2\pi}\phi\Big)$$

Un sistema di DDS si compone di due blocchi: il **phase accumulator** (PA), responsabile di generare la fase del segnale a una determinata frequenza, e il **phase to amplitude converter** (PAC) che, a partire dalla fase, ottiene l'ampiezza del segnale.

In un sistema a campioni, il PAC si avvale di una ROM di $2^{\mathtt{M}}$ parole contenente il segnale campionato a `b` bit. Altri modi di implementare il PAC sono algoritmici, come ad esempio l'algoritmo CORDIC che viene impiegato per generare segnali di tipo sinusoidale.

Il *phase accumulator* è un registro a `N` bit il cui valore rappresenta la frazione del periodo $[0, 2\pi]$ del segnale $x(\phi)$ in notazione fixed point. Ad ogni fronte di salita del clock il valore `F` del registro viene incrementato di un valore pari alla *frequency tuning word*, sempre di `N` bit, che è in relazione biunivoca con la frequenza f desiderata.

Nella sintesi digitale, la fase al tempo t si ottiene dal registro di fase secondo quando descritto prima:

$$\phi = 2\pi \frac{\mathtt{F}}{2^{\mathtt{N}}} \tag{4.2}$$

Essendo `F` a precisione finita, si avrà un errore sulla precisione della fase del segnale.

Sommando la frequency tuning word al valore del registro di fase `F` si ottiene il nuovo valore `F'` e poiché l'addizione avviene ad ogni periodo di clock T_{clk}, i valori della fase del segnale corrispondenti sono $\phi(t)$ e $\phi(t+T_{clk})$. Si può quindi ricavare la `FTW` utilizzando le eq. (4.1) e (4.2), imponendo:

$$\mathtt{FTW} = \mathtt{F'} - \mathtt{F} = \left\lfloor \frac{2^{\mathtt{N}}}{2\pi}\phi(t+T_{clk}) - \frac{2^{\mathtt{N}}}{2\pi}\phi(t) \right\rfloor = \lfloor 2^{\mathtt{N}} f T_{clk} \rfloor = \left\lfloor 2^{\mathtt{N}} \frac{f}{f_{clk}} \right\rfloor$$

dove $\lfloor \cdot \rfloor$ indica la parte intera.
La frequenza effettiva f_o del segnale risultante dalla DDS è dunque

$$f_o = \frac{\phi(T_{clk}) - \phi(0)}{2\pi T_{clk}} = \frac{2\pi\mathtt{FTW}}{2^{\mathtt{N}} 2\pi T_{clk}} = \frac{\mathtt{FTW}}{2^N} f_{clk} \tag{4.3}$$

in generale diversa dalla f desiderata per via dell'errore di troncamento della fase. Tuttavia, scegliendo un numero di bit sufficiente per il registro di fase, si riesce ad ottenere una precisione molto elevata: usando la eq. (4.3) e imponendo $\mathtt{FTW} = 1$ e $N = 32$, si ricava come risoluzione del registro di fase un valore di circa 0.02 Hz.

In generale non è necessario che il segnale da rappresentare sia a fase iniziale nulla ed è facile rappresentare un segnale con uno sfasamento iniziale arbirario usando il registro di fase. Cambiando la FTW durante il processo di DDS è possibile cambiare quasi istantaneamente la frequenza del segnale, cosa invece difficile da ottenere con tecniche analogiche.

Il registro di fase si può considerare come una versione quantizzata della fase reale del segnale $\phi(kT_{clk})$, a meno di multipli di 2π. La fase è quindi un segnale avente la stessa frequenza f rappresentata dalla frequency tuning word impiegata e campionato alla fequenza di clock, per cui vale il teorema del campionamento di Shannon:

$$2f < f_{clk} \implies f < \frac{f_{clk}}{2}$$

È quindi impossibile generare segnali aventi frequenza maggiore della metà della frequenza di clock, condizione ampiamente rispettata per i segnali audio: l'udito umano è capace di percepire frequenza fino a circa 21 kHz, per cui, dal teorema del campionamento, è necessaria una frequenza di campionamento f_s di almeno 42 000 Hz. Il PAC genera un nuovo valore alla frequenza

di campionamento scelta, usando una ROM contenente i campioni, a partire dal valore del registro di fase.

4.2 Formato della ROM dei campioni

La ROM viene generata quantizzando il segnale $Ax(\phi)$ nell'intervallo $[0, 2\pi]$ e $0 < A < 1$. Sebbene la precisione del registro di fase possa essere molto elevata, non è possibile quantizzare il segnale per ogni possibile valore del registro di fase: ad esempio, un registro di fase a 32 bit comporterebbe il campionamento di 2^{32} valori, e considerando una risoluzione di 11 bit si dovrebbero impiegare $11 \cdot 2^{32}$bit, corrispondenti a 5.5 GiB, che è una quantità di memoria proibitiva. Si campiona dunque il segnale $x(\phi)$ in $2^{\mathtt{M}}$ punti, con $\mathtt{M} < \mathtt{N}$ e si quantizza uniformemente con una precisione di $\mathtt{b}$ bit. All'indirizzo di memoria $\mathtt{I}$ si trova il campione corrispondente a $x(2\pi\frac{\mathtt{I}}{2^{\mathtt{M}}})$ il cui valore quantizzato $\mathtt{Q}$ a b bit è

$$\mathtt{Q} = \left\lfloor 2^{\mathtt{b}} A x(2\pi \frac{\mathtt{I}}{2^{\mathtt{M}}}) \right\rfloor \tag{4.4}$$

I valori quantizzati vengono rappresentati in notazione binaria in complemento a due, permettendo così l'uso dell'aritmetica con segno nella fase di mixing del sintetizzatore. L'indirizzamento a partire dal registro di fase si ottiene prendeno i primi $\mathtt{M}$ bit del registro come indirizzo di memoria della ROM: si ha quindi un errore aggiuntivo nella DDS dovuto alla limitatezza della memoria dei campioni.

4.3 Conversione D/A attraverso PWM

Per portare il segnale discreto generato attraverso la DDS al dominio analogico si fa uso di una tecnica di modulazione detta **pulse-width modulation** (PWM).

Per comprendere il funzionamento della PWM si consideri il caso in cui si voglia rappresentare un segnale costante $x(t) \equiv x_0$ con $x_0 \in [0, x_{max}]$. L'ipotesi di un segnale $x(t)$ positivo, sebbene non generale, è sufficiente a caratterizzare il segnale PWM processato dal filtro passa-basso della scheda Nexys, essendo questo un segnale a voltaggio maggiore o uguale a zero. Si dovrà quindi ricondurre il segnale campionato nella ROM, nel range $[-1, 1]$, ad un segnale nel range $[0, x_{max}]$ con opportune operazioni di scala e traslazione.

Sia quindi $x_{pwm}(t)$ un'onda quadra di ampiezza x_{max}, di periodo T e di duty cycle $d = \frac{x0}{x_{max}}$, $d \leq 1$:

$$x_{pwm}(t) = \begin{cases} x_{max} & \text{se } kT \leq t < (k+d)T \\ 0 & \text{se } (k+d)T \leq t < (k+1)T \end{cases}$$

con k intero.

Questo segnale è sviluppabile in serie di Fourier, ed essendo un'onda quadra conterrà i multipli dispari dell'armonica fondamentale di frequenza $\frac{2\pi}{T}$. Il coefficiente a_0 della serie è inoltre pari alla media del segnale sul periodo, ossia

$$a_0 = x_{max} \cdot d = x_0$$

È possibile quindi ricostruire il segnale $x(t) \equiv x_0$ a partire da $x_{pwm}(t)$ passandolo attraverso un filtro passa-basso che rimuova tutte le armoniche di frequenza più alta di a_0.

Nel caso in esame $x(t)$ non sarà costante, ma sarà variabile e rappresenterà la forma d'onda generata dal sistema di sintesi digitale diretta.

Al segnale costante verrà quindi sostituito il segnale a tempo discreto $x(kT), k \in Z$ generato alla frequenza di campionamento $f_s = \frac{1}{T}$ e con valori $0 \leq x(kT) \leq x_{max}$, e in prima approssimazione il segnale x_{pwm} corrispondente sarà

$$x_{pwm}(t) = \begin{cases} x_{max} & \text{se } kT \leq t < (k + d_k)T \\ 0 & \text{se } (k + d_k)T \leq t < (k + 1)T \end{cases}$$

dove $(d_k)_{k \in \mathbb{N}}$ è una sequenza di duty cycle non più costanti ma **modulati** e dati da

$$d_k = \frac{x(kT)}{x_{max}}$$

Tuttavia il segnale PWM è generato digitalmente, quindi supponendo per semplicità $T = 1/f_s$ multiplo del periodo di clock $1/f_{clk}$, in un singolo periodo di campionamento si avrà una successione di $N = \frac{f_{clk}}{f_s}$ impulsi pari a x_{max} (corrispondente ad un valore logico alto) o 0 (corrispondente a un valore logico basso). Poiché al fine della modulazione PWM conta solo il valore medio sul periodo, l'ordine degli impulsi non cambia il risultato, anche se cambia la distribuzione delle armoniche nello spettro del segnale. È quindi possibile rappresentare un numero finito di valori in uscita pari a N, e quindi i campioni forniti potranno al massimo avere una risoluzione di $\log_2 N$ bit.

La modulazione PWM non è lineare e introduce quindi distorsione nel segnale ricostruito.

Capitolo 5

Architettura ed implementazione

L'architettura da sintetizzare su FPGA è stata sviluppata utilizzando il linguaggio VHDL, facendo ampiamente uso di molte funzionalità della versione VHDL-2008. Il progetto è stato sintetizzato usando il software Vivado 2019.1 della Xilinx e simulato usando il compilatore/simulatore open-source GHDL.

Dal punto di vista architetturale si può considerare il progetto come composto da due macro-blocchi indipendenti: uno che si occupa di gestire i segnali MIDI dati in input al sintetizzatore e uno che si occupa di generare il segnale audio in uscita.

Le due componenti sono implementate in logica sincrona e sono collegate da una lookup table che memorizza le note attive. Alla ricezione di un evento MIDI *note on* il bit corrispondente alla nota nella LUT viene portato al valore logico alto, e al seguente istante di campionamento la componente che genera il segnale di output userà questo valore per determinare se suonare o meno la nota.

5.1 Scelta dei parametri di progetto

Mentre la frequenza di clock $f_{clk} = 100\,\text{MHz}$ e la frequenza di campionamento minima $f_{s,min} = 2 \cdot 21\,000\,\text{Hz}$ sono date, non sono note a priori i seguenti parametri:

- `b`: Numero di bit usati per quantizzare la forma d'onda
- `N`: Numero di bit usati per il registro di fase nel phase accumulator
- `M`: Numero di bit presi tra i primi bit del registro di fase e usati per indirizzare i campioni

Utilizzando i vincoli imposti dalla conversione D/A in PWM, si ottiene il numero massimo di bit del quantizzatore imponendo che

$$\texttt{b} < \left\lceil \log_2 \frac{f_{clk}}{f_{s,min}} \right\rceil = 12$$

Conviene quindi massimizzare `b` per avere una minore errore di quantizzazione, e allo stesso tempo massimizzare il più possibile f_s, scegliendola per comodità come sottomutiplo della frequenza di clock. Si ottiene così $\texttt{b} = 11$ e $f_s = 48\,828\,\text{Hz}$

Nella seguente tabella sono riportati i valori dei parametri e, in aggiunta, i relativi nomi all'interno del progetto VHDL.

Parametro	**Valore**	**Descrizione**	**VHDL constant**
f_{clk}	100 MHz	frequenza di clock del FPGA	`clock_frequency`
f_s	48 828 Hz	frequenza di campionamento	`sampling_frequency`
`b`	11	numero di bit dei campioni	`sample_bits`
`N`	32	numero di bit del registro di fase e della frequency tuning word	`phase_bits`
`M`	13	numero di bit degli indirizzi dei campioni	`address_bits`

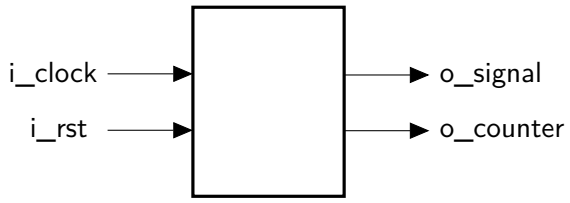
5.2 Componenti fondamentali

I seguenti blocchi fondamentali corrispondono in VHDL al costrutto di *entity* e veranno analizzati dettagliatamente in seguito:

- `counter_impulse_generator` Un contatore con frequenza di reset variabile, viene usato principalmente per la sincronizzazione temporale tra diversi componenti
- `low_to_high_detector` Componente ausiliario che trasforma la variazione del segnale in ingresso dal valore logico basso a quello alto in un impulso della durata di un ciclo di clock
- `uart` Si occupa della ricezione seriale a 31250 baud secondo le specifiche MIDI
- `midi_decoder` State machine che codifica i byte ricevuti dal blocco uart in eventi MIDI; gestisce solamente gli eventi note on/off mentre scarta tutti gli altri eventi MIDI.
- `active_notes_lut` Tabella di lookup contenente 128 valori booleani che indicano l'attivazione di una nota
- `phase_accumulator` Opera l'accumulazione di fase per la DDS
- `pwm_encoder` Genera la modulazione PWM corrispondente in uscita a partire da un campione fornito in entrata
- rom Package generico che rappresenta una read-only memory.

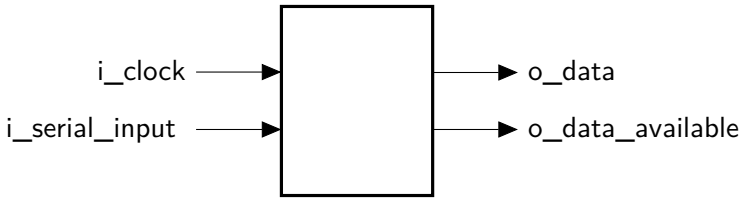
- `synth_engine` Cuore del sintetizzatore, provvede alla DDS a partire dai valori presenti in active_notes_lut, alla phase-to-amplitude conversion attraverso una ROM e al mixing.
- `synth_top` Entità top del progetto, cioè che contiene tutte le altre entity e le istanzia nel modo corretto, impostando le frequenze di lavoro dei componenti.

5.3 Contatore Generico



L'entity `counter_impulse_generator` implementa un contatore generico. Facendo uso dei parametri generici del VHDL è possibile impostare per ogni istanza del componente una diversa frequenza di reset. Il contatore utilizza un registro che viene incrementato ad ogni ciclo di clock e la sua dimensione in bit è calcolata automaticamente a partire dalle frequenze di reset e di clock. Il valore del registro è esposto alla porta `o_counter`. Al reset del contatore il segnale di output `o_signal` viene portato al valore logico alto per un ciclo di clock. Si utilizza questo segnale per abilitare i componenti che devono effettuare operazioni a una frequenza inferiore alla frequenza di clock: ad esempio per pilotare il processo di campionamento si fa uso di un contatore avente frequenza di reset uguale alla frequenza di campionamento `sampling_frequency`.

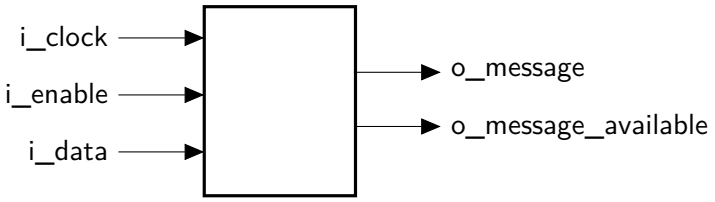
5.4 Blocco UART



Il blocco uart è composto da una finite state machine (fsm) e uno shift register da 9 bit, contenente inizialmente i valori '0' tranne per il bit più significativo che è posto a '1' e funge da sentinella. La lettura della linea dati avviene periodicamente alla frequenza $f = \frac{1}{T_{bit}} = 31250\text{Hz}$; alla ricezione dello start bit, la fsm entra nello stato di ricezione e si fa avanzare lo shift register leggendo il valore presente sulla linea nel bit più significativo,

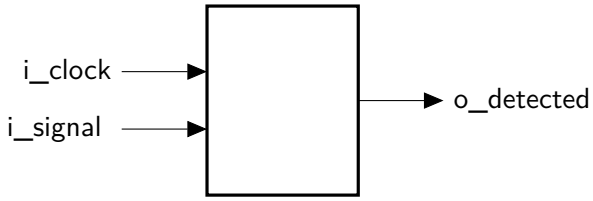
facendo contemporaneamente slittare gli altri verso destra. A lettura ultimata, lo shift register è avanzato di 8 posizioni e il bit meno significativo dello shift register contiene il bit sentinella; con questo si indica la fine della trasmissione.

5.5 MIDI decoder



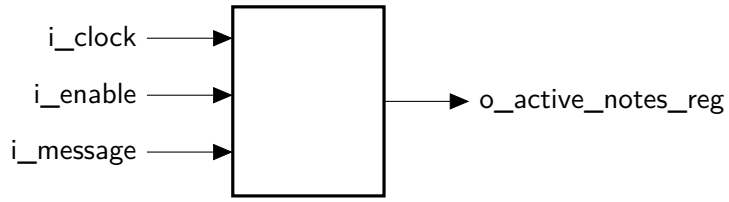
Il decoder MIDI si occupa di leggere i byte provenienti dal convertitore seriale/parallelo e di convertirli in messaggi MIDI. In realtà il componente è molto semplificato in quanto interpreta solo i messaggi di note on e note off, mentre scarta ogni altro messaggio MIDI. Il blocco è implementato attraverso una fsm; quando l'input `i_enable` viene mantenuto alto per un ciclo di clock, la state machine avanza leggendo il byte `i_data` in input. Quando il messaggio è stato completamente decodificato, l'output `o_message_available` assume un valore logico alto e il messaggio MIDI è disponibile in uscita al blocco.

5.6 Low to High detector



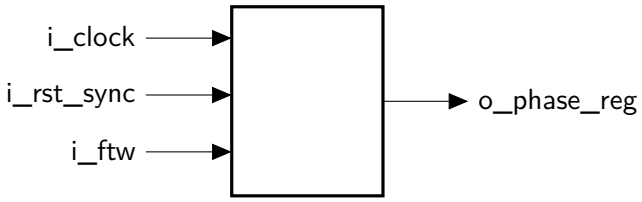
Quando l'entrata `i_signal` passa dal valore logico basso a quello alto, `o_detected` assume il valore logico alto per un singolo ciclo di clock. Un esempio di utilizzo di questo componente è la connessione tra i componenti uart e midi_decoder: il blocco `midi_decoder` avanza la macchina a stati leggendo dall'ingresso quando la porta `i_enable` si trova al valore logico alto. Se questa fosse collegata direttamente all'uscita `o_data_available` del blocco `uart`, lo stesso byte verrebbe processato più volte non permettendo una decodifica corretta. Si intermezza questa connessione con un componente `low_to_high_detector` cosicché alla presenza di un nuovo byte il decoder midi venga abilitato solo per un ciclo di clock. Nonostante la sua semplicità questa entity permette di semplificare altre situazioni in cui questo avviene all'interno del progetto.

5.7 Active Notes LUT



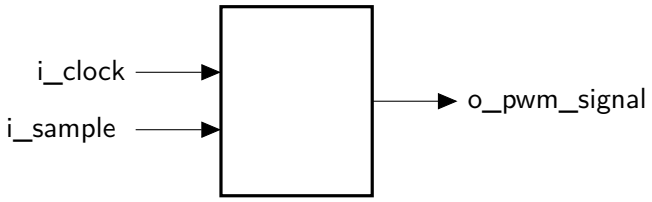
Quando il segnale `i_enable` è alto per un ciclo di clock, il valore del messaggio midi in input viene letto e il registro `o_active_notes_reg` viene modificato in modo che il bit in posizione corrispondente alla nota sia 1 o 0 a seconda del contenuto del messaggio.

5.8 Accumulatore di Fase



L'accumulatore di fase implementa il registro di fase utilizzato nella DDS. La grandezza del registro è impostabile attraverso il parametro generico `phase_bits`. È possibile il reset sincrono del registro impostando al valore logico alto il segnale `i_rst_sync`.

5.9 PWM Encoder



Il `pwm_encoder` funziona nel seguente modo: all'interno dell'entity è istanziato un registro contatore che viene incrementato ad ogni ciclo di clock e ritorna nullo dopo K cicli di clock, dove

$$K = \left\lfloor \frac{T_s}{T_{clk}} \right\rfloor = 2048$$

uguale ai 2^b possibili valori che può assumere un campione. Si ha quindi

$$K = 2^b = \texttt{i\_sample\_max} + 1$$

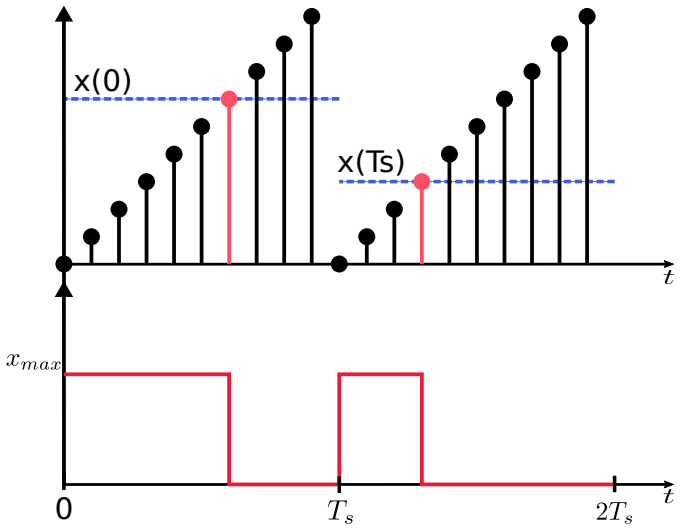


Figura 5.1: Esempio di PWM: nel grafico sopra è riportato il valore del campione $s(kT_s)$ - tratteggiato in blu - che viene comparato al contatore. Quando il contatore supera il valore di $s(kT_s)$, `o_pwm_signal` passa da x_{max} a 0.

Dopo $T_{clk}K \approx T_s$ il contatore si riazzera e si ha un nuovo campione in ingresso. Il valore del campione `i_sample` viene confrontato con quello del registro; se questo è maggiore del valore del contatore, l'uscita viene portata in alta impedenza, mentre se questo è minore l'uscita viene portata a zero. L'uscita `o_pwm_signal` pilota l'uscita audio monofonica e viene posta ad alta impedenza invece che al valore logico alto per permettere al circuito di gestire autonomamente il voltaggio di uscita. La frazione di tempo per cui `o_pwm_signal` rimane in alta impedenza in un intervallo di campionamento è di

$$\frac{T_{clk}\texttt{i\_sample}}{KT_{clk}} = \frac{\texttt{i\_sample}}{\texttt{i\_sample\_max} + 1}$$

Detto v_{gain} il voltaggio massimo fornito al filtro passa-basso, si vede facilmente che la media sul periodo del segnale è

$$v_{gain} \cdot \frac{\texttt{i\_sample}}{\texttt{i\_sample\_max} + 1} \approx v_{gain} \cdot \frac{\texttt{i\_sample}}{2^{\texttt{b}}} \approx v_{gain} s(kT_s)$$

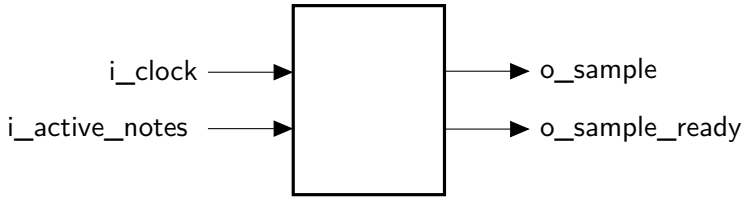
Dove si è usata all'ultimo passaggio l'eq. (4.4), ignorando gli errori dovuti alla DDS e alla quantizzazione. È chiaro quindi che il segnale dato in ingresso viene amplificato e ricostruito dal filtro in base a quanto detto nella sezione 4.2.

5.10 Componente ROM

Le ftw e la forma d'onda da riprodurre vengono generate attraverso due script python e salvate in un file. Il package generico rom permettere al progetto di avere queste informazioni in fase di sintesi: si specificano i tre parametri `word_bits`, che indica il numero di bit per parola nella memoria, `address_bits`, che indica il numero di bit necessari a indirizzare una parola, e `rom_filename`, che specifica il file testuale da leggere contenente le parole della rom. Il file deve essere formattato nel seguente modo: ci sono tante righe quanti sono gli indirizzi della memoria, ogni riga è una stringa binaria di lunghezza fissa e rappresenta la parola memorizzata. Gli indirizzi partono da 0 (prima riga) e aumentano di un unità ad ogni riga. È possibile leggere dalla ROM usando la funzione `read_at`, specificando come argomento l'indirizzo desiderato.

```
package rom is
  generic (
    word_bits: positive;
    address_bits: positive;
    rom_filename: string
  );
  subtype word_type is std_logic_vector(word_bits-1 downto 0);
  impure function read_at(
     address: in integer range 0 to 2**address_bits-1
  ) return word_type;
end package rom;
```

5.11 Synth Engine



Il componente principale del sintetizzatore ha accesso alla LUT contenenti le note attive e genera ad ogni periodo di campionamento un campione del segnale.

Attraverso il package rom vengono rese disponibili a questo componente la forma d'onda campionata e le frequency tuning word corrispondenti ad ogni possibile nota suonata. Attraverso il costrutto `for .. generate` vengono generati 128 accumulatori di fase indipendenti. Ad ognuno di questi viene accoppiata una entity che ne esegue il reset quando una nota passa dal

valore logico basso a quello alto nella LUT (si noti l'utilizzo del componente `low_to_high_detector`).

```
oscillators:
for i in 0 to MAX_MIDI_NOTE_NUMBER generate
    signal ftw: std_logic_vector(phase_bits-1 downto 0);
    signal reset_signal: std_logic := '0';
begin
    ftw <= phase_rom.read_at(i);
    reset_generator: entity work.low_to_high_detector
    port map (
        i_clock => i_clock,
        i_signal => i_active_notes(i),
        o_detected => reset_signal
    );
    accumulator: entity work.phase_accumulator
    generic map (
        clock_frequency => clock_frequency,
        phase_bits => phase_bits
    )
    port map (
        i_clock => i_clock,
        i_rst_sync => reset_signal,
        i_ftw => ftw,
        o_phase_reg => phase_vec(i)
    );
end generate oscillators;
```

Il campione viene fornito alla porta `o_sample` e viene generato attraverso un processo di mixing. Il mixing consiste nel seguente processo, ripetuto per ogni valore della LUT delle note attive:

- Se la nota è attiva, si carica il valore campionato della forma d'onda ad essa relativa: il valore campionato viene ottenuto dalla ROM dei campioni, usando come indice i primi `M` bit dell'accumulatore di fase: infatti questo è il **phase to amplitude converter** (PAC)
- Il valore del campione viene sommato su un registro inizialmente nullo

Come verrà chiarito nella sezione 5.12 l'implementazione fa uso di una pipeline a due stadi: il primo stadio della pipeline provvede alla PAC, mentre il secondo stadio esegue parallelamente il mixing con il valore ottenuto dallo stadio precedente.

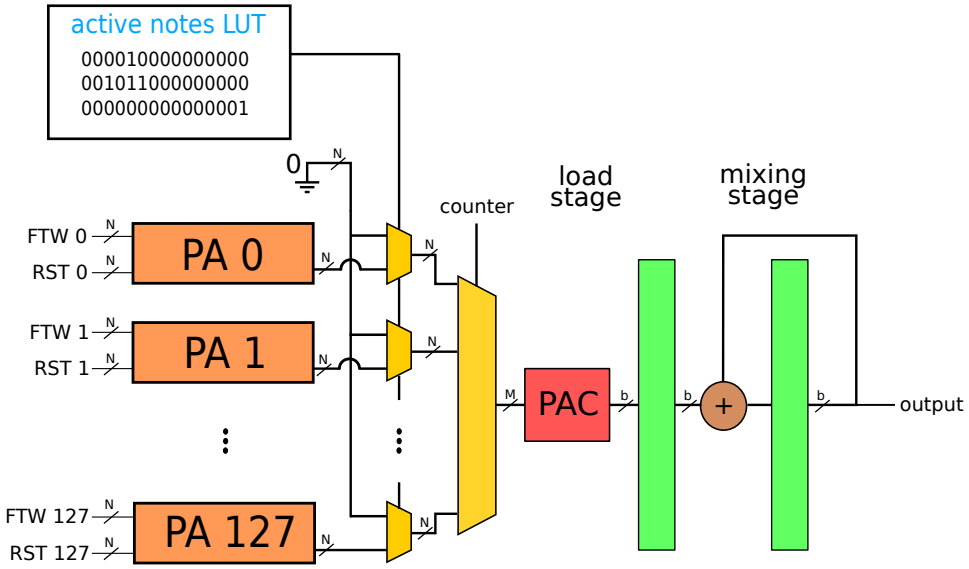


Figura 5.2: Architettura con pipeline a due stadi. Ogni multiplexer in uscita ad un phase accumulator (PA) controlla se la nota è attiva nella LUT; il multiplexer principale, comandato da un contatore, carica la pipeline con un nuovo campione. Seguono poi lo stadio di PAC (ossia il caricamento del campione dalla ROM) e il mixing.

Ogni avanzamento della pipeline viene eseguito in un ciclo di clock, e in totale sono necessari 129 cicli di clock (128 per ogni nota, più uno per caricare la pipeline) che vengono eseguiti appena prima della fine del periodo di campionamento. Questo avviene all'ultimo ciclo di clock della fase di produzione del campione, in cui l'output `o_sample_ready` assume il valore logico alto per indicare la presenza di un campione.

5.12 Difficoltà di implementazione e vincoli temporali

Nel corso dello sviluppo del progetto uno dei problemi più difficili da affrontare è stato quello del rispetto dei vincoli temporali nell'implementazione dell'engine del sintetizzatore.

Inizialmente questo era stato implementato come un componente asincrono dotato di 128 ingressi collegati alla LUT delle note attive e 128 ingressi da `sample_bits`, ciascuno collegati direttamente alla memoria dei campioni. In totale il componente aveva quindi $128 + 128 \cdot 11 = 1536$ bit in ingresso, senza contare il sommatore e i multiplexer necessari alla selezione delle note. Data la complessità del componente, questo veniva sintetizzato con una

lunga catena di sommatori a due ingressi collegati in cascata; tuttavia le catene di propagazione più lunghe non rispettavano i vincoli temporali,e si otteneva un WNS di $-1.684\,\text{ns}$.

Si è quindi convertito il mixer in un componene sincrono che sommasse ogni campione ad un ciclo di clock; tuttavia anche in questo caso si otteneva un WNS negativo pari a $-0.738\,\text{ns}$. Una soluzione si sarebbe potuta ottenere dilatando il tempo tra due operazioni di somma successive, impostando i parametri di progetti in modo da allentare i timing constraints su queste path. Un'altra soluzione simile sarebbe quella di usare un clock diverso per il processo di mixing. Entrambe le soluzioni tuttavia richiedevano una conoscenza approfondita di Vivado, e la seconda soluzione necessita della comunicazione tra componenti aventi clock diversi, detto *clock domain crossing*, e molto complesso da implementare.

Notando che la path che non rispettava i vincoli temporali attraversava il sommatore, si è deciso di rendere l'operazione di caricamento del campione e quella di somma distinte, formando così l'architettura a pipeline a due stadi descritta prima.

Così facendo il WNS diventa di $1.911\,\text{ns}$ e i vincoli temporali sono rispettati.

Capitolo 6

Analisi e conclusioni

Il testing del progetto sulla scheda Nexys4 DDR è stato condotto attraverso la porta seriale della scheda e mediante l'uso di uno script Python che automatizzasse la riproduzione di un file MIDI (si veda `scripts/play_midi_file.py`). Per effettuare il testing funzionale delle varie componenti del progetto si è fatto uso di *testbenches* scritti in VHDL, che fornissero un input prefissato a un certo componente per verificarne la risposta, utilizzando sempre GHDL. In particolare il testbench descritto nel file `vhdl/mono_synth_engine_tb.vhd` provvede a testare l'intera architettura riproducendo una singola nota. L'uscita del blocco `pwm_encoder` viene quindi salvata ad ogni ciclo di clock su un file txt, che viene in seguito processato dallo script `scripts/plot_pwm_output.py` che lo filtra digitalmente e visualizza il segnale ricostruito.

L'intero codice del progetto, compresi gli script, i testbench e questa tesi è disponibile su github [1].

Data la complessità computazionale di una simulazione, il testbench di prova dell'intero progetto può impiegare fino a diversi minuti per generare un output che realmente avrebbe durata nell'ordine della decina di millisecondi, tempo insufficiente a fare analisi significative sul segnale ottenuto. Si è quindi optato per la simulazione della sintesi DDS, della conversione in PWM e della ricostruzione del segnale attraverso uno altro script python, con cui si sono valutate le prestazioni del progetto.

6.1 Rapporto Segnale Rumore

Sia $s(t)$ il segnale da generare e $s'(t)$ il segnale ottenuto dal processo di sintesi. Si definisce il *segnale d'errore* $e(t) = s(t) - s'(t)$. Per valutare la qualità del segnale d'uscita rispetto al segnale di ingresso si fa uso del cosiddetto *signal to noise ration* (SNR), definito da [7]

$$\texttt{SNR} = \frac{P_s}{P_e} \tag{6.1}$$

Dove P_s e P_e sono la potenza del segnale e del segnale d'errore, che per segnali reali vale [6]

$$P_s = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} s^2(t)dt \tag{6.2}$$

Per misurare l'SNR si fa spesso l'utilizzo dei decibel, per cui

$$\texttt{SNR}_{\text{dB}} = 10 \log_{10} \texttt{SNR} \tag{6.3}$$

6.2 Errore di quantizzazione

La prima causa di errore nel segnale ricostruito si deve al processo di quantizzazione: il processo di quantizzazione descritto nella sezione 4.2 si dice quantizzazione uniforme. Dato l'intervallo di valori assunti da $x(t)$ pari a $[-x_{max}, x_{max}] = [-1, 1]$ e il numero di valori quantizzati $L = 2^{\texttt{b}}$ (numero di livelli del quantizzatore) si definisce il passo di quantizzazione

$$\Delta = \frac{2x_{max}}{L}$$

Si dimostra che se il segnale $s(t)$ rimane all'interno del range stabilito, l'errore di quantizzazione ha una potenza pari a [7]

$$P_e = \frac{\Delta^2}{12}$$

Per il testing di questo progetto si è scelto di utilizzare $A = 0.4$ e la forma d'onda sinusoidale, cosicché il valore massimo di due segnali sovrapposti non ecceda la soglia di quantizzazione e non si vada in saturazione con la polifonia a due voci. Come è noto la potenza di un segnale periodico è uguale alla potenza sul periodo e nel caso di un segnale $s(t) = A\sin(2\pi ft)$ è pari a

$$P_s = \frac{A^2}{2}$$

L'SNR dovuto all'errore di quantizzazione con $\texttt{b} = 11$ è pari a $\texttt{SNR}_{\text{dB}} = 60\,\text{dB}$.

La quantizzazione di un segnale di ampiezza minore di quella massima permessa dal quantizzatore peggiora la qualità del segnale, infatti se si fosse usato $A = 1$ si sarebbe ottenuto $\texttt{SNR}_{\text{dB}} = 67.8\,\text{dB}$, andando tuttavia incontro a problemi overflow in caso di polifonia che annullerebbero la qualità guadagnata.

6.3 Errore della PWM

Come si è ricavato nella sezione 4.3, un'onda quadra di duty cycle costante d e valore tra 0 e x_{max} produce, dopo un adeguato filtro passabasso, un segnale costante avente il valore dx_{max}. Quando il duty cycle cambia ad un nuovo valore (ossia alla codifica di un nuovo campione) si avrà un transitorio prima che la risposta in uscita al filtro si assesti sul nuovo valore. Intuitivamente si capisce che la qualità del segnale ricostruito dipende da quanto velocemente può variare il segnale rispetto alla frequenza di transizione della PWM.

La condizione per la ricostruzione esatta del segnale PWM si ricava dall'espressione analitica del suo spettro, molto complessa da ricavare e per cui si rimanda a [8]. Date $f_{pwm} = 1/T_{pwm}$, definita del periodo del segnale PWM, e la frequenza massima del segnale f_{max} si dimostra che

$$f_{pwm} > \pi f_{max} \tag{6.4}$$

è la condizione di ricostruzione del segnale PWM.

Essendo $f_{pwm} = f_s$ nel nostro progetto, si ricava che la frequenza massima del segnale è di poco più di 15 kHz, inferiore alla frequenza massima che ci si era impostati di riprodurre. Nel caso di segnali aventi contenuto armonico maggiore di questa frequenza, questo non verrà ricostruito accuratamente; nel caso delle sinusoidi usate per il testing il problema non si verifica poiché la nota più acuta, corrispondente al note number 127, ha una frequenza di 12 543 Hz.

6.4 Errore dovuto alla DDS

La processo di sintesi digitale diretta introduce due errori: il primo dovuto alla quantizzazione della fase, il secondo dovuto alla capacità ridotta della ROM e alla necessità di indirizzarla con i primi `M` bit del registro di fase. Usando un numero `N` di bit elevato per il registro di fase si può considerare trascurabile l'errore dovuto alla quantizzazione della fase, mentre non è trascurabile il secondo tipo di errore detto **errore di troncamento di fase**.

Data una qualsiasi `FTW`, il valore del registro i fase sarà periodico di un numero di cicli K pari a [9]

$$K = \frac{2^{\mathtt{N}}}{\mathrm{GCD}(\mathtt{FTW}, 2^{\mathtt{N}})}$$

Gli ultimi $N - M$ bit del registro di fase saranno anch'essi periodici di un numero di cicli al massimo uguale a K. Essendo questi a determinare l'errore di troncamento di fase, si evince che il segnale d'errore avrà un andamento periodico nel dominio del tempo, a causerà nel dominio della frequenza la presenza di frequenze spurie.

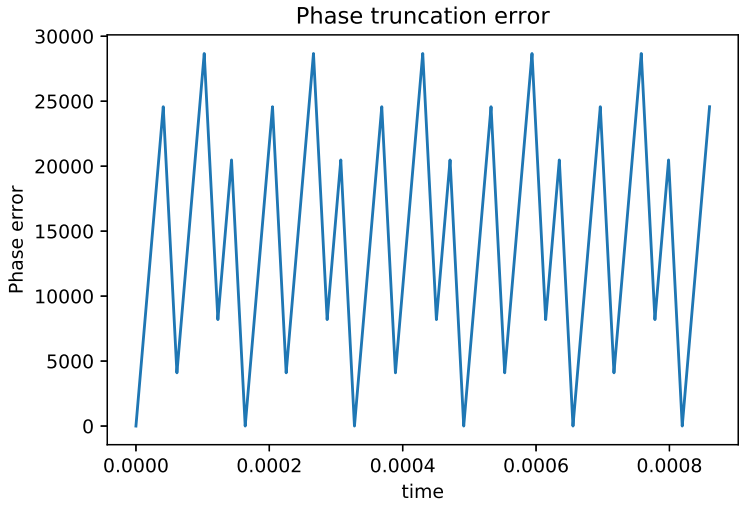


Figura 6.1: Periodicità dell'errore di troncamento di fase corrispondente alla `FTW` $= 63565$, con `N` $= 32$ e `M` $= 17$.

In generale quindi l'errore introdotto dalla DDS dipende da M e N ed è diverso per ogni scelta della `FTW`. Si veda [9] oppure [10] per una trattazione più dettagliata della distribuzione e ampiezza delle frequenze spurie nel segnale, omessa in quanto complessa dal punto di vista matematico.

La riduzione di SNR dovuta alla DDS è stata calcolata, attraverso le simulazioni, per ogni possibile valore del *note number* ed è riportata in fig. 6.2.

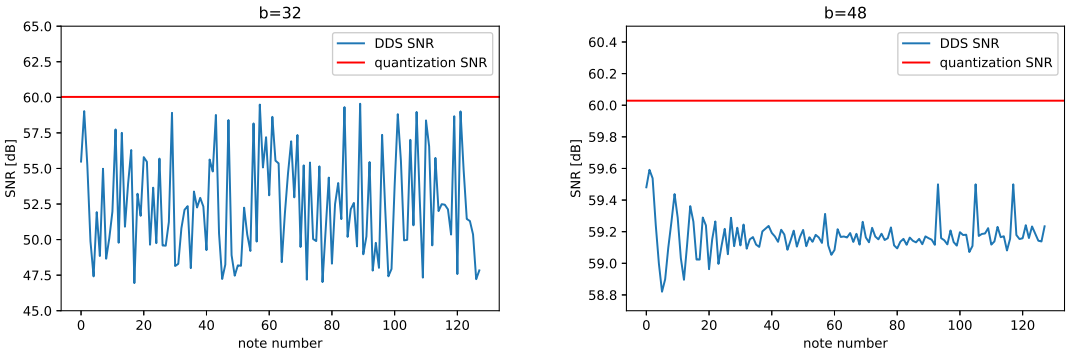


Figura 6.2: Confronto dell'SNR dovuto alla DDS con due grandezze diverse del registro di fase.

Per `N` $= 32$ l'SNR decresce fino a circa 47 dB per alcune frequency tuning words. Usando una precisione maggiore per il registro di fase si riesce

ad ottenere un SNR al minimo di 58.8 dB, molto vicino a quello di sola quantizzazione.

6.5 Considerazioni finali

Il sintetizzatore progettato, sebbene funzionante, presenta performance di SNR non ottimali, dovute alle limitazioni intriseche della pulse-width modulation. La PWM infatti limita il numero di bit di risoluzione del campionatore da un lato e la frequenza massima rappresentabile dall'altro. Sarebbe possibile bypassare l'uscita audio monofonica della scheda e utilizzare un DAC esterno, connesso attraverso le porte PMOD, per fornire una qualità audio migliore. Due esempi sono i componenti PMOD I2S2 e il PMOD AMP2 della Digilent.

La polifonia, essendo implementata con un semplice sommatore in fase di mixing, è soggetta ad overflow quando la somma dei campioni supera la precisione permessa dai `b` bits di quantizzazione. In campo audio questo fenomento viene detto *clipping*. Di fatto, essendo l'ampiezza del segnale campionato pari a poco meno della metà del valore massimo rappresentabile, la polifonia è limitata a due voci, oltre le quali il segnale presenta una degradazione notevole.

Far rientrare il range del segnale nei limiti con un divisore risolverebbe il clipping ma degraderebbe la qualità del segnale e causerebbe sbalzi di volume udibili nel passaggio da una singola nota a più note suonate contemporaneamente.

Tecniche più avanzate di compressione dinamica del segnale sono molto complesse e al di fuori dello scopo di questa tesi.

I messaggi MIDI contengono anche l'informazione relativa alla forza di pressione della nota, detta velocity, come descritto nel capitolo 2. Si potrebbe estendere il sintetizzatore per rispettare questa informazione in due modi differenti:

- Si stabiliscono dei range di velocity meno granulari e per ognuno di questi si accede a una rom diversa creata impostando una diversa ampiezza alla forma d'onda
- Si stabiliscono sempre dei range di velocity, ma lo scaling dei campioni avviene con un divisore

Il primo metodo produrrebbe risultati più precisi numericamente ma al prezzo di una notevole quantità di memoria in più, mentre il secondo otterrebbe una qualità peggiore ma con un'occupazione minore delle risorse del FPGA.

Il sintetizzatore è stato operato attraverso un computer per il testing, ma sarebbe possibile connettere la scheda direttamente utilizzando una porta PMOD generica e un circuito esterno (anche su breadboard) a cui connettere il jack MIDI.

Bibliografia

[1] Codice sorgente del progetto, testbench, latex e scripts. Link Github.

[2] The MIDI association. *The MIDI 1.0 Specification*

[3] Xilinx. *7 Series FPGAs Data Sheet: Overview*.

[4] Xilinx. *7 Series FPGAs Configurable Logic Block*

[5] Digilent. *Nexys4 DDR FPGA Board Reference Manual*

[6] Oppenheim, Allan V. and Willsky, Alan S. and Nawab, S. Hamid *Signals & Systems (2Nd Ed.)*. Prenctice-Hall, Inc., 1996.

[7] Nevio, Benveuto. *Principles of Communications Networks and Systems*. Wiley, 2011.

[8] Song, Sarwate. *The Frequency Spectrum of Pulse Width Modulated Signals*. Signal Processing Volume 83, Issue 10, October 2003, Pages 2227-2258.

[9] Analog Devices. *A Technical Tutorial on Digital Signal Synthesis*. Section 4.

[10] Torosyan, Willson. *Exact analysis of DDS spurs and SNR due to phase truncation and arbitrary phase-to-amplitude errors* Proceedings of the 2005 IEEE International Conference. Frequency Control Symposium and Exposition, 2005.