Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Elettronica

Progetto di un sistema di calibrazione di frequenza per un oscillatore integrato

Laureando:
Davide Pizzato

Relatore:
Prof Andrea Neviani

Anno accademico 2013/2014 - Padova, 14 Aprile 2014

Sommario

La radio a impulsi UWB (UWB-IR) è una tecnologia di trasmissione che può essere implementata con architetture relativamente semplici e con bassi consumi di energia. Per questi motivi sono ampiamente utilizzate in applicazioni wireless portatili che richiedono infatti ricetrasmettitori semplici e di ridotti consumi, per una più lunga durata di funzionamento.
La tecnologia UWB-IR trasmette le informazioni con impulsi di brevissima durata ad una data frequenza, tipicamente tra 3.1 e 10.6 GHz. La parte di generazione della frequenza di trasmissione è affidata ad oscillatori LC integrati (per un'occupazione di area ridotta). Questo tipo di oscillatori ha la necessità di avere un controllo sulla frequenza di oscillazione in quanto, a causa del processo di fabbricazione, le capacità possono subire una grande variazione del loro valore.
Per ridurre il problema viene introdotto un circuito di calibrazione che, creando un anello di feedback con l'oscillatore, misura e corregge la frequenza di trasmissione.
Questa tesi presenta l'implementazione di un circuito di calibrazione per un oscillatore controllato digitalmente (DCO) che abbia un ridotto consumo di energia, parametro fondamentale per un trasmettitore UWB-IR.
Il circuito di calibrazione proposto è composto da un misuratore di frequenze e dall'implementazione di un algoritmo di ricerca. Le ricerche prese in esame sono quelle: sequenziale, binaria e per interpolazione.
Tramite la simulazione a livello di porte logiche del circuito è stato stimato il consumo di energia per ogni algoritmo. Con questi dati è risultato che il circuito di calibrazione che esegue l'operazione con il minor consumo di energia è quello che implementa la ricerca binaria.

Indice

Capitolo 1

Radio a impulsi UWB (UWB-IR)

Lo sviluppo di nuove applicazioni che utilizzano reti di sensori wireless (WSN), ad esempio la localizzazione di oggetti o persone e la dispositivistica portatile, hanno portato alla realizzazione di trasmettitori e ricevitori di dimensioni più piccole e con consumi inferiori [1].
L' Ultra-wideband (UWB) è una tecnica di comunicazione che utilizza segnali con una maggiore banda di frequenza, definita dalla FCC (Federal Communications Commission) tra 3.1 e 10.6 GHz, rispetto alle tecniche di comunicazione tradizionali quali GSM, GPS e Bluetooth. Inoltre la potenza consentita per questo tipo di comunicazioni è molto bassa, $-41.3dBm/MHz$, rendendo questi sistemi ottimali per le applicazioni wireless. I sistemi che utilizzano la trasmissione UWB possono essere divisi in due categorie: Orthogonal Frequency Domain Multiplexing (OFDM-UWB) e Impulse Radio (IR-UWB).
La prima categoria trasmette le informazioni attraverso la divisione della banda in porzioni potendo sfruttare così un approccio tradizionale per i ricetrasmettitori.
Il sistema UWB-IR invece, occupa una grande porzione di banda attraverso impulsi di breve durata (e quindi di ampio spettro) [2].
Tra le due opzioni, la preferita è la radio a impulsi UWB in quanto prevede una struttura meno complessa e un minor consumo energetico [3] [4].
Le principali difficoltà che si incontrano nella realizzazione di una radio a impulsi sono la realizzazione di circuiti che generino impulsi di brevissima durata, dell'ordine dei ns, mantenendo un basso consumo e dimensioni ridotte. Per ridurre questi problemi si opta per una realizzazione della radio a impulsi completamente integrata. Inoltre si tende a costruire generatori di impulsi in grado di trasmettere a frequenze molto alte, tra 6 e 10.6 GHz, per ridurre le interferenze delle trasmissioni WiMax e Wi-Fi nella parte bassa della banda UWB.

1.1 Trasmettitori a impulsi

Un trasmettitore a impulsi per le comunicazioni UWB, come si può vedere in Figura 1.1, è composto da un generatore di impulsi e da un amplificatore di potenza. Gli impulsi possono essere generati con diverse modalità, ad esempio:

- modulazione dell'impulso [5];
- sintesi della forma d'onda [2] [3];
- oscillatori LC[1].

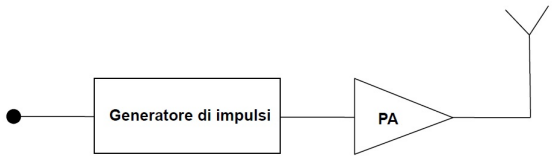

Figura 1.1: Schema di un trasmettitore a impulsi.

Il generatore ad impulsi realizzato tramite la modulazione dell'impulso (Figura 1.2.a) è formato da una porta logica in cui, in ingresso, sono presenti una serie di invertitori. Quando arriva il segnale di abilitazione, l'ingresso non ritardato cambia valore, mentre l'altro dovrà attendere un periodo di ritardo dipendente dagli invertitori. Durante questo tempo si ottiene una differenza di segnale all'ingresso della porta che genera un impulso di durata pari al tempo di ritardo. Il segnale così creato deve essere però filtrato adeguatamente per rispettare i requisiti dei sistemi UWB.
Un secondo metodo di generazione di impulsi è attraverso la sintesi della forma d'onda (Figura 1.2.b). Questo tipo di generatori utilizza una memoria, un convertitore digitale analogico (DAC) e un filtro per generare l'impulso voluto. All'interno della memoria è salvata la forma d'onda da realizzare, essa viene convertita in un segnale analogico da un DAC ad alta velocità e successivamente modellata da un filtro, per la ricostruzione completa dell'impulso. Il terzo tipo di generatori di impulsi (Figura 1.3) sono realizzati con un oscillatore e un circuito di controllo responsabile della generazione dell'impulso [1]. L'oscillatore genera, all'arrivo del segnale di abilitazione, un segnale alla frequenza di trasmissione che viene modulato con un impulso. Questo impulso deriva dal segnale dell'oscillatore stesso, diviso di una quantità nota (in questo caso 16).
Quest'ultimo tipo di circuiti sono più semplici da realizzare in quanto non necessitano di complessi filtri in uscita, come nella prima tipologia, o di circuiti particolarmente ottimizzati (i DAC ad alta velocità del secondo tipo di circuiti).

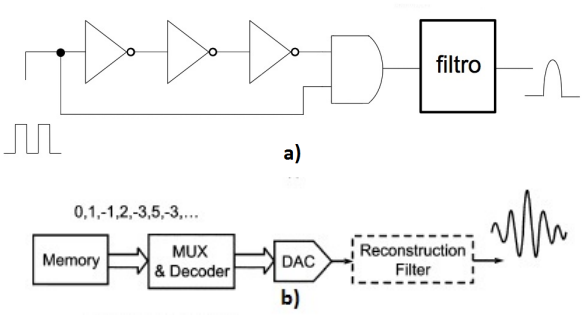

Figura 1.2: Esempi di generatori a impulsi digitali: a) modulazione dell'impulso e b) sintesi della forma d'onda [3].

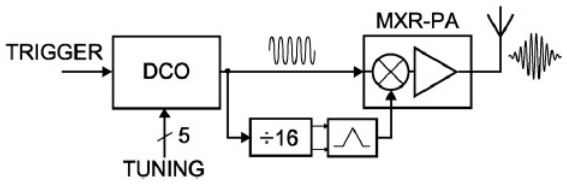

Figura 1.3: Esempio di generatore a impulsi con DCO [1].

1.2 DCO

Gli oscillatori controllati digitalmente (DCO) sono implementati tramite un oscillatore differenziale con tank LC e consentono di variare frequenza tramite un banco di capacità.
L'oscillazione viene generata con l'invio di un segnale di abilitazione, questo segnale crea uno sbilanciamento dei nodi di uscita dell'oscillatore che attiva la risonanza LC. La frequenza di oscillazione dipende quindi dai valori dell'induttore e del condensatore e risulta: $F_{DCO} = \frac{1}{2\pi}\frac{1}{\sqrt{LC}}$.
L'integrazione di questi componenti passivi richiede un'area molto elevata del circuito quindi, per poter ridurre il peso nell'economia del circuito di questi elementi, i condensatori vengono realizzati attraverso due soluzioni:

- giunzioni *pn*;
- strutture MOS.

La prima soluzione utilizza una giunzione *pn* polarizzata inversamente, ottenendo una tensione di breakdown elevata a discapito di valori molto bassi di capacità. La seconda soluzione è quella utilizzata più comunemente e realizza il condensatore con una struttura di tipo metallo-ossido-semiconduttore (MOS).
I valori di queste capacità hanno una grande variabilità, essa infatti può raggiungere il 20% del valore nominale, a causa di capacità parassite, errori

sulle dimensioni degli strati drogati e nelle formule approssimate utilizzate in fase di progettazione.
Questa grande variabilità ha introdotto la necessità di realizzare banchi di condensatori programmabili che, operando variazioni di capacità, permettono di correggere la frequenza di oscillazione al valore desiderato. L'introduzione di questi banchi di capacità programmabili ha portato alla realizzazione di circuiti di calibrazione.

1.3 Circuiti di calibrazione

I circuiti di calibrazione fungono da controllo della frequenza dell'oscillatore e sono inseriti in un circuito di feedback (Figura 1.4). Esistono diverse tipologie di circuiti di calibrazioni e i più utilizzati sono:

- circuiti ad aggancio di fase, o PLL (Figura 1.5);
- circuiti digitali con ricerca.

I PLL ricevono il segnale dall'oscillatore e lo comparano con il segnale alla frequenza desiderata. L'uscita dal comparatore viene successivamente filtrata e amplificata ottenendo il segnale di controllo in ingresso all'oscillatore. Nel caso in cui l'oscillatore sia realizzato tramite DCO occorre prevedere un converitote analogico digitale (ADC) che produca i bit di configurazione.
I DCO, grazie alla struttura del banco di capacità, favoriscono la realizzazione di circuiti digitali di calibrazione. Essi sono composti da:

- un contatore di frequenza;
- un circuito di ricerca per i codici del banco di capacità.

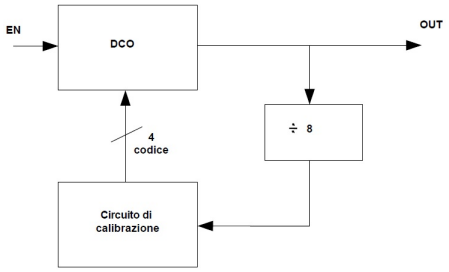

Figura 1.4: Circuito del generatore a impulsi con il controllo.

Il contatore di frequenza misura la frequenza di oscillazione proveniente dal DCO, il dato viene elaborato a seconda dell'algoritmo di ricerca scelto e viene fornito in uscita il codice relativo alla prossima frequenza da misurare o, in caso di termine ricerca, il codice finale.

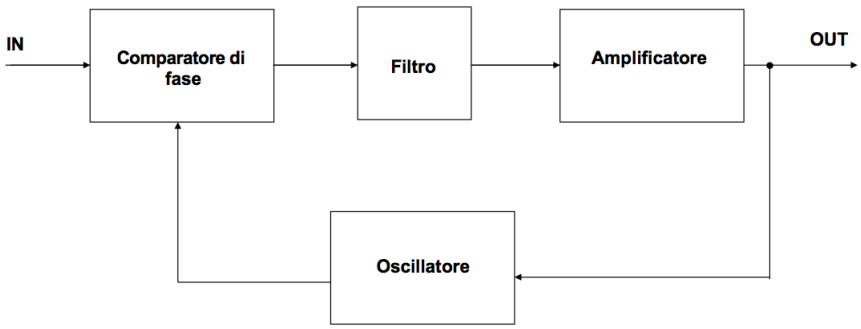

Figura 1.5: Circuito ad agganciamento di fase.

Gli algoritmi di ricerca sono diversi e risultano ottimali a seconda dell'obiettivo scelto (ad esempio la velocità o il consumo). In questo lavoro sono stati presi in considerazione tre algoritmi:

- sequenziale;
- binario [6] [7];
- per interpolazione [8].

La ricerca sequenziale è la più semplice da implementare, ma ha un tempo di ricerca che dipende dal numero di elementi. La ricerca binaria ha tempi ridotti di ricerca, dipendenti dal logaritmo del numero di elementi, a discapito di un aumento della difficoltà di implementazione. L'ultima ricerca valutata riduce ulteriormente i tempi di ricerca, come verrà spiegato nel capitolo 3, aumentando corrispondentemente la difficoltà di realizzazione.
L'obiettivo di questa tesi è quello di implementare un circuito di calibrazione integrato per il controllo del generatore a impulsi di un UWB-IR, costituito da un DCO, che abbia un consumo energetico ridotto.
Nel successivo capitolo sarà esposto in dettaglio il trasmettitore a impulsi da cui verranno estratte le specifiche utilizzate per la realizzazione del circuito di calibrazione, nel capitolo 3. Nel capitolo 4 viene effettuata la verifica delle specifiche e l'analisi dell'energia per le tre diverse soluzioni. Nel capitolo finale vengono fatte alcune considerazioni sulla scelta dell'algoritmo ottimale per questo circuito.

Capitolo 2

Specifiche

Il trasmettitore a impulsi utilizzato in questo lavoro, a cui sarà collegato il circuito di calibrazione, si basa sul circuito esposto nell'articolo [1].
Questo trasmettitore opera con impulsi di banda 1.25 GHz centrati ad una frequenza di 8 GHz, questi valori permettono di ridurre le interferenze e sono anche l'unico range all'interno del quale le radio ad impulsi UWB possono operare in Europa, Usa e Giappone.
In base alla scelta del data rate massimo a $R_b = 5Mb/s$ e alla scelta della massima densità spettrale di potenza (PSD), imposta dalla FCC a -41.3 dBm/MHz all'interno della banda che va da 3.1 a 10.6 GHz, si ottiene l'energia di un impulso ($E_{pTx} = 18.5pJ$) tramite la formula:

$$E_{pTx} \leq \frac{S_{FCC}B}{R_b} \tag{2.1}$$

Dalle misurazioni finali viene determinato il costo totale di energia per la trasmissione di un impulso che corrisponde a $E_{dTx} = 0.2nJ/impulso$.
Il circuito di calibrazione da implementare deve controllare l'oscillatore responsabile della generazione della frequenza di trasmissione. Tale oscillatore è realizzato da: un DCO, un divisore di frequenze (prescaler) per 8 e da un mixer. La generazione dell'impulso avviene con l'invio di un segnale di abilitazione del DCO, successivamente il prescaler divide la frequenza di oscillazione creando un segnale che, combinato con il segnale proveniente dal DCO, genera un impulso di durata ridotta centrato alla frequenza di oscillazione del DCO.
Il DCO è stato implementato utilizzando un oscillatore differenziale LC con la capacità realizzata da un banco programmabile a 4 bit. Il circuito di calibrazione opererà su questi bit per l'oscillazione del DCO alla frequenza di trasmissione.

2.1 Range di frequenze

Come specificato in precendenza il generatore ad impulsi utilizzato in questo lavoro è costituito da:

- un DCO;
- un prescaler;
- un mixer.

Il prescaler divide la frequenza del DCO per 8 generando un segnale che modulato con quello proveniente direttamente dal DCO realizza l'impulso.
Il DCO presenta un banco di 16 capacità che possono essere controllate tramite un codice di 4 bit. In Tabella2.1 sono elencate le frequenze associate ad ogni codice; sono inoltre indicate le frequenze divise dal prescaler che costituiscono i valori in ingresso al circuito di calibrazione.
Il range di frequenze ottenuto tramite simulazione non è centrato alla corretta frequenza di trasmissione, $F = 8GHz$, ma attorno alla frequenza $F = 8.64GHz$. Questa differenza è dovuta al fatto che le simulazioni per ricavare il range di frequenze non sono state effettuate post-layuot. Il mismatch tra i due risultati non interferisce però nell'estrapolazione delle specifiche di cui è stato tenuto comunque conto ipotizzando una variabilità di $\pm 5\%$ delle frequenze. Questa variazione del range porta a realizzare un circuito in grado di operare nei due casi estremi: $7.79 - 8.64$ GHz e $8.61 - 9.54$ GHz.
Il range di frequenze è non lineare e la distanza tra due valori varia. La minima distanza tra due frequenze è di $50MHz$, quindi la massima risoluzione di misura della frequenza non dovrà superare i $25MHz$. Questo parametro non tiene conto del prescaler che, dividendo le frequenze, riduce anche la risoluzione massima a $3MHz$.

2.2 Consumo di energia

L'energia di un singolo impulso trasmesso è di $E_{pTx} = 13pJ$, mentre la totale energia dissipata dal trasmettitore corrisponde a $E_{dTx} = 186pJ/impulso$.
Prima della trasmissione dell'informazione viene inviato un codice di sincronizzazione che utilizza 19 ripetizioni di un codice a 31 bit seguito dallo stesso codice invertito per indicare il termine della sincronizzazione. L'energia spesa durante questa fase equivale a $E_{syn} = 115.3nJ$.
Il circuito di calibrazione da progettare dovrà quindi consumare un'energia minore di questo valore mantenendo la precisione necessaria per la misura della frequenza.

Codice	$F_{VCO}(GHz)$	$F_{PRESCALER}(GHz)$
0000	9.09	1.136
0001	9.02	1.128
0010	8.95	1.119
0011	8.89	1.111
0100	8.82	1.103
0101	8.76	1.095
0110	8.70	1.088
0111	8.64	1.08
1000	8.58	1.073
1001	8.52	1.065
1010	8.47	1.059
1011	8.41	1.051
1100	8.36	1.045
1101	8.30	1.037
1110	8.25	1.031
1111	8.20	1.025

Tabella 2.1: Tabella delle frequenze misurate.

Capitolo 3

Implementazione del circuito di calibrazione

Il circuito di calibrazione controlla, tramite la misurazione, la frequenza proveniente dal DCO e imposta il codice del banco di capacità in modo tale da ottenere la frequenza desiderata per la trasmissione. Per eseguire questo processo necessita di due componenti principali: un contatore di frequenze e un blocco che realizza la ricerca.
All'interno di questo capitolo sono esposte le implementazioni del contatore di frequenze (con l'analisi della risoluzione della misura), degli algoritmi di ricerca utilizzati e dell'unità di controllo. Infine è presentata la struttura del circuito completo tramite i blocchi precedenti.

3.1 Contatore di frequenze e prescaler

Per poter misurare una frequenza incognita occorre utilizzare un misuratore che confronti la frequenza da calcolare con una frequenza nota. Quest'ultima frequenza determina la finestra temporale per il conteggio degli eventi. Dal rapporto tra il valore calcolato e la finestra temporale si ottiene una stima della frequenza cercata.
Il contatore è stato realizzato secondo lo schema in Figura 3.1 ed è composto dai seguenti blocchi:

- contatore dei tempi;
- base dei tempi;
- contatore di eventi;
- circuito di gate.

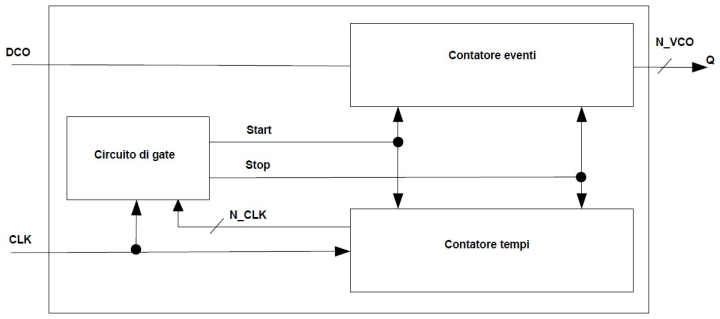

Figura 3.1: Schema di un contatore reciproco [9].

Il contatore dei tempi misura il numero di impulsi provenienti dalla base dei tempi, che in questo caso è il clock, mentre il contatore di eventi misura il numero di impulsi del segnale di ingresso. I due contatori sono realizzati con lo stesso modulo e hanno: due ingressi per il controllo, uno di abilitazione e uno di reset; un ingresso per il segnale da misurare e un'uscita per il valore contato.
Il circuito di gate ha il compito di abilitare il conteggio e di terminarlo quando è stato raggiunto il tempo di apertura della misurazione.
Al segnale di avvio i contatori vengono abilitati ma, mentre il contatore dei tempi inizia la misura nello stesso momento, il secondo contatore deve attendere il primo evento da misurare, in quanto non sincronizzato con il clock. Il segnale di termine conteggio avviene al raggiungimento della finestra temporale determinata come KT_R (dove T_R è il tempo riferimento o clock e K è un numero intero). Tra lo start e lo stop possono presentarsi due diversi casi, mostrati in Figura 3.2. L'assenza di sincronizzazione compor-

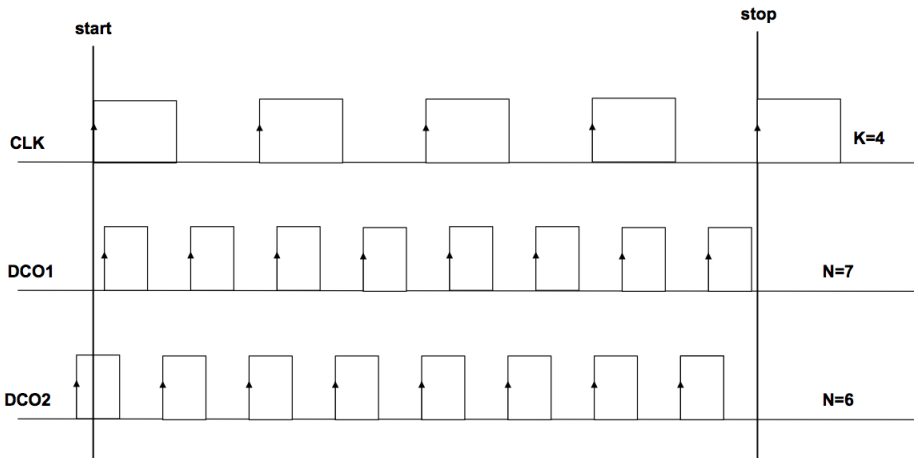

Figura 3.2: Effetto della non sincronizzazione dei segnali da misurare.

ta un'incertezza del conteggio pari a $\pm T_X$ quindi, considerando questa una variazione di misura, si può ricavare che la finestra temporale corrisponde a:

$$KT_R = (N_X \pm 1)T_X \tag{3.1}$$

Invertendo la precedente equazione si ottiene la formula per il calcolo della frequenza da misurare F_X:

$$F_X = N_X \frac{F_R}{K} \pm \frac{F_R}{K} \tag{3.2}$$

dove N_X è il numero di eventi contati nell'intervallo KT_R.
La misura ha una risoluzione pari a $\frac{F_R}{K} = \frac{1}{KT_R}$ e risulta inversamente proporzionale al tempo di apertura del gate.
Per determinare il valore di K occorre imporre sulla formula precedente la specifica sulla risoluzione della misura:

$$3MHz = \pm\frac{F_{CLK}}{K} \rightarrow K \simeq 27 \tag{3.3}$$

dove $F_{CLK} = 80MHz$.
Per semplificare l'elaborazione della misura, a causa della presenza di una divisione nella formula, il valore K è stato modificato portandolo alla potenza di due superiore più vicina: 32.
Cambiando il parametro K viene modificata anche la risoluzione della misura che diventa di $2,5MHz$. L'aumento di risoluzione provoca un incremento del tempo di misura ma diminuisce la potenza consumata grazie alla riduzione della complessità del circuito.
Il contatore degli eventi deve poter misurare il valore N_X calcolato come:

$$F_X = \frac{N_X F_{CLK}}{K} \rightarrow N_X \simeq 477 \tag{3.4}$$

Dove F_{CLK} e K sono i valori sopracitati e $F_X = 9.09GHz + 5\%$ che rappresenta la massima frequenza misurata dal circuito. Da K e N_X infine si ricavano il numero di bit necessari per il contatore dei tempi e per il contatore di eventi risultando rispettivamente 6 e 9 bit.
Il contatore degli eventi opera ad una frequenza che varia, nominalmente, da $8,20$ a $9,09$ GHz portando il componente ad un alto consumo di potenza e quindi di energia. Per ridurre questo contributo, rendendolo comparabile con il consumo del secondo contatore, è stato inserito un prescaler di valore $N = 8$ prima dell'ingresso del contatore degli eventi. Il valore N è stato scelto in modo da avere una frequenza da misurare comparabile con il clock. L'introduzione del prescaler però, porta alla modifica della specifica sulla risoluzione che diventa $0,4MHz$. In modo analogo a quello sopradescritto sono stati ricavati i valori $K = 200$, portato a 256, e $N_X = 477$ che necessitano entrambi di 9 bit per essere rappresentati.

Il prescaler è stato implementato digitalmente tramite un contatore asincrono con un reset con la funzione di azzerare i registri e mantenerlo disattivato quando non necessario.
L'uscita del contatore di frequenze fornisce il valore N_X misurato e non la frequenza stimata F_X, questo perchè il calcolo di quest'ultima comporterebbe l'introduzione di moltiplicatori e divisori. La frequenza obiettivo $F_{OBIETTIVO}$, da inserire in ingresso al circuito di calibrazione, dovrà quindi essere espressa secondo la seguente formula:

$$F = \frac{K}{F_{CLK}} \frac{F_{OBIETTIVO}}{8} \tag{3.5}$$

dove K è il numero di conteggi del clock, F_{CLK} è la frequenza di clock espressa in GHz e 8 è il valore della divisione introdotta dal primo prescaler. Ad esempio, se si vuole una $F_{OBIETTIVO} = 8.64GHz$, con $K = 32$ e $F_{CLK} = 0.08GHz$ si dovrà inserire un valore pari a $F = 432$.
Nel caso dell'introduzione del secondo prescaler si dovrà dividere ulteriormente per 8 senza ulteriori modifiche.

3.2 Algoritmi di ricerca

Gli algoritmi di ricerca scelti per questo circuito di calibrazione sono:

- ricerca sequenziale;
- ricerca binaria;
- ricerca per interpolazione.

L'algoritmo di ricerca sequenziale controlla in sequenza, crescente o decrescente, gli elementi di un insieme di dati e verifica se l'elemento controllato sia uguale all'elemento cercato. Se questa condizione è verificata allora la ricerca termina altrimenti continua fino al controllo dell'ultimo elemento.
Questo algoritmo è semplice da realizzare, necessita infatti solo di un'unità di controllo per l'iterazione e di un comparatore, ma il tempo di esecuzione è elevato, O(n), dove n rappresenta il numero di elementi dell'insieme.
La ricerca binaria è un algoritmo di ricerca che individua l'elemento cercato in un insieme ordinato di dati. L'algoritmo inizia la ricerca dall'elemento centrale dell'insieme e attraverso il confronto tra questo elemento e quello cercato viene determinata la successiva azione dell'algoritmo:

- se gli elementi sono uguali, la ricerca termina;
- se l'elemento cercato è inferiore, la ricerca ripete le operazioni precedenti sugli elementi della prima metà dell'insieme;

- se l'elemento cercato è superiore, la ricerca ripete le operazioni precedenti sugli elementi della seconda metà dell'insieme.

Nel caso in cui tutti gli elementi siano stati visionati, la ricerca termina senza trovare l'elemento cercato.
Per il circuito di calibrazione da realizzare è possibile utilizzare questo tipo di algoritmo perché le frequenze, ossia gli elementi della ricerca, sono in ordine decrescente al crescere dei codici del banco di capacità e crescente al decrescere dei codici.
Rispetto al primo algoritmo di ricerca questo è più complesso da realizzare ma riduce i tempi dell'esecuzione a O(logn).
La ricerca per interpolazione si compone di due fasi: nella prima viene realizzata una funzione tramite interpolazione polinomiale del secondo ordine degli elementi dell'insieme; mentre nella seconda viene eseguita l'effettiva ricerca, sequenziale, dell'elemento all'interno della funzione.
Questo algoritmo di ricerca può essere utilizzato per la calibrazione, perché le frequenze hanno un andamento crescente (o decrescente) del secondo ordine al diminuire (o aumentare) dei codici del banco di capacità.
L'ultimo algoritmo è anche quello di maggior complessità di realizzazione in quanto, prima della ricerca, deve effettuare numerosi passaggi matematici per ottenere la funzione. Il tempo di esecuzione dipende dalla ricerca utilizzata che, in questo caso, è sequenziale.

3.2.1 Ricerca sequenziale

L'algoritmo utilizzato controlla i codici del banco di capacità in ordine crescente, quindi in sequenza decrescente di frequenza e verifica se l'errore ottenuto sia minore dell'errore precedente. Se questo non avviene l'algoritmo termina la ricerca, altrimenti continua fino all'ultimo codice.
La ricerca sequenziale è stata implementata come una macchina a stati finiti (MSF) composta da tre stati:

- INIT: stato di inizializzazione;
- ERR1: stato di calcolo del primo errore;
- MIS: stato di calcolo degli errori successivi.

Lo stato INIT inizializza il codice di partenza per la ricerca a 0000. Il passaggio allo stato successivo è automatico ed avviene alla successiva abilitazione della macchina.
Lo stato ERR1 calcola l'errore della frequenza relativa al codice 0000, lo salva nel registro dell'errore, con il codice relativo, per le successive comparazioni. Prima di passare allo stato successivo, viene inviato il nuovo codice 0001.

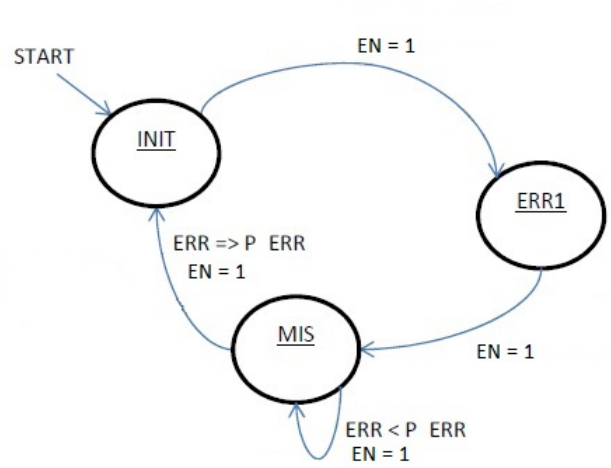

Figura 3.3: Diagramma degli stati della ricerca sequenziale.

Nello stato MIS avviene il confronto tra l'errore precedente e quello attuale, se quest'ultimo è maggiore la ricerca termina e viene inviato il codice precedente; altrimenti l'errore attuale sostituisce quello precedente e viene inviato il codice successivo.
L'aggiornamento degli stati e dei registri della macchina avviene tramite il segnale di abilitazione inviato dal blocco di controllo. Questo segnale funge anche da clock per la macchina. Il reset porta a zero tutti i registri e a INIT lo stato.

3.2.2 Ricerca binaria

L'algoritmo cerca il risultato iniziando la misurazione dal valore mediale del range, iterando il processo nelle successive metà, basandosi su una data condizione.
La direzione del percorso è decisa dal segno dell'errore, calcolato come $ERR = F_{MISURATA} - F_{OBIETTIVO}$. Se l'errore è minore di zero, andrà cercata una frequenza maggiore di quella attuale, e quindi un codice minore, altrimenti si cercherà una frequenza minore.
I codici da verificare sono sedici, da 0000 a 1111, e il valore da cui inizia la ricerca può essere 0111 o 1000. In base alla scelta si sbilancia l'albero binario aggiungendo un passo che, a seconda del codice iniziale, può essere rispettivamente 1111 o 0000.
La scelta del codice iniziale si è basata sulla semplicità della verifica della condizione per l'ultimo passo della ricerca. La misura del codice 0000 avviene solamente se in precedenza il codice era 0001 e l'errore negativo, mentre

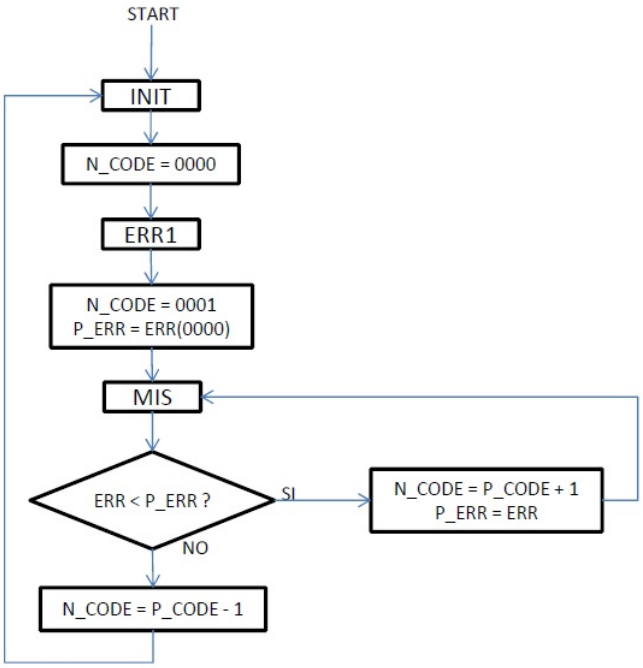

Figura 3.4: Diagramma di flusso della ricerca sequenziale.

la misura di 1111 avviene per 1110 ed errore positivo. Mentre la verifica del codice precedente è uguale in entrambi i casi, la verifica del segno dell'errore è più semplice nel primo caso perchè basta controllare il bit più significativo. L'algoritmo di ricerca è stato implementato con una MSF composta da sei

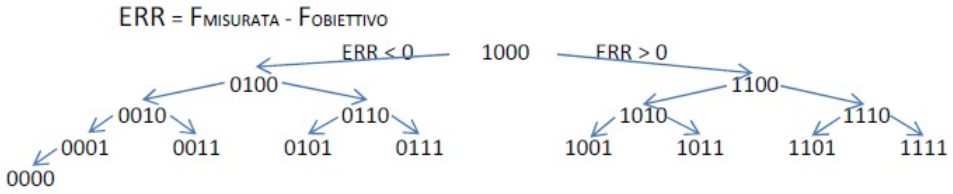

Figura 3.5: Albero della ricerca binaria.

stati:

- INIT: inizializzazione;
- P1: primo passo della ricerca;
- P2: secondo passo della ricerca;
- P3: terzo passo della ricerca;

- P4: quarto passo della ricerca;
- P5: quinto passo della ricerca.

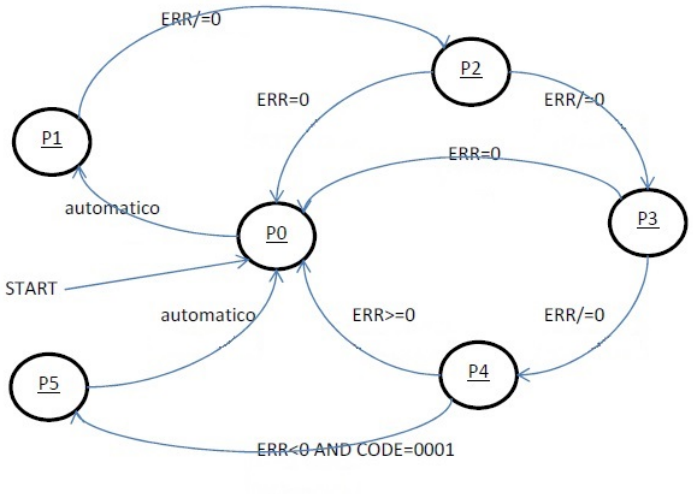

Figura 3.6: Diagramma degli stati della ricerca binaria.

Lo stato INIT invia il primo codice della ricerca (1000) e passa al successivo stato, P1. In questo stato viene verificato se l'errore è uguale a zero; se ciò accade la ricerca termina, altrimenti viene stabilito il codice successivo, in base alla struttura mostrata in Figura 3.5. L'errore nello stato P1 viene salvato, insieme al codice corrispondente, per essere successivamente confrontato. Il primo errore è necessariamente anche quello minore.
Lo stato P2 calcola il codice successivo, come nello stato precedente, ma, a differenza di P1, l'errore attuale non viene automaticamente salvato. Esso infatti viene confrontato con il precedente e solo se risulta minore viene salvato.
Lo stato P3 è del tutto simile allo stato P2.
Nello stato P4 possono verificarsi due casi:

- $ERR < 0$ e $CODE = 0001$: si comporta come gli stati precedenti e passa in P5;
- $ERR \geq 0$: la ricerca termina e viene scelto il codice che fornisce la frequenza più vicina all'obiettivo.

Lo stato P5 è l'ultimo passo della ricerca, qui viene confrontato l'errore attuale con il minore e si determina il codice corrispondente all'errore più piccolo tra i due.
L'aggiornamento degli stati e dei registri della macchina avviene tramite il segnale di abilitazione inviato dal blocco di controllo. Questo segnale funge

anche da clock per la macchina. Il reset porta a zero tutti i registri e a INIT lo stato.

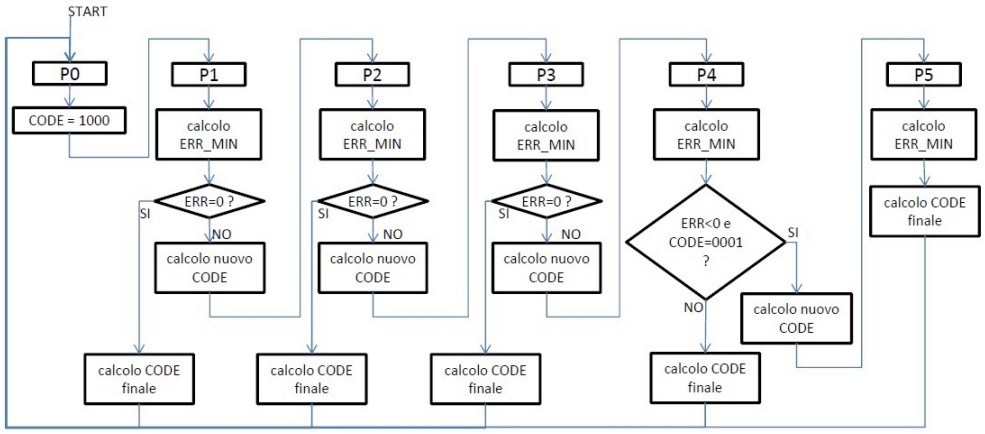

Figura 3.7: Diagramma di flusso della ricerca binaria.

3.2.3 Ricerca per interpolazione

E' un algoritmo (Figura 3.8) che sfrutta l'interpolazione per poter ricavare una funzione da cui estrarre, data la frequenza, il corretto codice. Una volta ottenuta la funzione occorre realizzare anche un algoritmo di ricerca.
La funzione da calcolare è un polinomio di secondo grado ottenuto dalla misura di tre frequenze (F_1, F_2 e F_3) relative a tre codici predefiniti (x_1, x_2 e x_3). Tramite queste misure si può approssimare il range di frequenze del DCO:

$$F_{DCO}(x) = ax^2 + bx + c \tag{3.6}$$

dove x rappresenta il codice e a, b e c sono i coefficienti ricavati dalla seguente relazione:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \tag{3.7}$$

Da cui si ricava:

$$\begin{cases} a = \frac{-F_1}{(x_2-x_1)(x_1-x_3)} + \frac{-F_2}{(x_3-x_2)(x_2-x_1)} + \frac{-F_3}{(x_3-x_2)(x_1-x_3)} \\ b = \frac{(x_3+x_2)F_1}{(x_2-x_1)(x_1-x_3)} + \frac{(x_1+x_3)F_2}{(x_3-x_2)(x_2-x_1)} + \frac{(x_2+x_1)F_3}{(x_3-x_2)(x_1-x_3)} \\ c = \frac{-x_3x_2F_1}{(x_2-x_1)(x_1-x_3)} + \frac{-x_3x_1F_2}{(x_3-x_2)(x_2-x_1)} + \frac{-x_2x_1F_3}{(x_3-x_2)(x_1-x_3)} \end{cases} \tag{3.8}$$

L'errore viene poi calcolato come $ERR(x) = F_{DCO}(x) - F_{OBIETTIVO}$ e, tramite iterazione, si ottiene il codice che fornisce l'errore minore.

A causa della complessità dei calcoli per i coefficienti, sono presenti infat-

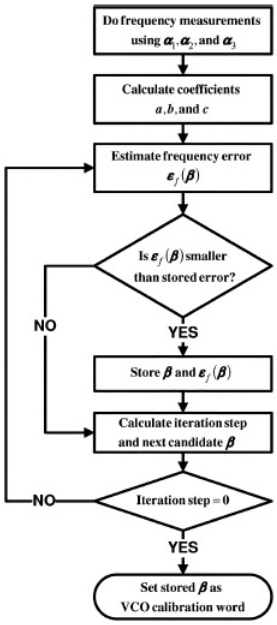

Figura 3.8: Diagramma di flusso base della ricerca per interpolazione [8].

ti sia moltiplicazioni che divisioni, è necessario scegliere attentamente i tre codici per le misure.

Il primo codice che è stato scelto è 0000 perché semplifica molto il calcolo del coefficiente c, rendendolo difatti uguale alla frequenza F_1. Gli altri codici sono stati scelti in modo da evitare il più possibile l'inserimento di moltiplicatori e divisori. Il metodo per evitare questi elementi, molto costosi in termini energetici, è quello di moltiplicare e dividere per potenze di due. In questo caso si sostituiscono le operazioni di moltiplicazione e divisione rispettivamente con degli shift a sinistra e a destra. Utilizzando questo metodo le scelte dei due codici restanti sono tra: 0001, 0010, 0100 e 1000. Con due qualsiasi di questi codici si ottiene una buona approssimazione del range di frequenze solamente nella prima parte dell'interpolazione. La seconda parte invece comincia a divergere introducendo un errore troppo elevato.

La scelta finale è ricaduta sui codici 1000 e 1100, in quanto forniscono una migliore approssimazione della curva (Figura 3.9) riducendo i calcoli dei coefficienti a somme, sottrazioni, shift e a una sola divisione per 3.

Le formule dei coefficienti diventano:

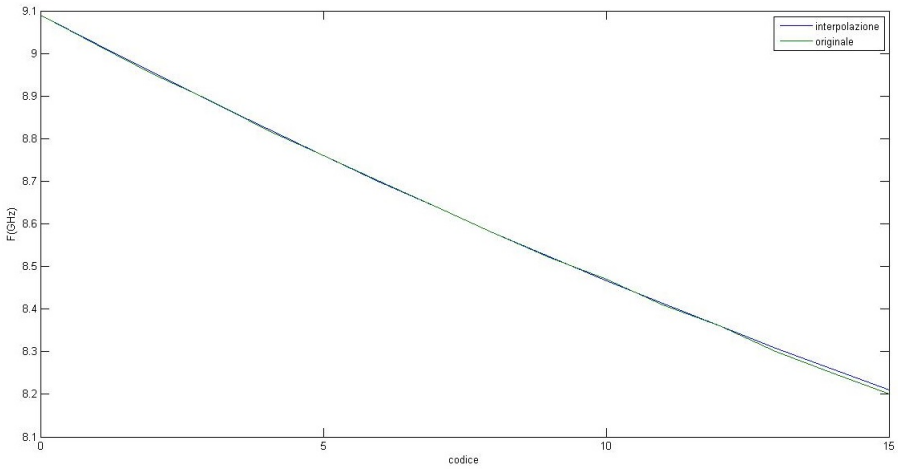

Figura 3.9: Grafico dell'interpolazione ricavata dalle frequenze di codice 0000, 1000 e 1100.

$$\begin{cases} a = \frac{1}{32}\left(\frac{F_1+2F_3}{3} - F_2\right) \\ b = -\frac{1}{8}\left(\frac{5F_1}{3} - 3F_2 + \frac{4F_3}{3}\right) \\ c = F_1 \end{cases} \tag{3.9}$$

I coefficienti sono stati moltiplicati per 3, in modo da eliminare la divisione e aggiungere delle moltiplicazioni che possano essere semplificate come uno shift a sinistra di 1 bit (equivalente alla moltiplicazione per due) e una somma del valore stesso, ovvero: $3x = 2x + x$.
Al coefficiente c è stato sottratto il valore della frequenza obiettivo, moltiplicata per 3 per mantenere lo stesso rapporto con i coefficienti; in questo modo la funzione finale rappresenterà l'errore:

$$ERR(x) = ax^2 + bx + c - F_{OBIETTIVO} = ax^2 + bx + c' \tag{3.10}$$

dove $c' = c - F_{OBIETTIVO}$.
Il codice corretto, ovvero quello che fornisce la frequenza più vicina all'obiettivo, viene ricavato tramite iterazione. Quando si verifica la condizione $ERR(x) < 0$ l'iterazione termina. A questo punto il valore assoluto dell'errore precedente viene confrontato con il valore assoluto dell'errore dell'ultimo calcolo; il codice risultante sarà quello corrispondente all'errore minore tra i due confrontati.
Ogni errore è composto dalla somma dell'errore precedente con un termine costante ($T = a + b$) e un termine che aumenta ad ogni passo:

$$ERR(0) = c$$

$$ERR(1) = a + b + c = ERR(0) + T$$

$$ERR(2) = 4a + 2b + c = ERR(1) + T + 2a$$

$$\dots$$

$$ERR(15) = 225a + 15b + c = ERR(14) + T + 28a$$

$$ERR(x+1) = ERR(x) + T + (2x-1)a \qquad ERR(0) = c \tag{3.11}$$

E' stato scelto di scomporre il calcolo dell'errore in questo modo per ridurre il consumo energetico, infatti vengono eliminate le operazioni più complesse quali moltiplicazione ed elevamento a potenza.
L'algoritmo è stato implementato tramite una MSF con quattro stati:

- SLEEP: stato di attesa e di inizializzazione della ricerca;
- MIS: stato di misura delle frequenze e calcolo dei coefficienti;
- CNT: stato di conteggio;
- INT: stato in cui avviene il calcolo iterativo del codice.

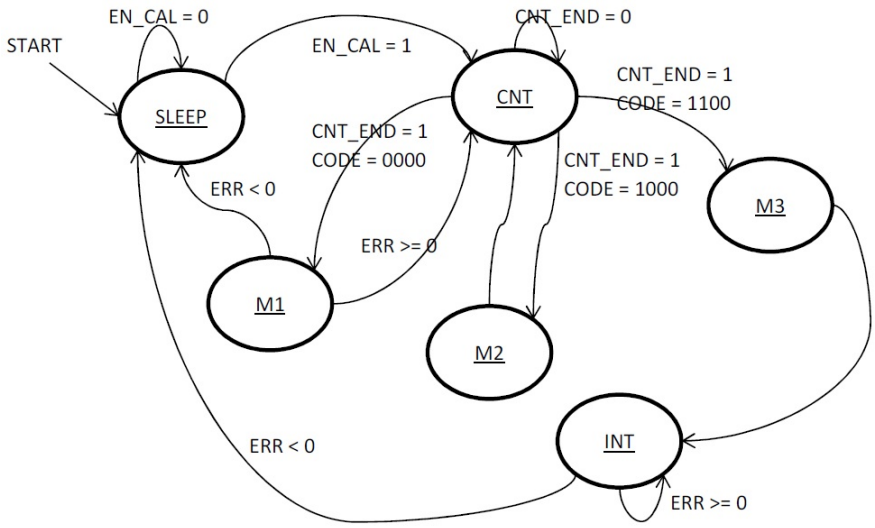

Figura 3.10: Diagramma degli stati della ricerca per interpolazione.

Il primo stato, SLEEP, inizializza tutti i registri a zero e attende l'abilitazione della calibrazione per avviare la ricerca. Lo stato successivo è CNT, qui avviene il conteggio della prima frequenza che, una volta terminato, passa allo stato M1. Questo stato, dopo aver letto la prima frequenza, calcola l'errore e se questo è negativo termina la ricerca (è la verifica di ERR(0)); altrimenti inizia il calcolo dei coefficienti a, b e c e invia il codice per la frequenza successiva. Lo stato M2, dopo aver ricevuto il valore della seconda frequenza dallo stato CNT, continua l'elaborazione dei coefficienti e invia

il codice per la terza frequenza. Il calcolo completo dei coefficienti avviene nello stato M3. Da M3 si passa poi allo stato INT, in cui avviene il calcolo iterativo del codice finale. Ad ogni ciclo di clock viene verificato il segno dell'errore appena calcolato. Se il segno dell'errore è negativo la ricerca si ferma ed esso viene confrontato con l'errore del ciclo precedente; tale confronto produrrà un codice secondo la condizione esposta in precendenza. La ricerca termina comunque al raggiungimento dell'ultimo codice: 1111. Terminata la ricerca si passa allo stato SLEEP.

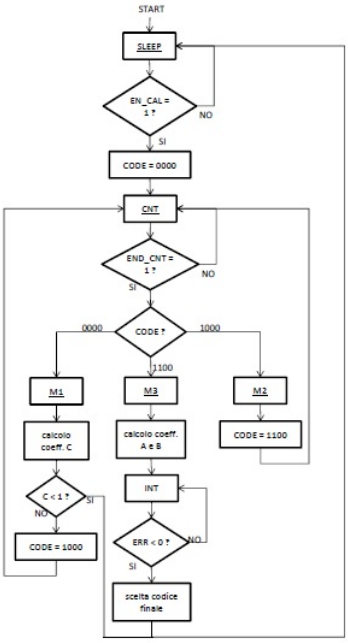

Figura 3.11: Diagramma di flusso della ricerca per interpolazione.

3.2.4 Rappresentazione binaria dell'errore nelle ricerche

Negli algoritmi di ricerca, che andranno ad essere implementati di seguito, occorre calcolare l'errore tra la frequenza misurata e la frequenza obiettivo. Per determinare il numero di bit per rappresentare correttamente questo valore bisogna cotrollare l'errore massimo che può essere calcolato in ogni ricerca.
Nella ricerca sequenziale la frequenza misurata iniziale è quella più elevata e, considerata una variazione di $\pm 5\%$, corrisponde a $9.54GHz$, mentre la frequenza obiettivo varia da $8.2GHz$ a $9.09GHz$. Da questi valori risulta che l'errore massimo da misurare sarà: $-1.34GHz$ ($8.20GHz - 9.54GHz$). Il numero di bit necessari per la rappresentazione di questo valore è di 9 bit in complemento a due.

Nella ricerca binaria la frequenza iniziale è quella con codice 1000, che nominalmente risulta essere $8.58GHz$. La prima frequenza misurata sarà quindi $8.64 \pm 5\% GHz$ e l'errore massimo risulterà: $-0.9GHz$ ($8.20GHz - 9.1GHz$). Per rappresentare questo valore occorrono 8 bit in complemento a due.
Nella ricerca per interpolazione l'errore necessita di un numero maggiore di bit poiché il coefficiente a del polinomio del secondo ordine è minore di uno. Senza aumento di bit il contributo di a sarebbe nullo, trasformando di fatto il polinomio di secondo grado in una retta (Figura 3.12).
Per portare il coefficiente a ad un valore maggiore di uno è stato modificato

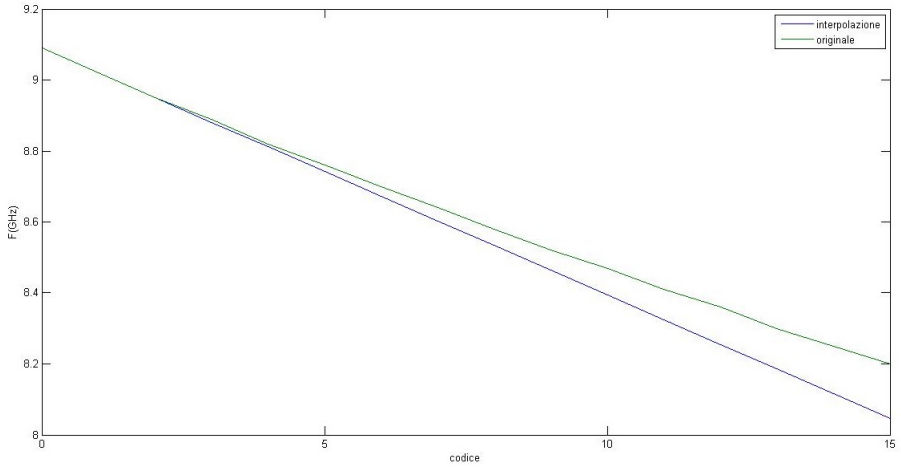

Figura 3.12: Interpolazione con $a = 0$.

il sistema di formule presentate nella sezione relativa alla ricerca per interpolazione, moltiplicando per 32 tutti i coefficienti. In questo modo però la rappresentazione dell'errore diventa di 15 bit in complemento a due.

3.3 Unità di controllo

Il blocco controllo realizza la MSF del circuito di calibrazione ed è composta da tre stati:

- SLEEP: stato di attesa e inizializzazione;
- CNT: stato di conteggio della frequenza;
- SEARCH: stato di ricerca.

SLEEP è lo stato di attesa del circuito di calibrazione e si attiva solo dopo aver ricevuto il segnale di abilitazione. Prima di passare allo stato successivo, CNT, viene abilitata la MSF della ricerca per effettuare il reset. Lo stato

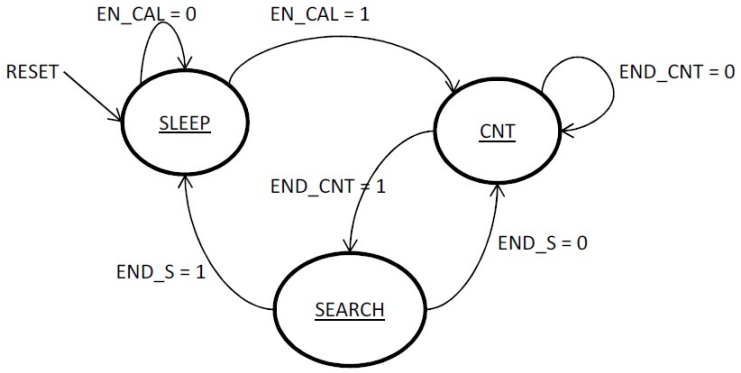

Figura 3.13: Diagramma degli stati del blocco di controllo.

CNT abilita il blocco di conteggio della frequenza, che lo mantiene abilitato fino al completamento del conteggio. Lo stato successivo è SEARCH, in cui viene attivata la MSF della ricerca. Ad ogni passaggio in questo stato la MSF della ricerca viene aggiornata fino al completamento della stessa. Terminata la ricerca la macchina si riporta nello stato di SLEEP, mantenendo il risultato.
L'aggiornamento degli stati avviene ogni ciclo di clock e il reset porta lo stato a SLEEP.
All'interno dello stato SEARCH sono stati implementati gli algoritmi di ricerca sequenziale e binaria, mentre la ricerca per interpolazione contiene già al suo interno il blocco di controllo. Questo perché, mentre le prime due ricerche si svolgono nello stesso modo, la terza ha un comportamento diverso che impedisce il riutilizzo del blocco.

3.4 Circuito completo

Il circuito completo è stato realizzato con due strutture diverse a seconda dell'algoritmo implementato. La prima struttura, mostrata in Figura 3.15, riguarda le ricerche sequenziale e binaria ed è composta dai seguenti blocchi:

- F_CNT: contatore di frequenze;
- DIV: prescaler;
- BIN o SEQ: algoritmo di ricerca, rispettivamente, binaria o sequenziale;
- CONTROL(UC): unità di controllo del circuito di calibrazione;
- REG_CODE: registro del codice del banco di capacità.

La seconda struttura, mostrata in Figura 3.16, realizza il circuito di calibrazione con la ricerca per interpolazione ed è composto dai blocchi:

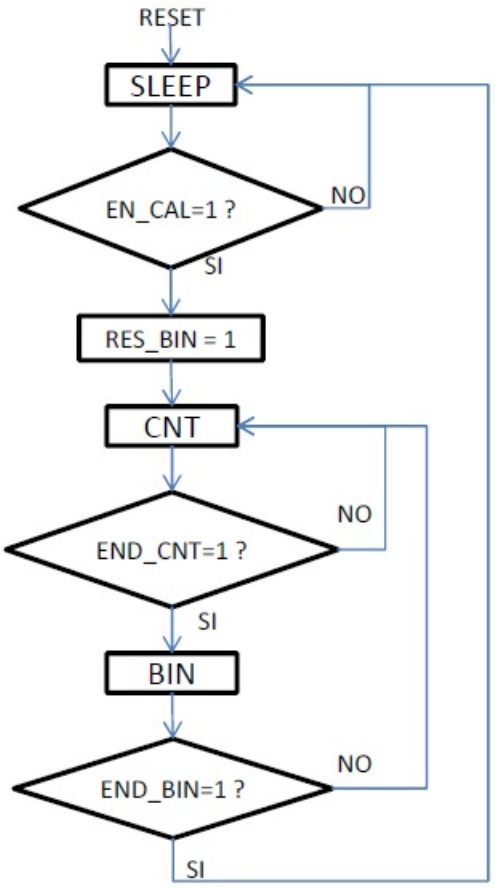

Figura 3.14: Diagramma di flusso del blocco di controllo.

- F_CNT: contatore di frequenze;
- DIV: prescaler;
- INT: algoritmo di ricerca per interpolazione.

In questo secondo circuito l'unità di controllo non è esterna, come nella struttura precedente, ma è interna al blocco di ricerca, come il registro del codice del banco di capacità.
Per entrambe le strutture i nomi dei collegamenti sono gli stessi che sono stati utilizzati per l'implementazione del circuito, il cui codice è riportato in appendice.

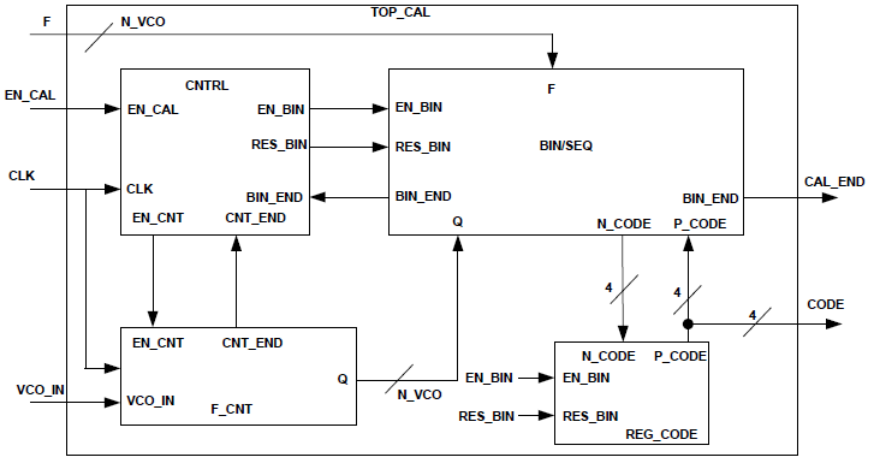

Figura 3.15: Rappresentazione a blocchi del circuito con le ricerche sequenziale e binaria.

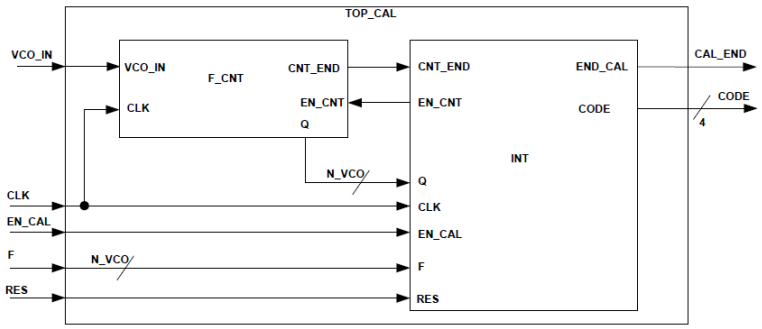

Figura 3.16: Rappresentazione a blocchi del circuito con la ricerca per interpolazione.

Capitolo 4

Verifica prestazioni

L'analisi del circuito si divide in tre parti:

- verifica della correttezza della ricerca;
- verifica della risoluzione;
- analisi del consumo di energia.

4.1 Correttezza della ricerca e risoluzione

Per ogni algoritmo di ricerca sono state eseguite delle simulazioni con lo scopo di verificare l'esatta correlazione tra le frequenze nominali e il codice corrispondente. L'esito di questa verifica è risultato corretto per tutte le ricerche.
Dopo aver verificato la correttezza della ricerca per le frequenze nominali si procede alla verifica della risoluzione. Data una frequenza casuale, diversa da quelle nominali, la ricerca deve fornire il codice corrispondente alla frequenza nominale con valore più vicino a quella data.
Nelle Figure 4.1, 4.2 e 4.3 sono rappresentate le simulazioni relative alle tre ricerche per la frequenza $F = 8.46GHz$ (rappresentata dal valore 423). La frequenza nominale più vicina al valore scelto è $F = 8.47GHz$ a cui corrisponde il codice 1010. Le simulazioni effettuate su queste ricerche hanno condotto allo stesso risultato corretto.

Nel caso particolare in cui la frequenza casuale si trovi a metà tra due frequenze nominali gli algoritmi di ricerca si comportano in modo diverso, come si può osservare nelle Figure4.4, 4.5 e 4.6. In questa simulazione è stata utilizzata una frequenza obiettivo pari a $8.44GHz$ (rappresentata dal valore 422); questa frequenza si posiziona a metà tra $8.41GHz$ e $8.47GHz$, i cui rispettivi codici sono 1011 e 1010.

Le ricerche sequenziale e binaria forniscono lo stesso codice, 1011, mentre la ricerca per interpolazione da come risultato il codice 1010. Questa

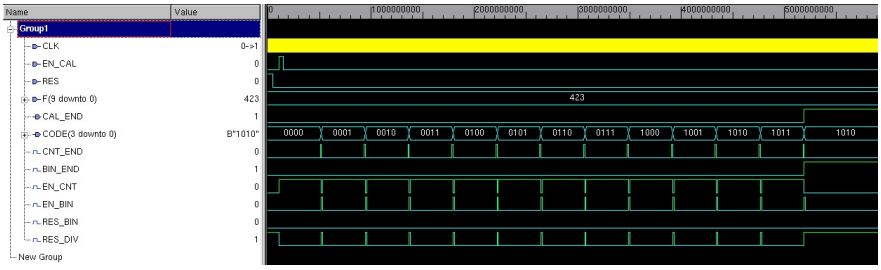

Figura 4.1: Verifica della correttezza della ricerca sequenziale.

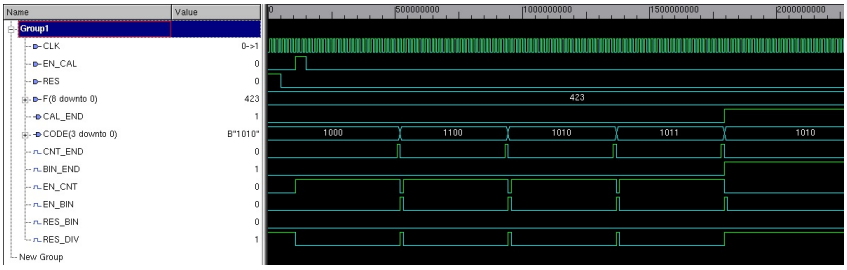

Figura 4.2: Verifica della correttezza della ricerca binaria.

variazione di comportamento dipende dalle condizioni di scelta del codice implementate nel precedente capitolo.

4.2 Consumo di energia

L'analisi delle tre tipologie di algoritmi è stata effettuata tramite il confronto del consumo di energia, calcolato come: $E = Potenza * Tempo$.
Il tempo è stato ricavato dalla simulazione del caso peggiore dei vari algoritmi. Per le ricerche sequenziale e per interpolazione il caso peggiore corrisponde al raggiungimento del codice 1111, mentre per la ricerca binaria del codice 0000. Questo periodo temporale inizia dall'abilitazione della calibrazione e termina con l'attivazione del segnale di fine ricerca.

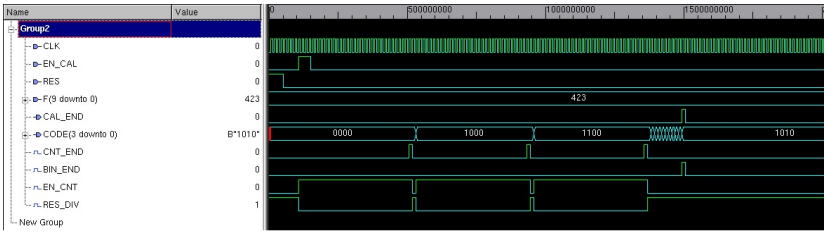

Figura 4.3: Verifica della correttezza della ricerca per interpolazione.

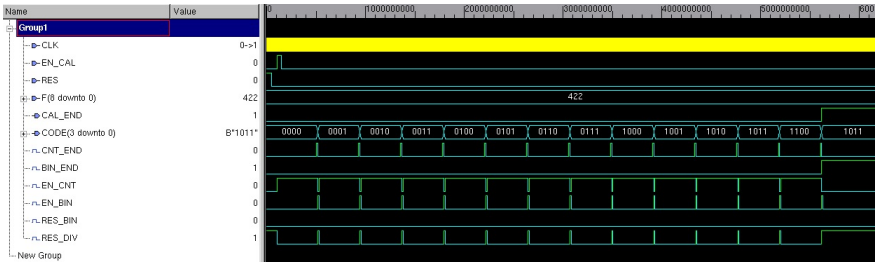

Figura 4.4: Verifica della risoluzione della ricerca sequenziale.

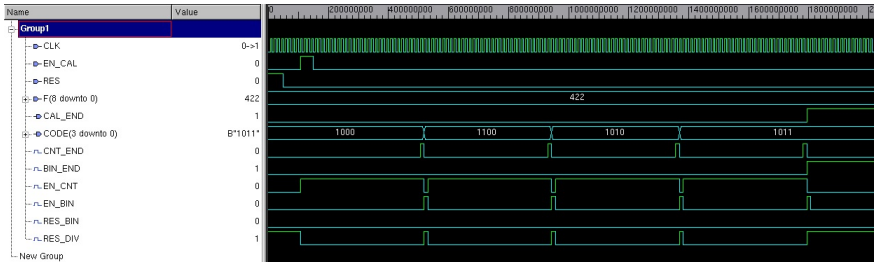

Figura 4.5: Verifica della risoluzione della ricerca binaria.

Il tempo di calibrazione dipende principalmente dal numero di misure necessarie per l'algoritmo di ricerca e in minor modo da un tempo fisso, sempre dipendente dalla ricerca implementata:

$$T_{tot} = ((K+1)N_{RICERCA} + C_{RICERCA})\frac{1}{F_{CLK}} \tag{4.1}$$

dove $K+1$ è il numero di conteggi del segnale di clock compreso l'abilitazione del contatore, $N_{RICERCA}$ identifica il numero di misure necessarie per la ricerca, $C_{RICERCA}$ è la costante dipendente dalla ricerca e F_{CLK} la frequenza di clock.
Il circuito opera principalmente con due segnali a frequenza distinta: il segnale di clock, a $F_{CLK} = 80MHz$, e il segnale proveniente dal DCO a frequenza F_{DCO}. La stima di potenza va quindi separata in due contributi, a seconda di quale frequenza venga usata nei blocchi che compongono il cir-

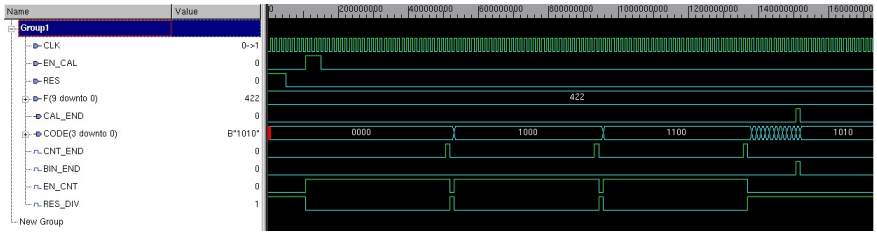

Figura 4.6: Verifica della risoluzione della ricerca per interpolazione.

cuito.
Per il calcolo della potenza totale si è proceduto sommando i vari componenti valutati alla propria frequenza di lavoro:

$$\begin{aligned} P_{TOT} = P_{RICERCA}(F_{CLK}) + P_{FCNT}(F_{DCO}) + P_{FCNT}(F_{CLK}) \\ + P_{PRESCALER}(F_{DCO}) + P_{NSP}(F_{CLK}) \end{aligned} \tag{4.2}$$

Il contatore di frequenze opera a entrambe le frequenze e ognuno delle due componenti di potenza considera il consumo del contatore a cui è associato. Gli altri blocchi vengono valutati con la sola frequenza a cui lavorano. Nella seguente Tabella4.1 sono presentati i dati di tempo, potenza ed energia dei tre algoritmi stimati nel caso peggiore e senza il prescaler in ingresso. La

	Sequenziale	Binaria	Interpolazione
Tempo (ns)	7212.5	2112.5	1450
Potenza (μW)	256.95	267.29	434.46
Energia (nJ)	1.85	0.56	0.63

Tabella 4.1: Stima di tempo, potenza ed energia delle ricerche implementate senza prescaler.

ricerca sequenziale presenta il minor consumo di potenza (grazie alla semplicità di realizzazione) ma ha un tempo di calibrazione molto elevato, in quanto ad ogni confronto di frequenza corrisponde una misura di frequenza, portando il consumo di energia al valore massimo tra le soluzioni proposte. L'algoritmo di ricerca per interpolazione ha il vantaggio di misurare solamente tre volte la frequenza proveniente dal DCO e, nonostante l'elevata complessità del circuito, fornisce un buon valore di energia consumata.
La ricerca binaria presenta un ridotto tempo di calibrazione (nel caso peggiore deve eseguire solamente cinque misure) e un consumo di potenza intermedio tra i due algoritmi precedenti. Come risultato di ciò quest'ultimo algoritmo realizza un circuito di calibrazione con il minor consumo di energia tra le soluzioni proposte.
Per provare a ridurre i consumi di potenza è stato introdotto un prescaler che divide la frequenza per 8. La successiva Tabella4.2 riporta i dati di tempo, potenza ed energia, ricavati con lo stesso principio esposto sopra. L'aggiunta del prescaler riduce significativamente la potenza consumata dalle tre soluzioni, avvenuta grazie alla riduzione dell'effettiva frequenza da misurare. A parità di risoluzione però, i tempi di calibrazione aumentano notevolmente a causa dell'aumento dei conteggi da effettuare, come spiegato nel paragrafo 3.1. La riduzione di potenza non riesce a compensare l'aumento del tempo portando così a un peggioramento dei consumi energetici. Il prescaler è stato quindi eliminato dalla realizzazione del circuito di calibrazione.

	Sequenziale	Binaria	Interpolazione
Tempo (ns)	54812.5	16112.5	9850
Potenza (μW)	201.42	209.67	338.73
Energia (nJ)	11.04	3.38	3.34

Tabella 4.2: Stima di tempo, potenza ed energia delle ricerche implementate con prescaler.

Capitolo 5

Conclusione

Lo scopo di questa tesi è quello di realizzare un circuito di calibrazione della frequenza per un trasmettitore a impulsi. Il circuito presenta due blocchi principali: il primo è un contatore di frequenze, mentre il secondo è un circuito che implementa una ricerca.
Il contatore di frequenze misura il segnale proveniente dal DCO le cui frequenze possono variare tra 8.2 a 9.09 GHz con un passo minimo di 50 MHz. Da queste specifiche di risoluzione sono stati dimensionati i contatori presenti nel contatore di frequenze come esposto nella sezione 3.1.
Gli algoritmi di ricerca implementati, la cui realizzazione è esposta nella sezione 3.2, sono tre:

- ricerca sequenziale;
- ricerca binaria;
- ricerca per interpolazione.

Successivamente si è proceduto alla verifica delle prestazioni del circuito, che possono essere separate in due parti: una sul funzionamento della ricerca e una sui consumi energetici, obiettivo della tesi.
Dai dati ricavati, esposti nel capitolo 4, si può ritenere che la soluzione che rispetta meglio le specifiche imposte è l'algoritmo di ricerca binaria. Infatti, oltre ad effettuare una corretta ricerca, il consumo energetico risulta il minore tra le soluzioni implementate.
La ricerca per interpolazione, a causa della complessità di circuito, risulta penalizzata per il consumo di potenza maggiore. Questo algoritmo però non è da sccartare in quanto ha migliori prestazioni in circuiti di calibrazione con un numero di codici (e quindi range di frequenza) elevati perchè il tempo di calibrazione aumenta in maniera poco rilevante rispetto alle altre due soluzioni proposte. Inoltre la ricerca utilizzata dall'algoritmo per interpolazione può essere modificata da sequenziale (utilizzata in questa tesi) a binaria. In questo caso i consumi di potenza aumentano ulteriormente ma si riduce il

tempo necessario per la calibrazione.
La ricerca sequenziale, infine, può risultare una soluzione accettabile solo nel caso di sistemi di calibrazione a pochissimi bit in quanto, con l'aumentare delle frequenze disponibili, il tempo necessario per la calibrazione in maniera lineare.

Appendice A

Codice VHDL

A.1 Algoritmi di ricerca

A.1.1 Ricerca sequenziale

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4
5  --algoritmo di ricerca sequenziale
6  entity R_SEQ is
7  generic(N_VCO: integer := 9;
8      N_ERR: integer := 8);
9  port(EN_R: in std_logic; --segnale di abilitazione
10   RES_R: in std_logic; --segnale di reset
11   Q: in unsigned(N_VCO-1 downto 0); --frequenza misurata
12   F: in unsigned(N_VCO-1 downto 0); --frequenza obiettivo
13   N_CODE: out unsigned(3 downto 0); --codice successivo
14   P_CODE: in unsigned(3 downto 0); --codice attuale
15   R_END: out std_logic); --segnale di termine ricerca
16 end R_SEQ;
17
18 architecture BHV of R_SEQ is
19 type STATE is (INIT, ERR1, MIS);
20 signal P_STATE: STATE := INIT;
21 signal N_STATE: STATE;
22 signal P_T_ERR, N_T_ERR: unsigned(N_ERR-1 downto 0);
23
24 begin
25   process(P_STATE, P_CODE)
26   variable ERR: signed(N_ERR-1 downto 0);
27   variable ABS_ERR: unsigned(N_ERR-1 downto 0);
28   begin
29     --ERR = F_misurata - F_obiettivo
30     ERR := conv_signed(Q - F, ERR'length);
31     --l'errore va confrontato in valore assoluto
32     ABS_ERR := conv_unsigned(abs(ERR), ERR'length);
33     case P_STATE is
34       --inizializzazione
35       --
36       --invia il primo codice per il calcolo dell'errore.
37       when INIT =>
38         N_CODE <= "0000"; --codice iniziale
```

```
39          N_T_ERR <= P_T_ERR;
40          N_STATE <= ERR1;
41          R_END <= '0';
42        --ERR1
43        --
44        --misura dell'errore del primo codice inviato, "0000",
45        --e invio del secondo codice.
46        when ERR1 =>
47          N_CODE <= "0001";
48          N_T_ERR <= ABS_ERR;
49          N_STATE <= MIS;
50          R_END <= '0';
51        --MIS
52        --
53        --viene verificato se l'errore attuale, ABS_ERR, e'
54        --minore dell'errore precedente, P_T_ERR, e calcolato
55        --il successivo codice da inviare.
56        --se l'errore attuale e' maggiore, l'algoritmo si ferma
57        --e fornisce il codice precedente = P_CODE - 1.
58        when MIS =>
59          if ABS_ERR < P_T_ERR then
60            N_T_ERR <= ABS_ERR;
61            N_CODE <= P_CODE + 1;
62            N_STATE <= MIS;
63            R_END <= '0';
64          --caso in cui termina la ricerca senza risultato
65          elsif P_CODE = "1111" then
66            N_T_ERR <= P_T_ERR;
67            N_CODE <= P_CODE;
68            N_STATE <= INIT;
69            R_END <= '1';
70          else
71            N_T_ERR <= P_T_ERR;
72            N_CODE <= P_CODE - 1;
73            N_STATE <= INIT;
74            R_END <= '1';
75          end if;
76      end case;
77    end process;
78
79    --aggiornamento degli stati
80    process(EN_R)
81    begin
82      if EN_R'event and EN_R = '1' then
83        if RES_R = '0' then
84          P_STATE <= N_STATE;
85          P_T_ERR <= N_T_ERR;
86        else
87          P_STATE <= INIT;
88          P_T_ERR <= (others => '0');
89        end if;
90      end if;
91    end process;
92  end BHV;
```

A.1.2 Ricerca binaria

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4
5  --algoritmo di ricerca binaria
```

```
6  entity R_BIN is
7  generic(N_VCO: integer := 9;
8      N_ERR: integer := 8);
9  port(EN_BIN, RES_BIN: in std_logic; --segnale di abilitazione
10   Q: in unsigned(N_VCO-1 downto 0); --frequenza misurata
11   F: in unsigned(N_VCO-1 downto 0); --frequenza obiettivo
12   N_CODE: out unsigned(3 downto 0); --codice del banco di capacita'
13   P_CODE: in unsigned(3 downto 0); --codice precedente
14   BIN_END: out std_logic); --segnale di termine ricerca
15 end R_BIN;
16
17 architecture BHV of R_BIN is
18 type STATE is (INIT, P1, P2, P3, P4, P5);
19 signal P_STATE: STATE := INIT;
20 signal N_STATE: STATE;
21
22 signal N_MIN_ERR, P_MIN_ERR: unsigned(N_ERR-1 downto 0);
23 signal N_MIN_CODE, P_MIN_CODE: unsigned(3 downto 0);
24
25 begin
26   process(P_STATE)
27   variable ERR: signed(N_ERR-1 downto 0);
28   variable ABS_ERR: unsigned(N_ERR-1 downto 0);
29   begin
30     --ERR = F_misurata - F_obiettivo
31     ERR := conv_signed(Q - F, ERR'length);
32     --l'errore va confrontato in valore assoluto
33     ABS_ERR := conv_unsigned(abs(ERR), ERR'length);
34     case P_STATE is
35       --inizializzazione
36       when INIT =>
37         N_CODE <= "1000"; --codice iniziale da cui parte l'algoritmo di ricerca
38         N_MIN_ERR <= P_MIN_ERR;
39         N_MIN_CODE <= P_MIN_CODE;
40         N_STATE <= P1;
41         BIN_END <= '0';
42       --primo passo dell'algoritmo
43       when P1 =>
44         --il primo errore ottenuto e' anche l'errore minore
45         N_MIN_ERR <= ABS_ERR;
46         N_MIN_CODE <= P_CODE;
47         --se l'errore e' nullo il codice minore e' quello inviato
48         if ERR = 0 then
49           N_CODE <= P_CODE;
50           N_STATE <= INIT;
51           BIN_END <= '1';
52         else
53           --se l'errore e' negativo si sceglie una frequenza maggiore
54           if ERR(N_ERR-1) = '1' then
55             N_CODE <= "0100";
56           else --altrimenti si sceglie una frequenza minore
57             N_CODE <= "1100";
58           end if;
59           N_STATE <= P2;
60           BIN_END <= '0';
61         end if;
62       --secondo passo
63       when P2 =>
64         --il codice e l'errore vengono salvati
65         --solo se l'errore attuale e' minore dell'errore minore
66         if ABS_ERR < P_MIN_ERR then
67           N_MIN_ERR <= ABS_ERR;
```

```
68            N_MIN_CODE <= P_CODE;
69          else
70            N_MIN_ERR <= P_MIN_ERR;
71            N_MIN_CODE <= P_MIN_CODE;
72          end if;
73          if ERR = 0 then
74            N_CODE <= P_CODE;
75            N_STATE <= INIT;
76            BIN_END <= '1';
77          else
78            --scelta del codice successivo
79            --caso in cui il codice precedente
80            --era 1100
81            if P_CODE(3) = '1' then
82              if ERR(N_ERR-1) = '1' then
83                N_CODE <= "1010";
84              else
85                N_CODE <= "1110";
86              end if;
87            --caso in cui il codice precedente
88            --era 0100
89            else
90              if ERR(N_ERR-1) = '1' then
91                N_CODE <= "0010";
92              else
93                N_CODE <= "0110";
94              end if;
95            end if;
96            N_STATE <= P3;
97            BIN_END <= '0';
98          end if;
99        --terzo passo
100       --
101       --il terzo stato è simile al secondo
102       --tranne nel calcolo del codice successivo
103       when P3 =>
104         if ABS_ERR < P_MIN_ERR then
105           N_MIN_ERR <= ABS_ERR;
106           N_MIN_CODE <= P_CODE;
107         else
108           N_MIN_ERR <= P_MIN_ERR;
109           N_MIN_CODE <= P_MIN_CODE;
110         end if;
111         if ERR = 0 then
112           N_CODE <= P_CODE;
113           N_STATE <= INIT;
114           BIN_END <= '1';
115         else
116           if ERR(N_ERR-1) = '1' then
117             N_CODE <= P_CODE - 1;
118           else
119             N_CODE <= P_CODE + 1;
120           end if;
121           N_STATE <= P4;
122           BIN_END <= '0';
123         end if;
124       --quarto passo
125       when P4 =>
126         if ABS_ERR < P_MIN_ERR then
127           N_MIN_ERR <= ABS_ERR;
128           N_MIN_CODE <= P_CODE;
129         else
```

```
130           N_MIN_ERR <= P_MIN_ERR;
131           N_MIN_CODE <= P_MIN_CODE;
132         end if;
133         --se si presenta questa condizione avviene il quinto passo
134         if ERR(N_ERR-1) = '1' and P_CODE = 1 then
135           N_CODE <= P_CODE - 1;
136           N_STATE <= P5;
137           BIN_END <= '0';
138         else --altrimenti viene scelto il codice finale
139           if ABS_ERR < P_MIN_ERR then
140             N_CODE <= P_CODE;
141           else
142             N_CODE <= P_MIN_CODE;
143           end if;
144           N_STATE <= INIT;
145           BIN_END <= '1';
146         end if;
147       --quinto passo
148       when P5 =>
149         --scelta del codice finale
150         if ABS_ERR < P_MIN_ERR then
151           N_MIN_ERR <= ABS_ERR;
152           N_MIN_CODE <= P_CODE;
153           N_CODE <= P_CODE;
154         else
155           N_MIN_ERR <= P_MIN_ERR;
156           N_MIN_CODE <= P_MIN_CODE;
157           N_CODE <= P_MIN_CODE;
158         end if;
159         BIN_END <= '1';
160         N_STATE <= INIT;
161     end case;
162   end process;
163
164   --aggiornamento degli stati
165   process(EN_BIN)
166   begin
167     if EN_BIN'event and EN_BIN = '1' then
168       if RES_BIN = '0' then
169         P_STATE <= N_STATE;
170         P_MIN_ERR <= N_MIN_ERR;
171         P_MIN_CODE <= N_MIN_CODE;
172       else
173         P_STATE <= INIT;
174         P_MIN_ERR <= (others => '0');
175         P_MIN_CODE <= "1000";
176       end if;
177     end if;
178   end process;
179 end BHV;
```

A.1.3 Ricerca per interpolazione

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.STD_LOGIC_ARITH.all;
4
5 --algoritmo di ricerca per interpolazione
6 entity R_INT is
7 generic(N_VCO: integer := 10;
8     N_ERR: integer := 15);
9 port(CLK, RES: in std_logic;
```

```
10    EN_CAL: in std_logic; --segnale di avvio calibrazione
11    CNT_END: in std_logic; --segnale di termine conteggio
12    END_CAL: out std_logic; --termine calibrazione
13    EN_CNT: out std_logic; --segnale di abilitazione del contatore
14    RES_DIV: out std_logic; --reset del prescaler
15    Q: in unsigned(N_VCO-1 downto 0); --frequenza misurata
16    F: in unsigned(N_VCO-1 downto 0); --frequenza obiettivo
17    CODE: out unsigned(3 downto 0)); --codice
18  end R_INT;
19
20  architecture BHV of R_INT is
21  type STATE is (M1, M2, M3, CNT, INT, SLEEP);
22  signal P_STATE: STATE := SLEEP;
23  signal N_STATE: STATE;
24
25
26  signal N_CODE, P_CODE: unsigned(3 downto 0);
27  --coefficienti a, b e c
28  signal N_A, P_A, N_B, P_B, N_C, P_C: signed(N_ERR-1 downto 0);
29  signal N_V, P_V, N_PRE_ERR, P_PRE_ERR: signed(N_ERR-1 downto 0);
30
31  begin
32    process(P_STATE, CNT_END, EN_CAL, P_C)
33    variable T, T_A, T_B, T_C: signed(N_ERR-1 downto 0);
34    begin
35
36      case P_STATE is
37        --stato di attesa
38        when SLEEP =>
39          if EN_CAL = '1' then
40            N_STATE <= CNT;
41            N_CODE <= "0000";
42          else
43            N_STATE <= SLEEP;
44            N_CODE <= P_CODE;
45          end if;
46          N_A <= (others => '0');
47          N_B <= (others => '0');
48          N_C <= (others => '0');
49          N_V <= (others => '0');
50          N_PRE_ERR <= (others => '0');
51          EN_CNT <= '0';
52          RES_DIV <= '1';
53          END_CAL <= '0';
54        --misura della prima frequenza
55        --e calcolo del coefficiente c
56        when M1 =>
57          N_A <= conv_signed(Q, N_A'length);
58          N_B <= conv_signed(shl(Q, "10") + Q, N_B'length);
59          N_V <= P_V;
60          N_PRE_ERR <= P_PRE_ERR;
61          --al coefficiente c è stato sottratto la frequenza
62          --obiettivo per rappresentare l'errore relativo
63          --al codice 0000
64          T_C := conv_signed(Q - F, T_C'length);
65          N_C <= shl(T_C + shl(T_C, "1"), "101");
66          --se c < 0 la ricerca termina in quanto
67          --il codice 0000 rappresenta la massima
68          --frequenza
69          if T_C(N_VCO-1) = '1' then
70            N_CODE <= "0000";
71            N_STATE <= SLEEP;
```

```
72            END_CAL <= '1';
73          else
74          N_CODE <= "1000";
75          N_STATE <= CNT;
76          END_CAL <= '0';
77          end if;
78          EN_CNT <= '0';
79          RES_DIV <= '1';
80        --misura della seconda frequenza
81        --e calcolo della prima parte
82        --dei coefficienti a e b
83        when M2 =>
84          N_A <= P_A - shl(Q, "1") - Q;
85          N_B <= P_B - shl(Q, "11") - Q;
86          N_C <= P_C;
87          N_V <= P_V;
88          N_PRE_ERR <= P_PRE_ERR;
89          N_CODE <= "1100";
90          N_STATE <= CNT;
91          EN_CNT <= '0';
92          RES_DIV <= '1';
93          END_CAL <= '0';
94        --misura della terza frequenza
95        --e calcolo finale dei coefficienti a e b
96        when M3 =>
97          T_A := P_A + shl(Q, "1");
98          N_A <= T_A;
99          T_B := -shl(P_B + shl(Q, "10"), "10");
100         N_B <= T_B;
101         N_V <= P_V;
102         N_PRE_ERR <= P_PRE_ERR;
103         N_C <= P_C + T_A + T_B;
104         N_STATE <= INT;
105         EN_CNT <= '0';
106         RES_DIV <= '1';
107         END_CAL <= '0';
108         N_CODE <= "0001";
109       --conteggio della frequenza
110       when CNT =>
111         EN_CNT <= '1';
112         RES_DIV <= '0';
113         if CNT_END = '1' then
114           --CODE = 1100
115           if P_CODE(2) = '1' then
116             N_STATE <= M3;
117           --CODE = 1000
118           elsif P_CODE(3) = '1' then
119             N_STATE <= M2;
120           --CODE = 0000
121           else
122             N_STATE <= M1;
123           end if;
124         else
125           N_STATE <= CNT;
126         end if;
127         N_A <= P_A;
128         N_B <= P_B;
129         N_C <= P_C;
130         N_V <= P_V;
131         N_PRE_ERR <= P_PRE_ERR;
132         N_CODE <= P_CODE;
133         END_CAL <= '0';
```

```
134       -- interpolazione
135       when INT =>
136         T := P_A + P_B;
137         --se ERR < 0 o i codici sono finiti la ricerca termina
138         if (P_C(N_ERR-1) = '1') or (P_CODE = 15) then
139           --scelta del codice finale
140           if P_PRE_ERR > abs(P_C) then
141             N_CODE <= P_CODE;
142           else
143             N_CODE <= P_CODE - 1;
144           end if;
145           END_CAL <= '1';
146           N_V <= P_V;
147           N_PRE_ERR <= P_PRE_ERR;
148           N_C <= P_C;
149           N_STATE <= SLEEP;
150         else
151           --iterazione per il calcolo del nuovo errore
152           END_CAL <= '0';
153           N_CODE <= P_CODE + 1;
154           N_V <= P_V + shl(P_A, "1");
155           N_PRE_ERR <= P_C;
156           N_C <= P_C + P_V + T;
157           N_STATE <= INT;
158         end if;
159         N_A <= P_A;
160         N_B <= P_B;
161         EN_CNT <= '0';
162         RES_DIV <= '1';
163     end case;
164   end process;
165
166   --aggiornamento degli stati
167   process(CLK)
168   begin
169     if CLK'event and CLK = '1' then
170       if RES = '0' then
171         P_STATE <= N_STATE;
172         P_A <= N_A;
173         P_B <= N_B;
174         P_C <= N_C;
175         P_V <= N_V;
176         P_PRE_ERR <= N_PRE_ERR;
177         P_CODE <= N_CODE;
178         CODE <= N_CODE;
179       else
180         P_STATE <= SLEEP;
181         P_A <= (others => '0');
182         P_B <= (others => '0');
183         P_C <= (others => '0');
184         P_V <= (others => '0');
185         P_PRE_ERR <= (others => '0');
186         P_CODE <= "0000";
187         CODE <= "0000";
188       end if;
189     end if;
190   end process;
191 end BHV;
```

A.2 Unità di controllo

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4
5  --unità di controllo per le ricerche binaria e sequenziale
6  entity CONTROL is
7  port(CLK, RES: in std_logic;
8    EN_CAL: in std_logic; --segnale di avvio calibrazione
9    CNT_END: in std_logic; --segnale di termine conteggio
10   BIN_END: in std_logic; --segnale di termine ricerca
11   EN_CNT: out std_logic; --abilitazione del contatore
12   EN_BIN: out std_logic; --abilitazione della ricerca
13   RES_BIN: out std_logic; --segnale di reset della ricerca
14   RES_DIV: out std_logic); --segnale di reset del prescaler
15 end CONTROL;
16
17 architecture BHV of CONTROL is
18 type STATE is (CNT, SEARCH, SLEEP);
19 signal P_STATE: STATE := SLEEP;
20 signal N_STATE: STATE;
21 begin
22   process(P_STATE, CNT_END, BIN_END, EN_CAL)
23   begin
24     case P_STATE is
25       --attesa
26       --
27       --mantiene disabilitati gli altri blocchi
28       --fino all'abilitazione della calibrazione
29       when SLEEP =>
30         if EN_CAL = '1' then
31           N_STATE <= CNT;
32           RES_BIN <= '1';
33           EN_BIN <= '1';
34         else
35           N_STATE <= SLEEP;
36           RES_BIN <= '0';
37           EN_BIN <= '0';
38         end if;
39         EN_CNT <= '0';
40         RES_DIV <= '1';
41       --conteggio
42       --
43       --il contatore viene mantenuto abilitato
44       --fino al termine del conteggio
45       when CNT =>
46         EN_CNT <= '1';
47         EN_BIN <= '0';
48         RES_BIN <= '0';
49         RES_DIV <= '0';
50         if CNT_END = '1' then
51           N_STATE <= SEARCH;
52         else
53           N_STATE <= CNT;
54         end if;
55       --ricerca
56       --
57       --attivazione dell'algoritmo di
58       --ricerca che può essere lo stato
59       --finale o passare ad un altro
60       --conteggio
61       when SEARCH =>
62         EN_CNT <= '0';
```

```
63          EN_BIN <= '1';
64          RES_BIN <= '0';
65          RES_DIV <= '1';
66          if BIN_END = '1' then
67            N_STATE <= SLEEP;
68          else
69            N_STATE <= CNT;
70          end if;
71      end case;
72    end process;
73
74    --aggiornamento degli stati
75    process(CLK)
76    begin
77      if CLK'event and CLK = '1' then
78        if RES = '0' then
79          P_STATE <= N_STATE;
80        else
81          P_STATE <= SLEEP;
82        end if;
83      end if;
84    end process;
85  end BHV;
```

A.2.1 Registro del codice

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4
5  --registro del codice del banco di capacita'
6  entity REG_CODE is
7  port(EN_BIN: in std_logic; --abilitazione dei registri
8    RES_BIN: in std_logic; --reset dei registri
9    N_CODE: in unsigned(3 downto 0); --scrittura del codice
10   P_CODE: out unsigned(3 downto 0)); --lettura del codice
11 end REG_CODE;
12
13 architecture BHV of REG_CODE is
14 begin
15   process(EN_BIN)
16   begin
17     if EN_BIN'event and EN_BIN = '1' then
18       if RES_BIN = '0' then
19         P_CODE <= N_CODE;
20       else
21         P_CODE <= "1000";--"0000" per la ricerca sequenziale
22       end if;
23     end if;
24   end process;
25 end BHV;
```

A.3 Contatore di frequenze

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4
5  --contatore di frequenze
6  entity FREQ_CNT is
7    generic(N_CLK: integer := 6; --numero di bit del contatore del clock
```

```
8         N_VCO: integer := 9; --numero di bit del contatore del VCO
9         K: integer := 32); --numero di conteggi del clock
10    port(CLK: in std_logic; --clock
11      VCO_IN: in std_logic; --segnale da misurare la frequenza
12      EN_CNT: in std_logic; --abilitazione del contatore
13      CNT_END: out std_logic; --segnale di termine conteggio
14      Q: out unsigned(N_VCO-1 downto 0)); --numero di conteggi del VCO
15 end FREQ_CNT;
16
17 architecture RTL of FREQ_CNT is
18 signal EN_C, R: std_logic;
19 signal Q_CLK: unsigned(N_CLK-1 downto 0);
20 signal Q_VCO: unsigned(N_VCO-1 downto 0);
21 begin
22   --contatore del clock
23   CNT_CLK: entity WORK.CNT(BHV) generic map(N => N_CLK)
24   port map(CK => CLK, EN => EN_C, R => R, Q => Q_CLK);
25   --contatore del VCO
26   CNT_VCO: entity WORK.CNT(BHV) generic map(N => N_VCO)
27   port map(CK => VCO_IN, EN => EN_C, R => R, Q => Q_VCO);
28
29   --controllo dei contatori
30   process(EN_CNT, Q_CLK, Q_VCO)
31   begin
32     if EN_CNT = '1' then
33      --verifica della finestra temporale K/F_CLK
34       if Q_CLK(N_CLK-1) = '0' then
35         EN_C <= '1';
36         CNT_END <= '0';
37       else
38         EN_C <= '0';
39         CNT_END <= '1';
40       end if;
41       R <= '0';
42     --il reset dei contatori avviene solo quando
43     --il contatore viene disabilitato
44     else
45       R <= '1';
46       EN_C <= '0';
47       CNT_END <= '0';
48     end if;
49     Q <= Q_VCO;
50   end process;
51 end RTL;
```

A.3.1 Contatore

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.STD_LOGIC_ARITH.all;
4
5 --contatore a N bit
6 entity CNT is
7   generic(N: integer := 10); --numero di bit del contatore
8   port(CK: in std_logic; --segnale da contare
9     R: in std_logic; --reset
10    EN: in std_logic; --abilitazione
11    Q: inout unsigned(N-1 downto 0)); --valore contato
12 end CNT;
13
14 architecture BHV of CNT is
15 begin
```

```
16   process(CK, R, EN)
17   begin
18     if R = '0' then
19       if EN = '1' then
20         if CK'event and CK = '1' then
21           Q <= Q + 1;
22         end if;
23       end if;
24     else
25       Q <= (others => '0');
26     end if;
27   end process;
28 end BHV;
```

A.3.2 Prescaler

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
4
5  --prescaler
6  --divisione per 8 del segnale in ingresso
7  entity DIV is
8  port(F_VCO: in std_logic; --segnale da dividere
9    RES: in std_logic; --reset
10   F_DIV: out std_logic); --segnale diviso
11 end DIV;
12
13 architecture BHV of DIV is
14 signal N: unsigned(2 downto 0);
15 begin
16   process(F_VCO, RES)
17   begin
18     if RES = '0' then
19       if F_VCO'event and F_VCO = '1' then
20         --nella prima metà del conteggio
21         --l'uscita è pari a 1
22         if N = 3 then
23           N <= N + 1;
24           F_DIV <= '1';
25         --nella seconda metà l'uscita
26         --è pari a 0
27         elsif N = 7 then
28           N <= N + 1;
29           F_DIV <= '0';
30         else
31           N <= N + 1;
32         end if;
33       end if;
34     else
35       N <= "000";
36       F_DIV <= '0';
37     end if;
38   end process;
39 end BHV;
```

A.4 Circuito completo

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.all;
```

```
4
5  --top del circuito di calibrazione
6  entity TOP_CAL is
7  generic(N_CLK: integer := 6; --numero di bit del contatore del clock
8      N_VCO: integer := 9; --numero di bit del contatore del segnale dal VCO
9      N_ERR: integer := 8; --numero di bit dell'errore
10     K: integer := 32); --numero di conteggi del clock
11 port(CLK, EN_CAL, RES, VCO_IN: in std_logic;
12   F: in unsigned(N_VCO-1 downto 0); --frequenza obiettivo
13   CAL_END: out std_logic; --segnale di termine calibrazione
14   CODE: out unsigned(3 downto 0)); --codice del banco di capacita'
15 end TOP_CAL;
16
17 --la frequenza obiettivo va codificata come:
18 --F = (K*F_OBIETTIVO)/(8*F_CLK)
19 --F_CLK va espressa in GHz
20 --Esempio: F_obiettivo = 8.64 GHz => F = 432
21
22 architecture RTL of TOP_CAL is
23 signal CNT_END, BIN_END, EN_CNT, EN_BIN, RES_BIN, F_DIV, RES_DIV: std_logic;
24 signal Q: unsigned(N_VCO-1 downto 0);
25 signal T_N_CODE, T_P_CODE: unsigned(3 downto 0);
26
27 begin
28
29 --la configurazione attuale è impostata per utilizzare la ricerca binaria,
30 --se si desidera la ricerca sequenziale basta commentare il blocco della
31 --ricerca binaria e decommentare quello della ricerca sequenziale.
32 --per la ricerca per interpolazione non sono necessari i blocchi dell'unità
33 --di controllo e dei registri del codice quindi vanno commentati.
34 --per utilizzare il prescaler è necessario sostituire il segnale VCO_IN
35 --del blocco contatore di frequenze con il segnale F_DIV.
36
37   --unità di controllo
38   CNTRL: entity WORK.CONTROL(BHV)
39   port map(CLK => CLK, EN_CAL => EN_CAL, RES => RES,
40            CNT_END => CNT_END, BIN_END => BIN_END,
41            RES_BIN => RES_BIN, EN_CNT => EN_CNT,
42            EN_BIN => EN_BIN, RES_DIV => RES_DIV);
43
44   --ricerca binaria
45   BIN: entity WORK.R_BIN(BHV) generic map(N_VCO => N_VCO, N_ERR => N_ERR)
46   port map(EN_BIN => EN_BIN, RES_BIN => RES_BIN, Q => Q, F => F,
47            BIN_END => BIN_END, N_CODE => T_N_CODE, P_CODE => T_P_CODE);
48
49   --registri del codice del banco di capacità
50   REG_CODE: entity WORK.REG_CODE(BHV)
51   port map(EN_BIN => EN_BIN, RES_BIN => RES_BIN, N_CODE => T_N_CODE,
52            P_CODE => T_P_CODE);
53
54   --ricerca sequenziale
55 --  SEQ: entity WORK.R_SEQ(BHV) generic map(N_VCO => N_VCO, N_ERR => N_ERR)
56 --  port map(EN_R => EN_BIN, RES_R => RES_BIN, Q => Q, F => F,
57 --           R_END => BIN_END, N_CODE => T_N_CODE, P_CODE => T_P_CODE);
58
59   --ricerca per interpolazione
60 --  INT: entity WORK.R_INT_ORIGINALE(BHV) generic map(N_VCO => N_VCO)
61 --  port map(CLK => CLK, RES => RES, EN_CAL => EN_CAL, CNT_END => CNT_END,
62 --           EN_CNT => EN_CNT, RES_DIV => RES_DIV, Q => Q, F => F,
63 --           END_CAL  => BIN_END, CODE => T_N_CODE);
64
65   --contatore della frequenza
```

```
66    F_CNT: entity WORK.FREQ_CNT(RTL)
67    generic map(N_CLK => N_CLK, N_VCO => N_VCO, K => K)
68    port map(CLK => CLK, VCO_IN => VCO_IN, EN_CNT => EN_CNT,
69             CNT_END => CNT_END, Q => Q);
70
71    --prescaler
72 --  DIV: entity WORK.DIV(BHV) port map(F_VCO => VCO_IN,
73 --                                     F_DIV => F_DIV, RES => RES_DIV);
74
75    CODE <= T_N_CODE;
76    CAL_END <= BIN_END;
77
78 end RTL;
```

Bibliografia

[1] S. Soldà, M. Caruso, A. Bevilacqua, A. Gerosa, D. Vogrig and A. Neviani, *A 5 Mb/s UWB-IR Transceiver Front-End for Wireless Sensor Networks in 0.13 μ m CMOS*, IEEE J. Solid-State Circuits, vol. 46, no. 7, pp. 1636-1647, Jul. 2011.

[2] F. Zhang, A. Jha, R. Gharpurey and P. Kinget, *An Agile, Ultra-Wideband Pulse Radio Transceiver With Discrete-Time Wideband-IF*, IEEE J. Solid-State Circuits, vol. 44, no. 5, pp. 1336-1351, May 2009.

[3] Y. Zhu, J. D. Zuegel, J. R. Marciante and H. Wui, *Distribuited Waveform Generator: A New Circuit Technique for Ultra-Wideband Pulse Generation, Shaping and Modulation*, IEEE J. Solid-State Circuits, vol. 44, no. 3, pp. 808-823, Mar. 2009.

[4] M. U. Nair, Y. Zheng, C. W. Ang, Y. Lian, X. Yuan and C-H. Heng, *A Low SIR Impulse-UWB Transceiver Utilizing Chirp FSK in 0.18 μm CMOS*,IEEE J. Solid-State Circuits, vol. 45, no. 11, pp. 2388-2403, Nov. 2010.

[5] Y. J. Zheng, S-X. Diao, C-W. Ang, Y. Gao, F-C. Choong, Z. Chen, X. Liu, Y-S. Wang, X-J. Yuan, C. H. Heng, *A 0.92/5.3nJ/b UWB Impulse Radio SoC for Communication and Localization*, in IEEE ISSCC Dig. Tech. Papers, 2010, pp. 230-231.

[6] H. Lee, J. Cho, K. Lee, I. Hwang, T. Ahn, K. Nah and B. Park, *A $\Sigma - \Delta$ Fractional-N Frequency Synthesizer Using a Wide-Band Integrated VCO and a Fast AFC Technique for GSM/GPRS/WCDMA Applications*, IEEE J. Solid-State Circuits, vol. 39, no. 7, pp. 1164-1169, Jul. 2004.

[7] J. Shin and H. Shin, *A $1.9 - 3.8$ GHz $\Delta\Sigma$ Fractional-N PLL Frequency Synthesizer With Fast Auto-Calibration of Loop Bandwidth and VCO Frequency*, IEEE J. Solid-State Circuits, vol. 47, no. 3, pp. 665-675, Mar. 2012.

[8] P. Väänänen, N. Mikkola and P. Heliö, *VCO Design With On-Chip Calibration System*, IEEE Trans. Circuits Syst. I, vol. 53, no. 10, pp. 2157-2166, Oct. 2006.

[9] C. Narduzzi, *Dispense di Misure Elettroniche*, Università degli Studi di Padova, pp. 39-42, 2010.