Generatori di forme d'onda periodiche in VHDL e sintesi dei clock in FPGA*

Corso di Architetture dei Sistemi di Elaborazione

Prof. Antonino Mazzeo
Dip. di Informatica e Sistemistica Univ. di Napoli Federico II

9 dicembre 2018

*Il presente capitolo è in bozza in quanto ancora incompleto e in fase di revisione

1 Caratteristiche di un clock

Un clock è una forma d'onda rettangolare caratterizzata da un periodo T, o equivalentemente dalla frequenza f, uguale a 1/T, e da una fase φ (fig.1.1).

Si definisce *duty cycle* D (ciclo di lavoro utile) il rapporto tra il tempo T_H in cui il clock è nello stato alto e il periodo T. Il duty cycle indica la frazione di T in cui un segnale di clock è nello stato alto, esso è spesso utilizzato anche per indicare la frazione di ciclo in cui un apparato compie un *lavoro utile* (supponendo che tale lavoro sia compiuto durante lo stato alto di un segnale periodico).

Se si indica con T_L la durata del clock nello stato basso valgono le relazioni:

$$D = \frac{T_H}{T} = \frac{T_H}{T_H + T_L}$$

$$T_H = D \cdot T$$

$$T_L = \frac{1-D}{T}$$

Si fa osservare che il duty cycle D è un numero sempre compreso tra 0 e 1. I casi D=0 e D=1 sono casi limite e sono relativi a segnali continui i cui livelli assumono, rispettivamente, il valore zero e il valore uno per tutto il periodo T.

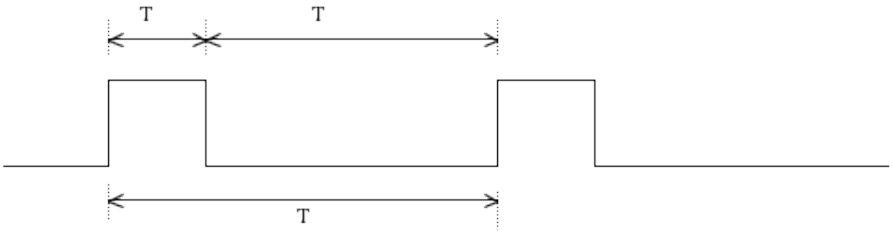


Figura 1.1: Parametri caratteristici di un segnale di clock

Il duty cycle è espresso anche come valore percentuale $D\%$ del rapporto Tp/T. La percentuale esprime la frazione del periodo in cui il segnale è alto. Ad esempio, un duty cycle *D=0.4* indica che per il 40% del periodo T, il segnale è nello stato alto, mentre per il 60% è basso. Il caso D%=50% caratterizza, ovviamente, il pattern di un'onda quadra simmetrica in cui per metà del periodo il segnale è alto e per l'altra basso.

La *fase* di una forma periodica indica lo spostamento relativo della forma rispetto all'asse dei tempi, ovvero la frazione di periodo trascorsa rispetto a un riferimento (ad esempio l'istante di inizio del periodo). La fase è solitamente espressa mediante il valore di un angolo misurato in gradi, o radianti, compreso tra 0° e 360°, tenendo conto che un intero periodo T corrisponde a un angolo di 360° o 2π radianti. Ad esempio, in fig.1.2 è mostrata una forma d'onda sfasata di 45° o $\pi/4$ o T/4 (in quadratura di fase) .

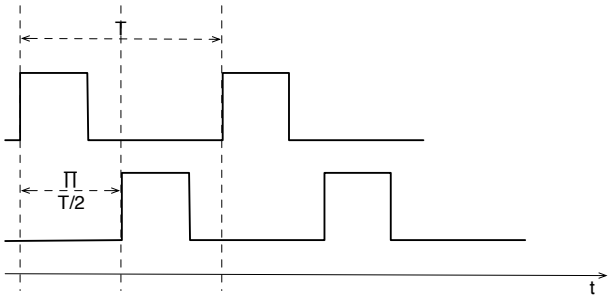


Figura 1.2: Forma d'onda sfasata di 45°

1.1 Descrizione di forme periodiche in VHDL

Il VHDL consente di programmare forme periodiche (quali un segnale di clock) in vario modo ricorrendo a tecniche che fanno riferimento a specifici costrutti del linguaggio. Di seguito sono riportati taluni esempi di forme periodiche utilizzate spesso nei moduli di test bench, sviluppati secondo un approccio di tipo comportamentale e strutturale.

Usando l'approccio comportamentale, la tecnica utilizzata per descrivere un clock fa ricorso al costrutto *process*. In VHDL il *process* permette di realizzare un'unità di programma eseguita in concorrenza con gli altri processi o con le altre istruzioni concorrenti del sistema di cui fa parte. Tutte le istruzioni poste all'interno del blocco (begin. . . end) del corpo di un processo sono eseguite in sequenza. Esse possono fare riferimento a variabili locali al process e a segnali che hanno una visibilità globale e operano in concorrenza. Un processo è attivato all'atto della variazione avvenuta in uno dei segnali in esso utilizzati o, se dotato di lista di sensibilità, all'occorrenza di un evento relativo ai segnali di tale lista e, quindi, l'attivazione del processo avviene solo se risulta vera l'espressione booleana presente nella lista[1].

A titolo d'esempio si riporta in fig.1.3 il codice per la generazione di un clock con duty cycle 50% avente periodo $T = (T/2 + T/2) = (10 + 10) = 20ns$ e in fig.1.4 il pattern del clock.

[1] Tutti i processi di seguito descritti sono stati inseriti nel file *differenti_ clock.vhd* e possono essere simulati nell'ambiente ISE XILINX.

```
--Esempio_1
CLK_Gen_base:

    process(clk_base1)
    begin
        clk_base1<=not clk_base1 after 10ns;
    end process;
    clk_1<=clk_base1;
```

Figura 1.3: Generatore comportamentale di un clock con duty cycle 50% e periodo 20 ns

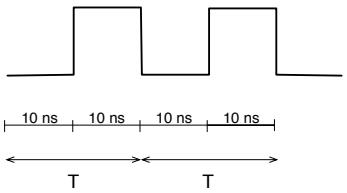


Figura 1.4: Forma del clock generato

Durante la simulazione, il processo CLK_Generator1 è sempre attivo. Esso, dotato di una lista di sensibilità contenente il segnale clk_base1, è riattivato ogni qualvolta si genera una variazione nel valore di clk_base1, cosa che avviene, a ogni attivazione del processo, con l'esecuzione dell'istruzione al suo interno. Tale processo consente di generare forme d'onda con periodo dato dal doppio del valore posto dopo la clausola *after* (nell'esempio il periodo è di $10 \cdot 2 = 20$ nsec a cui corrisponda una frequenza di $1000/20 \cdot 10^6 = 50$ Mhz). Con tale schema di processo non è possibile generare forme con duty cycle diverso dal 50%.

Onde disegnare un segnale periodico di forma generica, si può ricorrere al processo descritto nell'esempio_2:

```
--Esempio_2: generare una forma d'onda con un pattern che si istanzia ogni PATTERN_PERIOD.
--
CLK_Gen_pattern:

    process
    begin
        I1: clk_base2 <= '1' after 10 ns, '0' after 15 ns,
            '1' after 45 ns,'0' after 65 ns;
        I2: wait for PATTERN_PERIOD;
    end process;
    clk_2<=clk_base2;
```

Tale codice genera un pattern che si istanzia ad ogni PATTERN_PERIOD. Fare attenzione al fatto che, essendo assoluti i tempi indicati nelle clausole after e wait, il valore

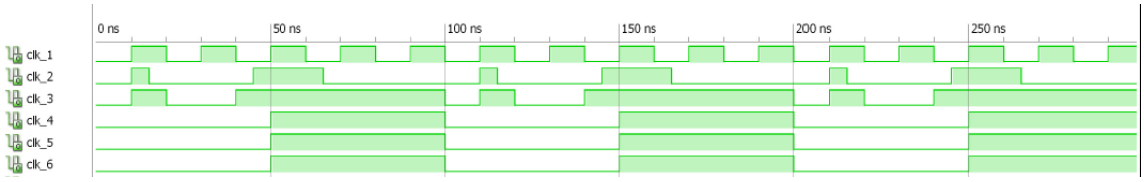


Figura 1.5: Pattern generati con il simulatore Xilinx ISI

di PATTERN_PERIOD deve essere maggiore della durata dell'intero shape altrimenti il pattern non è iterato correttamente.

Nell'esempio riportato il valore di PATTERN_PERIOD deve essere maggiore di 65ns (65 ns è l'ultimo valore della clausola *after* nell'istruzione I1). Nella simulazione del codice riportata in fig.1.5, PATTERN_PERIOD è inizializzato a 100 ns, da cui deriva il diagramma temporale di figura indicato con clk_2.

L'esempio_3 di seguito riportato è analogo all'esempio_2 con la differenza che si opera direttamente con il segnale dell'interfaccia clk_3 e che si usa un pattern differente.

```
--Esempio_3: analogo a esempio_2 con la differenza che si
--opera direttamente con il segnale clk_3
--e con un pattern differente.
clk_gen1:

    process is
    begin
        I1: clk_3<='0', '1' after 10 ns, '0' after 20 ns,
            '1' after 40 ns;
        I2: wait for CLK_PERIOD;
    end process;
```

Un segnale di clock con un preassegnato duty cycle può essere generato ricorrendo all'uso di due parametri che indicano, rispettivamente, la durata del segnale nello stato 0 e nello stato 1. Tali parametri possono essere definiti come:

```
constant Tpieno: time := Duty_cycle*CLK_PERIOD;
constant Tvuoto: time := (1.0-Duty_cycle)*CLK_PERIOD;
```

Nell'esempio_4 descritto di seguito, è riportato il codice per la generazione di un clock di periodo 20 ns e con un duty cycle del 50%.

```
--Esempio_4: generare un pattern  con duty 50% e periodo 20 ns.
clk_gen1a:

    process is
6   begin

        clk_4<='0', '1' after Tvuoto, '0' after CLK_PERIOD;
        wait for CLK_PERIOD;
    end process;
```

Un modo equivalente per generare un clock con duty cycle 50%, riportato nell'esempio_5 seguente, fa uso di un'attesa indicata mediante la costante CLK_PERIOD/2:

```
--Esempio_5: genera un clock con duty cycle del 50%
clk_gen2:
process
begin

    clk_5 <= '0';
    wait for CLK_PERIOD/2;
    clk_5 <= '1';
    wait for CLK_PERIOD/2;

end process;
```

L'esempio_6, introducendo i due parametri Tpieno e Tvuoto, consente di generare forme periodiche con un qualunque duty cycle.

```
--Esempio_6: genera un clock con duty cycle dato da:
--                Tpieno/(Tpieno+Tvuoto)
--
clk_gen3:

    process
    begin
        clk_6 <= '0';
        wait for Tpieno;
        clk_6 <= '1';
        wait for Tvuoto;
    end process;
```

Il processo seguente genera un treno di cicli di clock aventi un preassegnato duty cycle, mediante l'uso di un costrutto for che conteggia i num_cycle=32 cicli.

```
--Esempio_7:
constant num_cycles : integer := 32;
signal clock : std_ulogic := '1';
-- architecture statement part
process
begin

    for i in 1 to num_cycles
    loop
        clock <= not clock;
        wait for Tpieno;
        clock <= not clock;
        wait for Tvuoto;
    end loop;

end process;
```

Il codice seguente descrive un dispositivo che conta i fronti di salita di un clock. L'interfaccia ha in ingresso il segnale di clock *clkin* e un segnale di abilitazione del conteggio *en* e in uscita fornisce il segnale *clkcount* che riporta su un intero con un range di 15

cifre i fronti conteggiati. Il segnale di *en*, se posto a zero, resetta il conteggio. Si faccia attenzione all'uso del tipo intero, il numero delle sue cifre deve essere tale da poter gestire i conteggi senza andare in overflow.

```
-- Esempio_8:
library ieee;
use ieee.std_logic_1164.all;
entity clkgen isport(clkin: in std_logic; en: in std_logic;
                     clkcount: out integer range 0 to 15);
end clkgen;
architecture behavioral of clkgen is
signal count16: integer range 0 to 15;
begin
    process(en,clkin)
    begin
        if (en='0') then
            count16<=0;
        elsif(clkin'event and clkin='1') then
            count16<=count16+1;
        end if;
    clkcount<=count16;
    end process;
end behavioral;
```

Tale tipologia di circuito può essere utilizzata per realizzare dei divisori di frequenza in grado di generare, a partire da un segnale a frequenza elevata, detto base dei tempi, uno o più segnali a frequenze più basse da utilizzare all'interno di un sistema digitale. Ad esempio, di seguito è mostrato il codice per dividere la base dei tempi *clock_ in* per un fattore *div* inizializzato con la direttiva GENERIC. Il processo di divisione è reso sensibile al fronte di salita di *clock_ in*, il cui verificarsi produce l'incremento del conteggio. Contati $\frac{div}{2} - 1$ fronti, il clock *clock_ o* commuta il suo valore generando, in tal modo un clock con un periodo *div* volte maggiore di quello della base dei tempi. Nell'esempio descritto il divisore genera , a partire da clock_in di frequenza f e periodo T, un clock clock_out di frequenza $10^6 f$ e periodo $10^{-6}T$.

```
Esempio_9:
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity divider is

    generic( div : integer := 10000000 );
    Port ( clock_in : in STD_LOGIC; clock_out : out STD_LOGIC);

end divider;
architecture Behavioral of divider is

    signal counter : integer range 0 to div/2-1 := 0;
    signal clock_o : std_logic := '0';

begin

    clock_out <= clock_o;
    clock_divider: process(clock_in)
    begin


        if rising_edge(clock_in) then counter <= counter + 1;
            if counter = div/2-1 then
              clock_o <= not clock_o;
              counter <= 0;
            end if;
        end if;
    end process;

end Behavioral;
```

Nelle due figure seguenti si riporta il listato vhdl usato per generare tutti i vari tipi di clock e forme sopra descritti e simulato nell'ambiente ISE XILINX con il simulatore ISI.

1.2 Sintesi di un generatore di clock

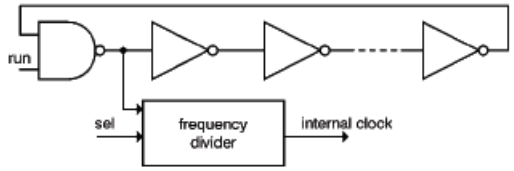
Per sintetizzare su di un dispositivo programmabile (ASIC o FPGA) un clock, occorre descriverlo in VHDL in modo tale da essere sicuri di generare un'architettura funzionante secondo lo schema strutturale che si intende realizzare. In fig.1.8 è riportato lo schema strutturale di un generatore di clock composto da un circuito oscillatore e da un divisore di frequenza. L'oscillatore è realizzato mediante una catena chiusa di k invertitori e una porta nand. Per generare l'oscillazione l'intera catena dei k inverter più la nand deve essere dispari, per cui deve essere k pari. La porta nand è utilizzata per controllare, per tramite del segnale run la generazione o meno del segnale generato, un'onda quadra di periodo dato da $2 \cdot (k\tau_I + \tau_N)$, essendo τ_I il ritardo di ciascun inverter e τ_N quello della porta Nand.

```
----codice progetto clock-generator-pattern.xise----
--
library IEEE; use IEEE.std_logic_1164.all;
entity periodic_form is port(clk_1, clk_2, clk_3, clk_4, clk_5,
                             clk_6:out std_logic);
end entity periodic_form;
architecture behavioral of periodic_form is
constant CLK_PERIOD: time := 100 ns;
constant PATTERN_PERIOD: time := 100 ns;
constant Duty_cycle: real := 0.5;
constant Tpieno: time := Duty_cycle*CLK_PERIOD;
constant Tvuoto: time := (1.0-Duty_cycle)*CLK_PERIOD;
signal clk_temp: std_logic := '0';
signal clk_base1, clk_base2: std_logic := '0';
begin
--Esempio_1: genera un clock con duty 50% e periodo 20 ns

    CLK_Gen_base: process(clk_base1)
    begin
        clk_base1<=not clk_base1 after 10ns;
    end process;
    clk_1<=clk_base1;

----
--Esempio_2: genera un pattern che si instanzia ogni PATTERN_PERIOD.
--Fare attenzione che i tempi indicati nelle clausole after sono
--assoluti e non relativi. Il valored di PATTERN_PERIOD deve essere
--maggiore della durata del pattern. Se esso è inferiore, il pattern
--non viene iterato correttamente.
--Si fa uso di un segnale interno clk_base che alla fine della
--esecuzione del processo è caricato sul segnale dell'interfaccia
--clk_2.
----

    CLK_Gen_pattern: process is
    begin
        I1: clk_base2 <= '1' after 10 ns, '0' after 15 ns,
                         '1' after 45 ns, '0' after 65 ns;
        I2: wait for PATTERN_PERIOD;
    end process;
    clk_2<=clk_base2;

----
--Esempio_3: analogo a esempio_2 con la differenza che si opera
--direttamente con il segnale dell'interfaccia clk_3 e con un
--pattern differente.
clk_gen1: process is
begin

    I1: clk_3<='0', '1' after 10 ns, '0' after 20 ns,
              '1' after 40 ns;
    I2: wait for CLK_PERIOD;

end process;
----
```

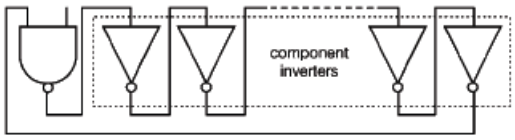
Figura 1.6: Listato degli esempi descritti/1

```
----codice progetto clock-generator-pattern.xise/2----
--Esempio_4: genera un clock iterato con duty 50% e periodo 20 ns
clk_gen1a: process is
begin

    clk_4<='0', '1' after Tvuoto, '0' after CLK_PERIOD;
    wait for CLK_PERIOD;

end process;
----
--Esempio_5: genera un clock con duty cycle del 50%
clk_gen2: process is
begin

    clk_5 <= '0';
    wait for CLK_PERIOD/2;
    clk_5 <= '1';
    wait for CLK_PERIOD/2;

end process;
----
--Esempio_6: genera un clock con duty cycle dato da
--          Tpieno/(Tpieno+Tvuoto)
clk_gen3: process is
begin

    clk_6 <= '0';
    wait for Tpieno;
    clk_6 <= '1';
    wait for Tvuoto;

end process;
end architecture behavioral;
```

Figura 1.7: Listato degli esempi descritti/2



(a) Schema strutturale di un generatore di clock



(b) Catena aperta di inverter come componente

Figura 1.8: Schema strutturale di un generatore di clock e catena aperta di inverter come componente

La fig.1.8.b mostra lo schema dell'oscillatore strutturato come un unico componente invertente costituito da una catena aperta di inverter, che chiamiamo inv_chain, di cui si vuole effettuare la sintetisi sul silicio. La sintesi di un tale circuito è fatta mediante un compilatore che genera una rappresentazione intermedia simulabile in modo behavioral e una rappresentazione *schematica* (post translate) del dispositivo con uso di componenti logici non specifici. A tale fase segue quella del mapping tecnologico che associa a ciascuno dei componenti logici un circuito sintetizzabile (fase di map) e la fase di mappatura dell'intero circuito nel dispositivo selezionato, ad esempio un FPGA, con risoluzione di tutte le interconnessioni fra i componenti che lo compongono ottenuta mediante applicazione di complessi algoritmi di routing (fase di route).

Durante tutte le fasi della traduzione, da una descizione VHDL di un dispositivo alla sua rappresentazione sintetizzabile (memorizzata in un file di tipo *.bit contenete tutte le informazioni necessarie alla sintesi), il compilatore VHDL effettua anche diverse forme di ottimizzazione che iteressano lo spazio (numero di gate utilizzate) e/o il tempo (ritardi dei percorsi critici associati alle varie funzioni logiche da implementare). Tali ottimizzazioni ritenendo inutili funzionalmente i k inverter del generatore di fig.1.8, effettuano la loro sostituzione con un cortocircuito fra la linea di ingresso della catena e quella d'uscita connessa all'ultimo invertitore, se in numero pari, o con un solo inverter, se dispari. Per ovviare a tale inconveniente, occorre indicare esplicitamente che non è richiesta per il circuito alcuna ottimizzazione ricorrendo all'uso di specifiche direttive al compilatore quali quelle mostrate di seguito:

```
attribute KEEP : string;
attribute KEEP of inv_chain : signal is "true";
```

La prima direttiva serve a dichiarare che la parola chiave KEEP è una stringa di caratteri, mentre la seconda associa KEEP al segnale inv_chain, indicando, in tal modo, al compilatore di procedere senza effettuare l'ottimizzazione dei codici a esso associati.

L'esercizio riportato nelle fig.1.9 e successive realizza un generatore di clock la cui frequenza è determinata dal numero di inverter utilizzati, tale numero è indicato con il parametro N definito nel modulo inv_chain, nell'esempio è posto a 64 (pertanto, il periodo è $2 \cdot 64\tau_I + \tau_N$). Il componente count_clock16 è stato aggiunto per contare i fronti di salita del clock generato e usato come base per realizzare un divisore di clock. Si fa, inoltre, uso di una package contenente le gate di base utilizzate per la simulazione behavioral e che, in un'implementazione strutturale, sono sostituite dai componenti della libreria tecnologica prescelta.

1.3 Distribuzione dei clock in un sistema digitale

La distribuzione del segnale di clock in un sistema digitale è effettuata con collegamenti elettrici fra il nodo circuitale in cui esso è generato e quelli in cui esso è utilizzato e che avvengono seguendo differenti percorsi. Tali percorsi si differenziano per la distanza e per le caratteristiche elettriche (resistive, capacitive e induttive) delle linee di collegamento. Giocano, inoltre, un ruolo attivo anche gli effetti termici, meccanici e le imperfezioni nei materiali utilizzati per realizzarle, che producono alterazioni che si traducono in ritardi e deformazioni della forma d'onda generata rispetto a quella fruibile nei vari punti di destinazione. In fig.1.14 è riportato lo schema esemplificativo della distribuzione di un clock, Clk, ai vari sottosistemi.

Durante il trasporto il clock, come detto, subisce dei ritardi che si traducono in variazioni di fase tra il valore del segnale generato in un punto del circuito è quello ricevuto in un altro. Tali ritardi sono noti come *skew (*il termine inglese è traducibile con *distorto* o *non preciso*) . Lo skew di un segnale f(t), che da un punto P_0 con valore $f(t_0)$, si propaga verso due differenti punti del circuito, P_1al tempo $(t_0 + \tau_1)$ e P_2 al tempo $(t_0 + \tau_2)$, essendo τ_1 e τ_2 i rispettivi ritardi, è dato dalla differenza di tempo: $\Delta t_{skew} = \tau_2 - \tau_1$. La figura mostra un diagramma di tempificazione con indicato il segnale di clock Clk e i segnali fruibili nei sottosistemi C1, Ci, Cn con skew dati dai ritardi τ_1, τ_i e τ_n.

Con riferimento al trasferimento fra registri, così come schematizzato in fig.1.15, lo skew può assumere un valore positivo o negativo, dipendentemente dal fatto che i fronti del clock utilizzati per l'abilitazione dei flip flop sorgente e destinazione, siano abilitati nella sequenza corretta sorgente->destinazione o in quella inversa destinazione->sorgente. Nel primo caso si ha uno sfasamento tra il fronte attivo del clock di F_2e quello di F_1 positivo, nel secondo caso negativo, così come indicato rispettivamente nella parte superiore e inferiore della figura..

In fig.1.16, è riportato l'esempio di distribuzione del clock relativamente a trasferimenti fra flip-flop connessi in cascata su due linee indipendenti. Ad ogni impulso di clock, il

valore del bit proveniente da una rete combinatoria posta nello stadio i-esimo e sostenuto dal corrispondente flip-flop che attua una trasformazione funzionale, è propagato a valle verso il flip-flop dello stadio i+1-esimo. Il circuito opera sincronizzando prima l'attivazione del flip-flop in posizione i-esima e successivamente, dopo un intervallo di tempo maggiore al tempo di propagazione della rete combinatoria i+1-esima, del successivo flip-flop i+1-iesimo. Per un corretto funzionamento lo skew deve essere positivo, e cioé, deve prima arrivare l'abilitazione a F1 e poi a F2.

In figura sono, inoltre, evidenziati gli skew locali che interessano un'unica linea di ritaro e quelli globali che relazionano sfasamenti di segnali indipendenti.

1.3.1 Phase Locked Loop (PLL) e Delay-Locked Loop (DLL)

La distribuzione dei clock in un circuito su più punti è affetta da skew fra i vari segnali derivati da un unico clock comune. Inoltre, stante l'impossibilità di eliminare completamente i fenomeni di deriva di fase, negli oscillatori, al fine di poterne controllare la stabilità, in frequenza e fase, della forma periodica generata, si ricorrere a circuiti di due principali tipologie: *Phase Locked loop* (PLL) e *Delay-Locked Loop* (DLL). Entambi i i circuiti operano confrontando il clock generato con un clock di riferimento e usano forme di controreazione negativa per controllarne la frequenza e la fase. In particolare, il segnale di errore, opportunamente filtrato, provvede a effettuare l'*aggancio in frequenza* e, quindi, il controllo della fase del segnale generato.

La differenza fondamentale fra le due tipologie di circuiti PLL e DLL, sta nel tipo di componente utilizzato per svolgere l'azione di regolazione della fase che, nel caso dei circuiti PLL, fa uso di un oscillatore controllato dalla tensione (VCO), mentre nel caso dei DLL di una linea a ritardo variabile, così come evidenziato in fig.1.19. In particolare, il circuito PLL fa uso di un clock esterno di riferimento immesso sulla linea CLKIN, pertanto, stabile in frequenza (si ricorre ad un oscillatore quarzato), e isofrequenziale (o multiplo in frequenza) al clock CKOUT che si vuole controllare all'atto dell'aggancio della fase. In un PLL, un rivelatore di fase confronta le frequenze dei due segnali di ingresso e genera un segnale di errore proporzionale alla loro differenza. Agganciata la frequenza, il VCO procede con l'aggancio di fase e, quindi, con la sua successiva regolazione. Il segnale di errore, dopo essere stato condizionato elettronicamente con un filtro passa basso, regola l'oscillatore VCO (Voltage Controlled Oscillator) adibito a generare il segnale CLKOUT, reazionato all'ingresso del rilevatore di fase. Uno shift di fase ($\pm\Delta\phi$) fa aumentare il segnale di errore che pilota il VCO in modo da compensare lo spostamento di fase evidenziato dall'errore (lo slittamento di fase aumenta se l'errore è negativo e diminuisce se positivo). L'oscillatore di riferimento esterno utilizzato nel PLL introduce forme di instabilità che degradano le prestazioni del PLL quando si tenta di compensare il ritardo dovuto alla distribuzione del clock. Di contro, l'architettura del DLL che usa lo stesso segnale di clock che si vuole controllare, risulta essere più stabile

per la compensazione e il condizionamento del clock, ma meno flessibile qualora venga impiegata per sintetizzare clock operanti con nuove frequenze.

Tali circuiti possono essere realizzati sia in modo analogico che digitale, ovviamente, quelli di tipo analogico permettono di ottenere migliori prestazioni e consentono di operare a frequenze più elevate, quelli digitali, di contro, sono più flessibili e semplici da progettare, meno critici nel funzionamento e facilmente integrabili all'interno dei chip VLSI.

1.3.2 Il Digital Clock Manager (DCM)

Il digital clock manager è un dispositivo in grado di eliminare il clock skew, e quindi di aumentare le prestazioni del sistema, ricorrendo a uno shift di fase del clock generato, ottenuto ritardandone l'inizio di una frazione o di suoi multipli o sottomultipli interi, in modo da tenere conto dei ritardi medi rilevati nel path di trasmissione del segnale. I DCM, al fine di sintetizzare un nuovo clock a frequenza diversa, possono effettuare operazioni di moltiplicazione o divisione della frequenza del clock in ingresso con quest'ultimo sincronizzato.

Si riporta di seguito la descrizione del DCM disponibile negli FPGA della famiglia Spartan3e di Xilinx e il cui schema funzionale è riportato in fig.1.21.

L'architettura del DCM si compone di quattro parti funzionali:

- un delay loocked loop (DLL) per l'aggancio in frequenza e fase fase del clock di uscita rispetto a quello di ingresso;
- un sintetizzatore digitale, il digital frequency synthesizer (DFS), per la generazione di clock di frequenze preassegnata determinata a partire dal clock di ingresso CLKIN modificato per tramite del rapporto di due numeri interi, di un moltiplicatore (CLKFX_MULTIPLY) e di un Divisor (CLKFX_DIVIDE);
- un phase shifter (PS), per determinare, anche dinamicamente, lo sfasamento che il clock in uscita deve avere rispetto a quello in ingresso;
- una logica di Status che codifica tutte le informazioni che caratterizzano lo stato corrente del DCM.

Ill DLL è adibito, come visto, al controllo dello skew dei segnali in modo da distribuire, nei vari punti del circuito segnali di clock con un effettivo ritardo a zero.

I segnali di ingresso che interessano il DLL sono: CLKIN, per immettere il clock generato internamente o esternamente, da distribuire e CLKFB, per immettere il segnale prelevato sul punto di distribuzione reazionato al fine di determinarne nel DLL lo scostamento di fase.

Quelli d'uscita sono CLK2X, a frequenza doppia, generato dal duplicatore di clock, e CLK2X180 in controfase ; il segnale proveniente dal divisore di frequenza CLKDV; i quattro segnali CLK0 , CLK90, CLK180, CLK270, in quadratura l'uno con il precedente

(Quadrant Phase Shifted Outputs functions) e i due segnali CLKFX e CLKFX180, in opposizione di fase, generati dal sintetizzatore di frequenze DFS. Il sintetizzatore di frequenze può anche operare indipendentemente dal DLL e generare clock non sincronizzati con CLKIN, reazionato su CLKFB.

Il digital frequency synthesizer (DFS), può generare, quindi, clock unitamente o separatamente al DLL. Nel primo caso le frequenze generate sono sincronizzate con il clock CLKIN, nel secondo caso ne sono completamente indipendenti. Il DFS può essere utilizzato per generare un clock interno al dispositivo FPGA. Il range di frequenze dipende dal particolare dispositivo utilizzato.

Il phase shifter (PS) controlla la relazione di fase fra CLKIN e tutti e nove i segnali del DCM, che risultano, pertanto, sfasati simultaneamente dello stesso valore di fase. Lo sfasamento può essere tanto positivo che negativo ed è di una frazione del periodo del clock di ingresso. Il valore dello sfasamento può essere determinato staticamente in fase di progetto e caricato dinamicamente nel dispositivo FPGA con la fase di programmazione di una configurazione, oppure essere definito dinamicamente per quanti di $\frac{1}{256}$ del periodo di CLKIN per taluni modelli o per step programmabili nell'intervallo tra 15 e 35 ps per altri.

I segnali di ingresso al PS sono PSINCDEC che indica un'operazione di shift di un quanto negativo (se posto al valore 0) o positivo (se posto al valore 1); PSEN, che abilita le operazioni di PS se posto a uno e le disabilita se posto a zero; PSCLK è il clock di ingresso al PS. I segnali di uscita sono PSDONE che indica se posto a 0 che è in corso una fase di shift della frequenza e se posto a 1 che non è attiva alcuna fase di shift o che è stata completata la fase di shift richiesta.

Il sottositema dedicato alla logica di stato presenta in ingresso il segnale di reset asincrono (RST) per resettare il DCM e in uscita il segnale LOCKED, usato per indicare l'aggancio di fase del DCM con il CLKIN, e la variabile di 8 bit STATUS che codifica con i bit [0], [1], e [2] lo stato del DCM, mentre i flag [3]-[7] non sono usati.

In fig.1.22 è riportato uno schema delle operazioni di shift e i segnali coinvolti in esse.

1.3.3 Esempi di generazioni di clock mediante DCM

<riferirsi agli esercizi messi sul sito>

1.4 Esempio IPcore DCM

La libreria di Ip core della digilent fornisce vari tipi di sistemi con funzionalità differenti e organizzati per tipologia. Tali IP core possono essere utilizzati all'interno di un progetto ISE utilizzando il tool core generator tramite cui generare tutti i file, inclusi i vhd necessari alla sintesi dell'IP core. Tali file vanno aggiunti agli altri file del progetto da sviluppare.

Nel caso dell'esempio che si vuole mostrare, ci si riferirà all'IP core DCM_SP il cui schema è mostrato in fig.xx

Lo schema della fig.1 mostra l'interfaccia del componente DCM e una maschera di configurazione di aiuto all'utente per la sua sintesi.

1.4.1 Descrizione dell'architettura del sistema da sintetizzare

E' possibile costruire un progetto inserendovi un ip core, facendo generare un test bench a ISE e effettuarne una simulazione. Per la sintesi e l'esecuzione sulla board Nexys occorre utilizzare un file *.ucf (di seguito nel riquadro è riportato un pezzo del file ucf per la Nexys2) e definire le necessarie connessioni da e per la board.

```
# clock pin for Nexys 2 Board
NET "clock_50MHz" LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type = GCLK, Sch name = GCLK0
# Leds
NET "clock_out_div"  LOC = "J14"; # Bank = 1, Pin name = IO_L14N_1/A3/RHCLK7, Type = RHCLK/DUAL, Sch name =
NET "clock_out_div_2"  LOC = "J15"; # Bank = 1, Pin name = IO_L14P_1/A4/RHCLK6, Type = RHCLK/DUAL, Sch name
NET "clock_out_div_3"  LOC = "K15"; # Bank = 1, Pin name = IO_L12P_1/A8/RHCLK2, Type = RHCLK/DUAL, Sch name
# Switches
NET "en" LOC = "G18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW0
NET "clock_out" LOC = "L15"; # Bank = 1, Pin name = IO_L09N_1/A11, Type = DUAL, Sch name = JA1
```

La Nexys2 è dotata di un oscillatore a quarzo (SG8002JF) che genera un segnale alla frequenza di 50MHz connesso al piedino B8 dell'FPGA utilizzato come base dei tempi per tale dispositivo. Nella scheda è anche presente anche un socket per connettere un ulteriore oscillatore esterno, il cui segnale di uscita è connesso al piedino U9.

Il sistema che si vuole sintetizzare è composto, così come mostrato al par.1.2, da un generatore di clock sintetizzato all'interno del componente FPGA mediante una catena di inverter chiusa da una nand a cui è anche connesso il segnale En che abilita con il suo valore 1-attivo il funzionamento del clock.

Il clock generato è inviato a un DCM per generare altri clock

Simulazione

Si effettuano tre tipi di simulazione, la behavioral, la post-mapping e qualla post sintesi. Di seguito i risultati come si può notare dai dati riportati, prelevati dai pattern delle fig xx relativi alle tre simulazioni effettuate, il tempo di ritardo della catena di inverter, che determina il periodo del clock generato, è inferiore rispetto a quello della simulazione post sintesi che include anche i ritardi dei collegamenti delle varie lut in cui sono sintetizzati gli inverter. Si ha una differenza di 146,068 ns - 99,144 ns = 46,924 ns, per cui mediamente si può asserire che tale è il ritardo dei collegamenti che, quindi, pesa circa il 32% del ritardo complessivo (46,924/146,068=0,321). Nel caso della simulazione behavioral e post translate, non si ha un ritardo nullo della catena di inverter in quanto tali componenti sono stati modellati senza alcuna clausa "after" e non hanno, pertanto alcun ritardo.

	Behav	post translate	
periodo (nsec)	0	0	
frequenza (MHz)			
Commento	gli inv non hanno ritardo	gli inv non hanno ritardo	non si tiene conto dei ritardi dovuti ai collegan

```
----------------------------------------------------------------
-- Company:Dip. di Informatica e Sistemistica
   Univ. di Napoli Federico II
-- Engineer: Antonino Mazzeo
-- Create Date: 16:34:44 10/13/2012
-- Design Name: clock-generator
-- Module Name: clock_generator/clk-gen-TB.vhd
-- Project Name: clock_generator
-- Target Device: spartan 3e 1200
-- Tool versions:
-- Description:
-- VHDL Test Bench Created by ISE for module: clock_generator
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- Notes:
----------------------------------------------------------------------
LIBRARY ieee; USE ieee.std_logic_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY clk_gen_TB IS GENERIC(N: Integer:=4);
END clk_gen_TB;
ARCHITECTURE behavior OF clk_gen_TB IS
-- Component Declaration for the Unit Under Test (UUT)

     COMPONENT clock_generator PORT(enable : IN std_logic;
                                    clk : OUT std_logic );
     END COMPONENT;
     COMPONENT count_clock16 PORT(clkin: in std_logic;
                                  en: in std_logic;
     clkcount: out integer range 0 to 15);
     END COMPONENT;

--Inputs signal enable : std_logic := '0';

     signal clkcount : integer range 0 to 15;

--Outputs signal

     clk : std_logic;

BEGIN
-- Instantiate the Unit Under Test (UUT)


     uut: clock_generator PORT MAP ( enable => enable, clk => clk );
     uut2: count_clock16 PORT MAP ( clkin=>clk, en=>enable,
                                    clkcount=>clkcount);

-- Stimulus process

     stim_proc: process begin
     -- hold reset state for 100 ns.
         wait for 100 ns;
     -- insert stimulus here
         enable<='1';
         wait;
     end process;

END;
```


Figura 1.9: Listato del clock-generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity inv_chain is

    GENERIC(N: Integer:=64);
    Port ( chain_in : in STD_LOGIC; chain_out : out STD_LOGIC);

end inv_chain;
--
architecture structural of inv_chain is

    attribute BOX_TYPE : string ;
    component INV port(i1: in STD_LOGIC; o1: out STD_LOGIC);
    end component;
    for all: INV use entity work.inv(single_delay);
    signal inv_io: STD_LOGIC_VECTOR (N downto 0);
    signal enable: std_logic;

begin

    gen1: for i in N - 1 downto 0 GENERATE
    begin
        i1: inv port map(inv_io(i+1), inv_io(i));
    end GENERATE gen1;
    chain_out<=inv_io(0);
    inv_io(N)<=chain_in;

end structural;
```

Figura 1.10: Listato inv_chain del clock-generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM; use UNISIM.VComponents.all;
entity inverter_chain is generic ( N : integer := 8 );

    Port ( i : in STD_LOGIC; o : out STD_LOGIC); end inverter_chain;

architecture Structural of inverter_chain is
signal inv_connect : std_logic_vector(N downto 0);
attribute KEEP : string;
attribute KEEP of inv_connect : signal is "true";
component INV port(I: in std_logic; O: out std_logic);
end component;
begin

    inv_connect(0) <= i; o <= inv_connect(N);
    inv_chain: for i in 0 to N-1 generate

        inverter: INV port map(inv_connect(i), inv_connect(i+1));
    end generate;

end Structural;
```

Figura 1.11: Listato del componente inverter_chain/3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
package gates is component

    inv port(i1: in STD_LOGIC; o1: out STD_LOGIC);
    end component inv;

--

    component nand2 port(i1,i2: in STD_LOGIC;
        o1: out STD_LOGIC);
    end component nand2;

--

    component nand3 port(i1,i2,i3: in STD_LOGIC;
        o1: out STD_LOGIC);
    end component nand3;

--


    component bit_comparator port (a,b,gt,eq,lt: in STD_LOGIC;
        a_gt_b,a_eq_b,a_lt_b: out STD_LOGIC);
    end component bit_comparator;

end gates;
-------------------------------------------------------------
-- inv gate ----
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
entity inv is port(i1: in STD_LOGIC; o1: out STD_LOGIC);
end inv;
architecture single_delay of inv is
begin

    o1<=not i1 after 20 ps;

end architecture single_delay;
-- nand_2 gate ---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
entity nand2 is port(i1,i2: in STD_LOGIC; o1: out STD_LOGIC);
end nand2;
architecture single_delay of nand2 is
begin

    o1<=i1 nand i2 after 50 ps;

end architecture single_delay;
-- nand_3 gate ---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
entity nand3 is port(i1,i2,i3: in STD_LOGIC;
o1: out STD_LOGIC);
end nand3;
architecture single_delay of nand3 is
begin
o1<=not(i1 and i2 and i3) after 50 ps;
end architecture single_delay;
```


Figura 1.12: Listato della libreria Package gates/1

```
-- bit comparator gate ------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
entity bit_comparator is port (a,b,gt,eq,lt: in STD_LOGIC;
                     a_gt_b,a_eq_b,a_lt_b: out STD_LOGIC);
end entity bit_comparator;
architecture gate of bit_comparator is

     component n1 port(i1: in STD_LOGIC; o1: out STD_LOGIC);
     end component n1;
     component n2 port(i1,i2: in STD_LOGIC;o1: out STD_LOGIC);
     end component n2;
     component n3 port(i1,i2,i3: in STD_LOGIC;
                       o1: out STD_LOGIC);
     end component n3;
     for all: n1 use entity work.inv(single_delay);
     for all: n2 use entity work.nand2(single_delay);
     for all: n3 use entity work.nand3(single_delay);
     signal im1, im2, im3, im4, im5, im6, im7, im8, im9,
            im10: STD_LOGIC;
     begin

-- a_gt_b output

     g0: n1 port map(a, im1);
     g1: n1 port map(b, im2);
     g2: n2 port map(a, im2, im3);
     g3: n2 port map(a, gt, im4);
     g4: n2 port map(im2, gt, im5);
     g5: n3 port map(im3, im4, im5, a_gt_b);

-- a_eq_b output

     g6: n3 port map(im1, im2, eq, im6);
     g7: n3 port map(a, b, eq, im7);
     g8: n2 port map(im6, im7, a_eq_b);

-- a_lt_b output

     g9: n2 port map(im1, b, im8);
     g10: n2 port map(im1, lt, im9);
     g11: n2 port map(b, lt, im10);
     g12: n3 port map(im8, im9, im10, a_lt_b);

end architecture gate;
architecture functional of bit_comparator is

     function fgl(w, x, gl: STD_LOGIC) return STD_LOGIC is
     begin

         return(w and gl) or (not x and gl) or (w and not x);
     end fgl;

function feq(w, x, eq: STD_LOGIC)
return STD_LOGIC is
begin
return(w and x and eq) or (not w and not x and eq);
end feq;
begin
a_gt_b <= fgl(a, b, gt) after 12 ns;
a_eq_b <= feq(a, b, eq) after 12 ns;
a_lt_b <= fgl(b, a, lt) after 12 ns;
end architecture functional;
```


Figura 1.13: Listato della libreria Package gates/2

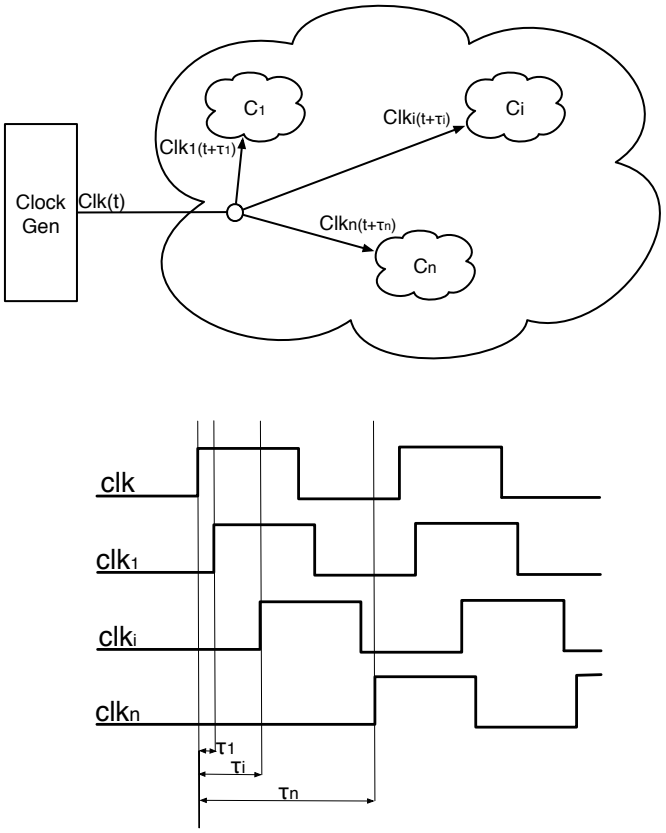


Figura 1.14: Distribuzione clock e ritardi per Skew

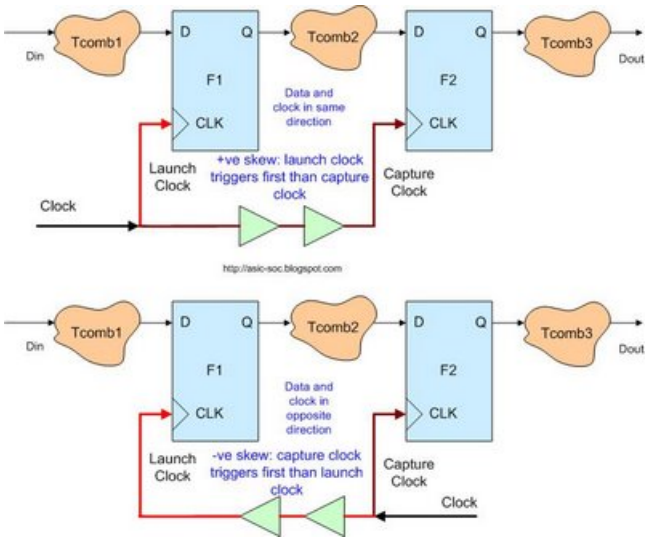


Figura 1.15: Skew nel trasferimento tra registri

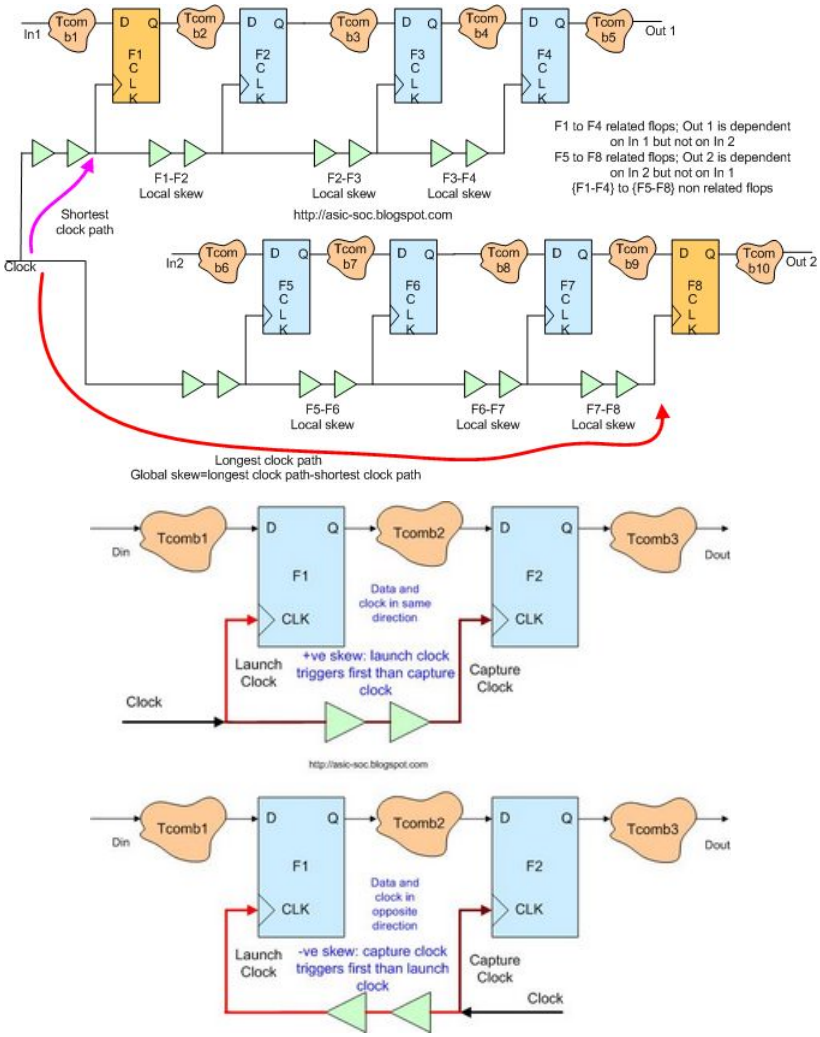


Figura 1.16: Distribuzione del clock in una rete di flip-flop

Figura 1.17: Clock Skew, il solito problema della sincronizzazione degli orologi...

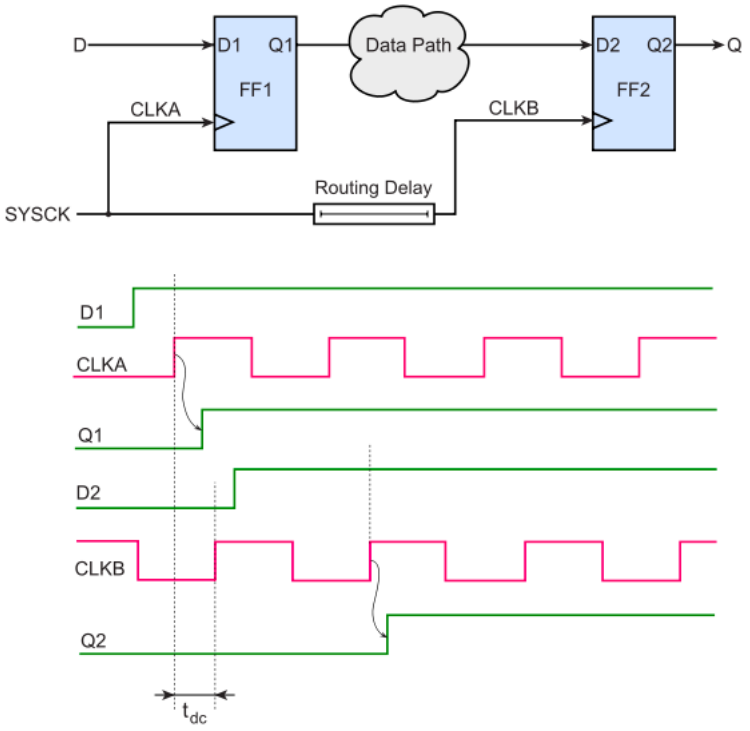


Figura 1.18: Clock skew

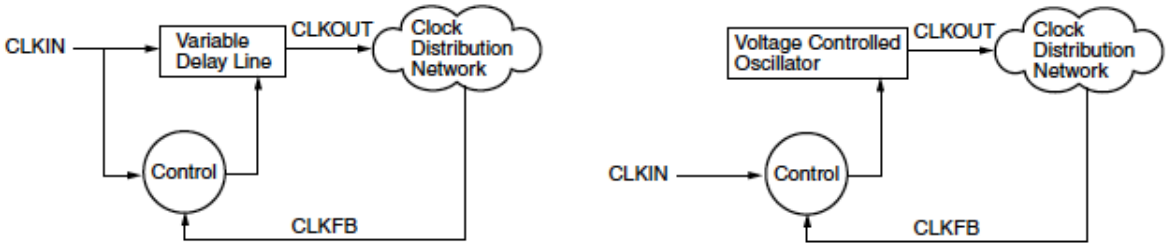


Figura 1.19: Schemi di un circuito DLL (a sinistra) e di un PLL (a destra)

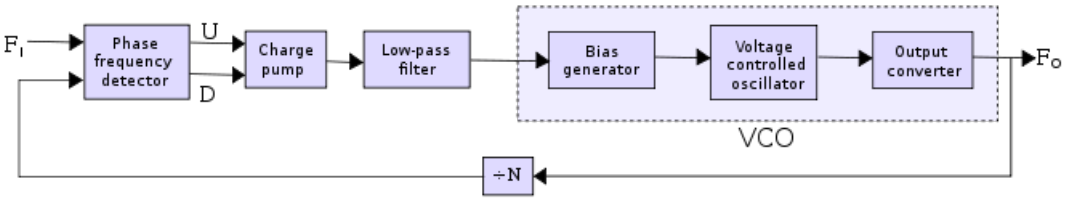


Figura 1.20: Schema di un cicrcuito per il controllo di fase PPL

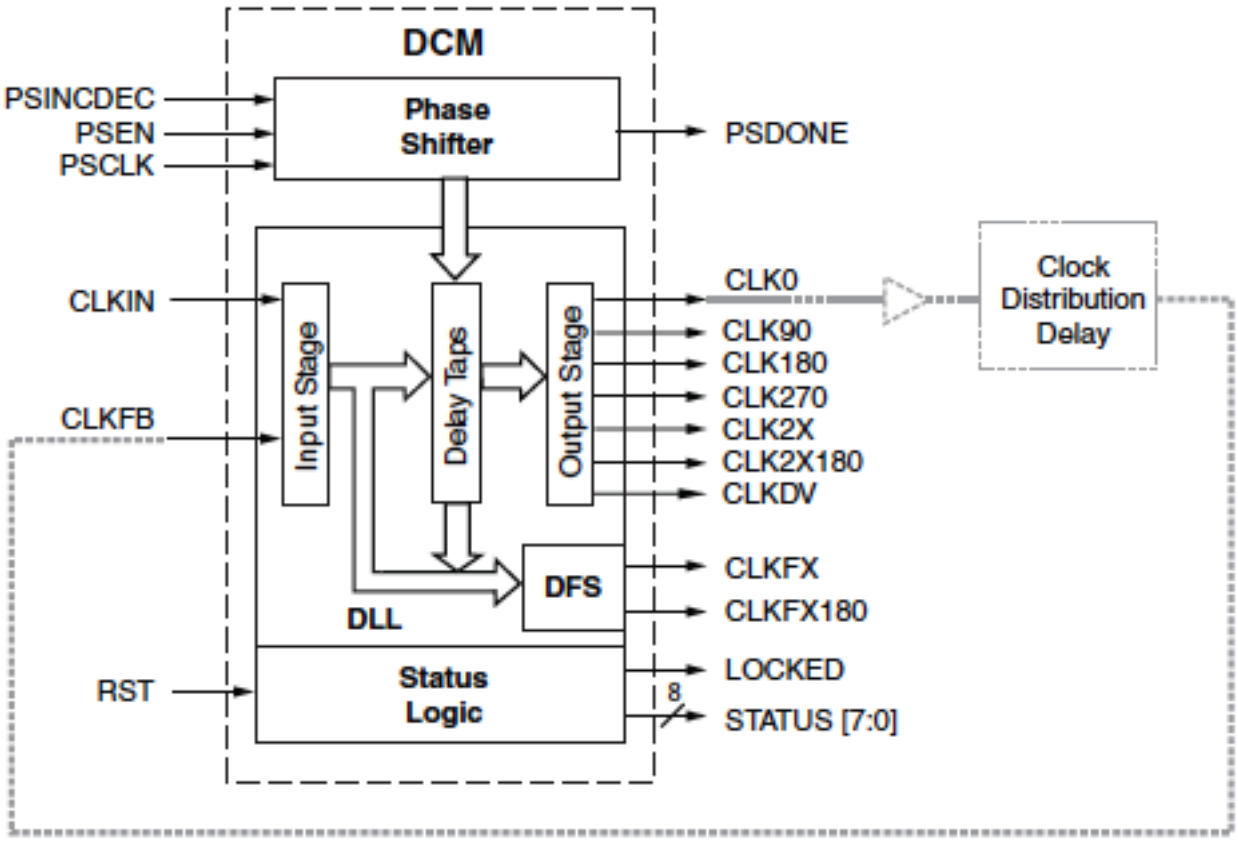


Figura 1.21: Architettura di un DCM

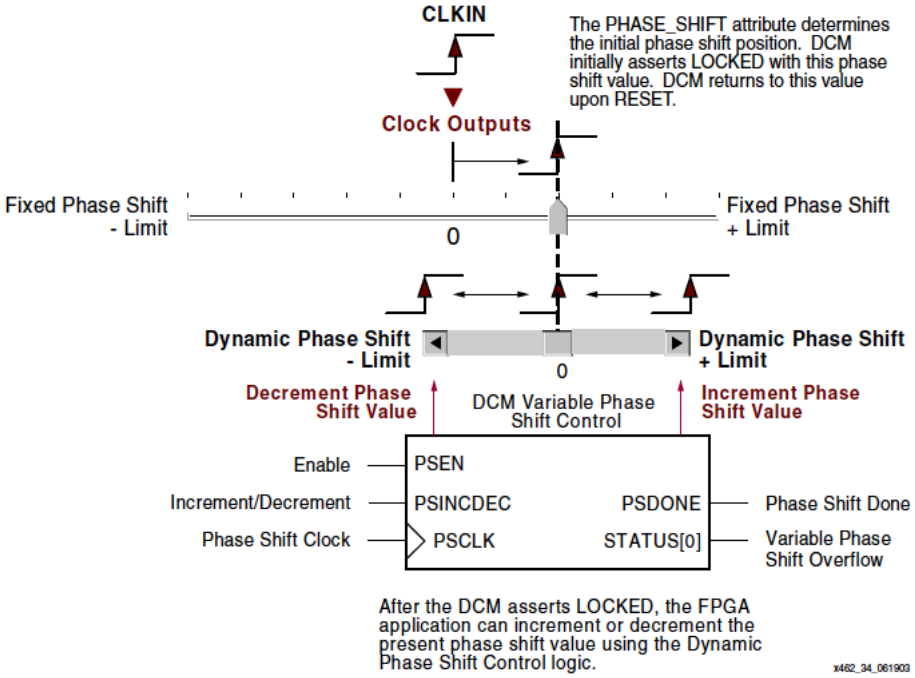


Figure 3-36: **Variable Phase Shift Controls**

Figura 1.22: Controllo Phase Shift

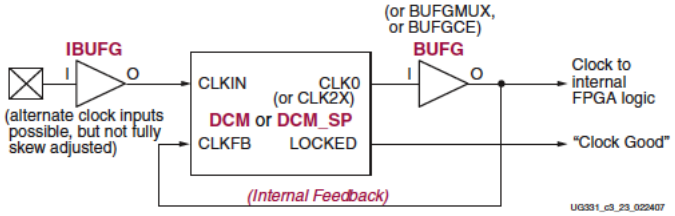


Figura 1.23

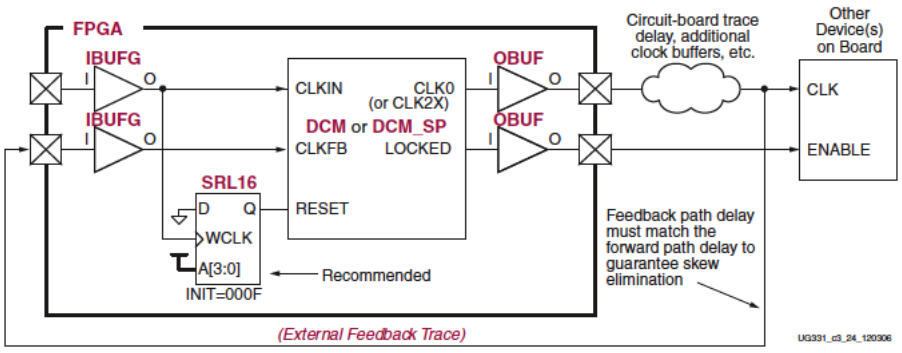


Figura 1.24: Uso dei DCM per la generazione di un clock

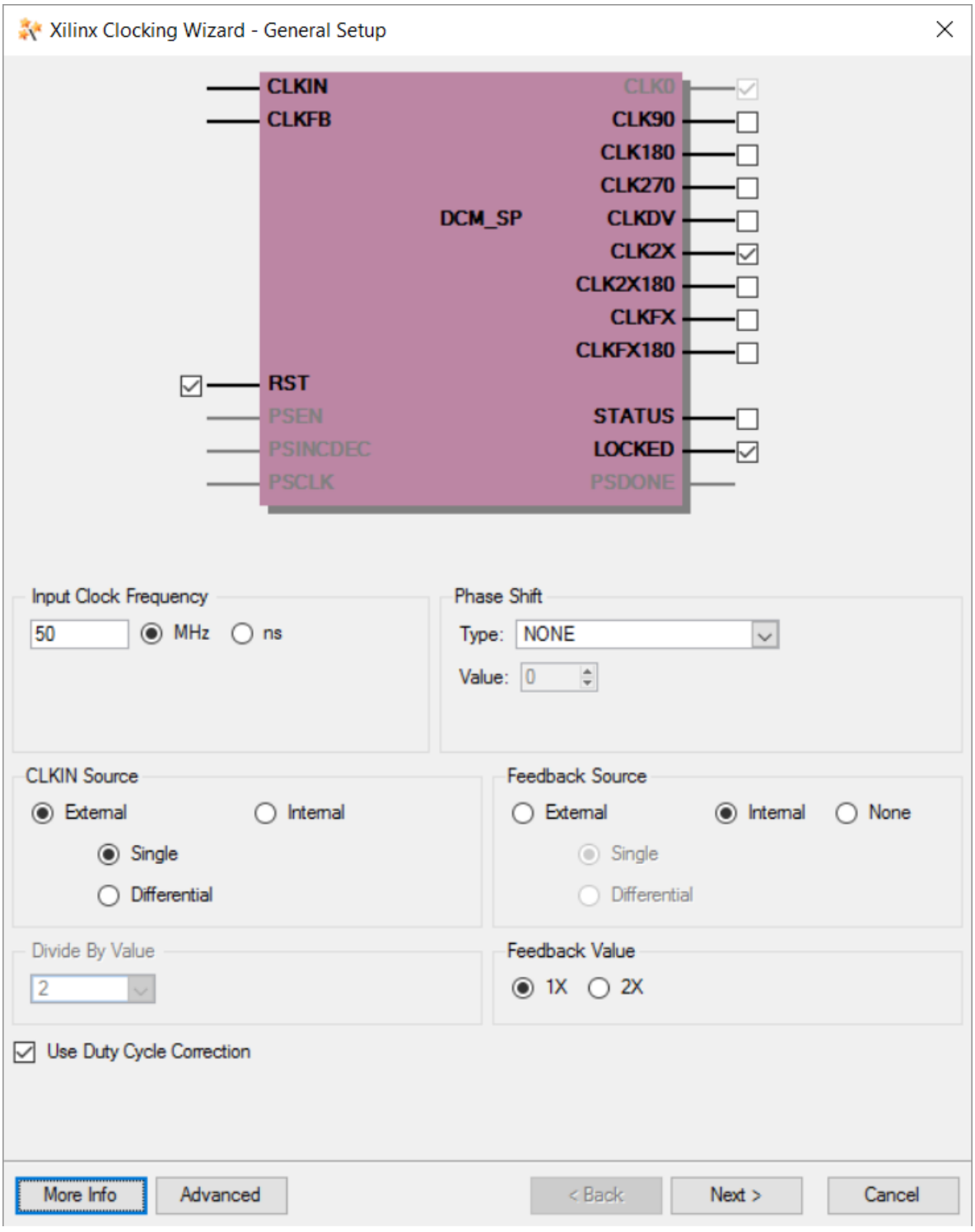
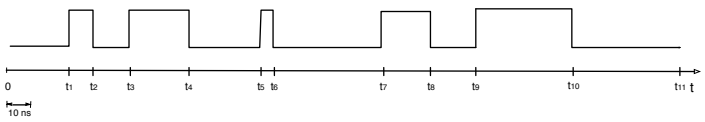
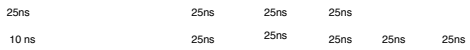


Figura 1.25: Digilent DCM_SP

2 Esercizi

1. Descrivere in VHDL comportamentale, un pattern del tipo mostrato in fig.2.1, ricorrendo alle clausole after n per la tempificazione. Simularne il codice per verificarne la correttezza.





(a) Pattern da sintetizzare

ti	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
ns	0	25	35	50	75	105	110	155	175	195	235	280

(b) Valori dei tempi t_i

Figura 2.1: Pattern della forma da progettare

2. Ripetere l'esercizio n.1 in cui i ritardi sono realizzati mediante circuiti sintetizzati in modo strutturale.

3. Progettare un dispositivo in grado di generare, a partire da un clock base *clk,* il pattern del segnale *forma_ clk* di seguito descritto:

forma_clk	tempo t	valore
stato iniziale	0	0
15 edge ↑	dopo 15 fronti di salita di clk	1
10 edge ↓	dopo 10 fronti di discesa	0
15 edge ↓	dopo 15 fronti di discesa	1
10 edge ↑	dopo 10 fronti di salita	0

Tabella 2.1: andamento del pattern

4. Progettare un generatore di clock per un sistema digitale in grado di fornire, a partire da un clock di 100 MHz con duty cycle=50%, usato come base dei tempi, quattro clock di 90° ciascuno.

5. Progettare un generatore di clock per un sistema digitale in grado di fornire, a partire da un clock di 100 MHz con duty cycle=50%, usato come base dei tempi, quattro segnali di clock di 90° riportati in tabella.

6. Generare in VHDL behavioral, a partire da una base dei tempi a 100 MHz e adoperando dei contatori modulo n, i seguenti segnali di clock: a) Clk_0 a 25 MHz con Duty Cycle 50%; b) Clk_1 a 78 MHz con Duty Cycle 50%; c) Clk_2 a 25 MHz con Duty Cycle 70%; d) Clk_3 a 78 MHz con Duty Cycle 70%. Tale esercizio va ripetuto successivamente ricorrendo all'uso dei DCM presenti nel componente FPGA Spartan3E.

7. Realizzare un orologio con visualizzazione binaria delle ore, minuti e secondi utilizzando la scheda Basys o Nexys2 di Digilent. L'orologio, a partire da un clock di riferimento che opera da base dei tempi di adeguata precisione, genera mediante uso di contatori il secondo, il minuto e l'ora. I minuti (da 1 a 60) e le ore (da 1 a 24) sono visualizzati in formato 8-4-2-1 mediante le due cifre meno e più significative del diaplay a 4 cifre a sette segmenti. I secondi (da 1 a 60) utilizzando i quattro led di peso meno significativo dell'array di otto led presenti nella scheda. Il valore del tempo deve poter essere inizializzato mediante la sequenza obbligata di valori esadecimali dell'ora, minuti e secondi.