Università degli Studi di Padova

Università degli Studi di Padova – Dipartimento di Tecnica e Gestione dei Sistemi Industriali
Corso di Laurea in Ingegneria Meccatronica

Relazione per la prova finale

Sviluppo di un modulo VHDL per modulazioni DPWM e SVM di convertitori AC-DC trifase

Data: 27/05/2024

Relatore: Biadene Davide
Correlatore: Caldognetto Tommaso

Laureandi: Lunardelli Stefano
Milani Francesco
Perazzolo Filippo
Rossi Alessandro

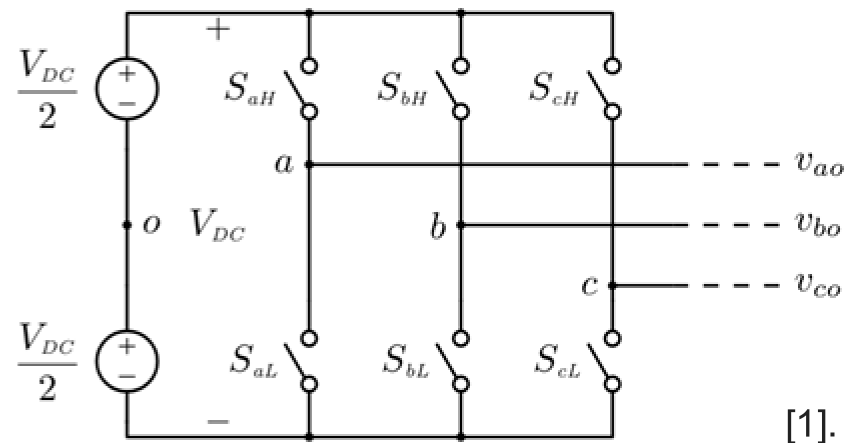
Abstract

La transizione energetica verso fonti rinnovabili rappresenta una delle sfide più significative nell'ambito della sostenibilità globale.

Al centro di questa trasformazione, i convertitori elettronici di potenza sono dispositivi indispensabili per l'integrazione efficace di sorgenti come impianti fotovoltaici e turbine eoliche nella rete elettrica. Questi convertitori sono altresì cruciali nei sistemi di accumulo energetico, come batterie, volani o elettrolizzatori, fungendo da collegamento vitale tra le fonti rinnovabili e la rete.

Il focus di questa tesi è rivolto ai convertitori trifase con tre ponti ad H (3AHB), una configurazione prevalentemente utilizzata nei moderni sistemi di conversione energetica.

Il progetto mira a sviluppare e validare un modulatore basato su tecnologia FPGA che implementi sia la modulazione Pulse Width Modulation (PWM) che la Space Vector Modulation (SVM). Utilizzando dispositivi tradizionali come MOSFET o IGBT, il lavoro si concentra sulla progettazione di un sistema modulare per facilitarne la riusabilità del codice in diverse applicazioni.

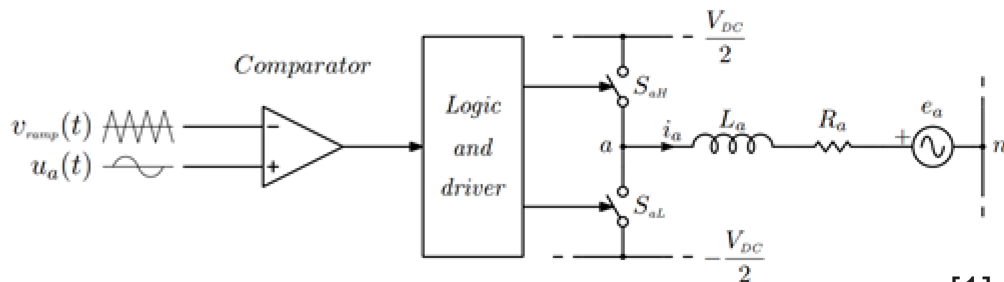


[1].

Per verificare l'efficacia del modulatore, è stato inoltre sviluppato un codice di test in VHDL, che emula un filtro passa basso per analizzare la qualità delle forme d'onda generate in uscita.

COMPITO	RESPONSABILE
PWM	
FSM, sample_and_hold, switch_generator	Stefano Lunardelli
input_lut_mux, lut_sine, out_lut_reg	Alessandro Rossi
phase_ampl_gen, comparator, filtro_sim (processo cicli fasi)	Francesco Milani
cnt_carrier_wave, filtro_sim (processo filtro), constraints, libreria: my_package	Filippo Perazzolo
SVM	
switch_gen, lut_sin	Stefano Lunardelli
generation_controll, num_sector_samp, ottimizzazione	Alessandro Rossi
phase_ampl_gen, sector_ident, vector_gen_time	Francesco Milani
vector_piloting, switch_piloting, constraints	Filippo Perazzolo

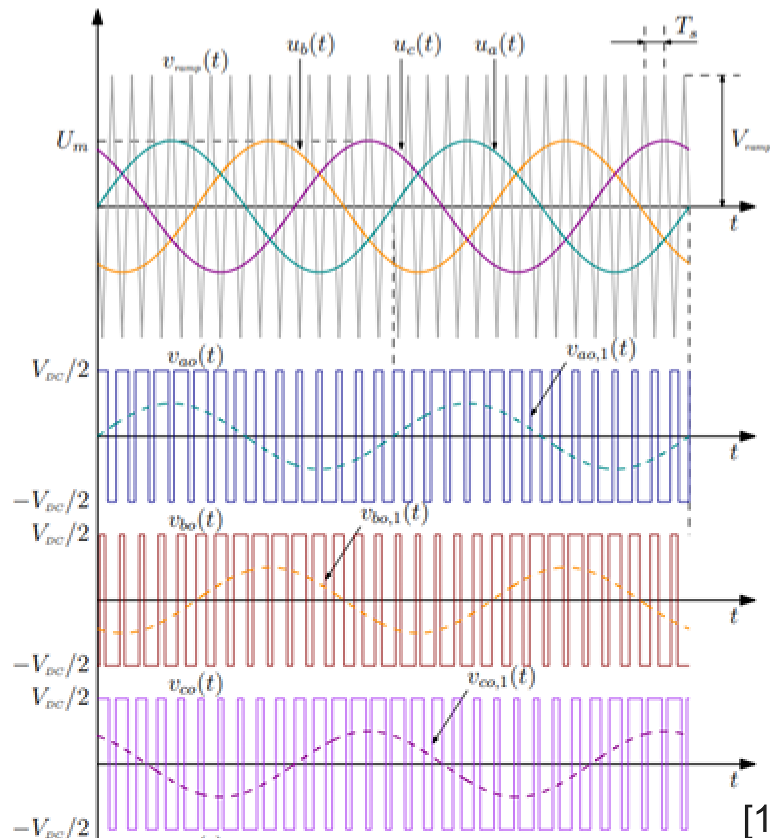
PWM MODULATION



[1].

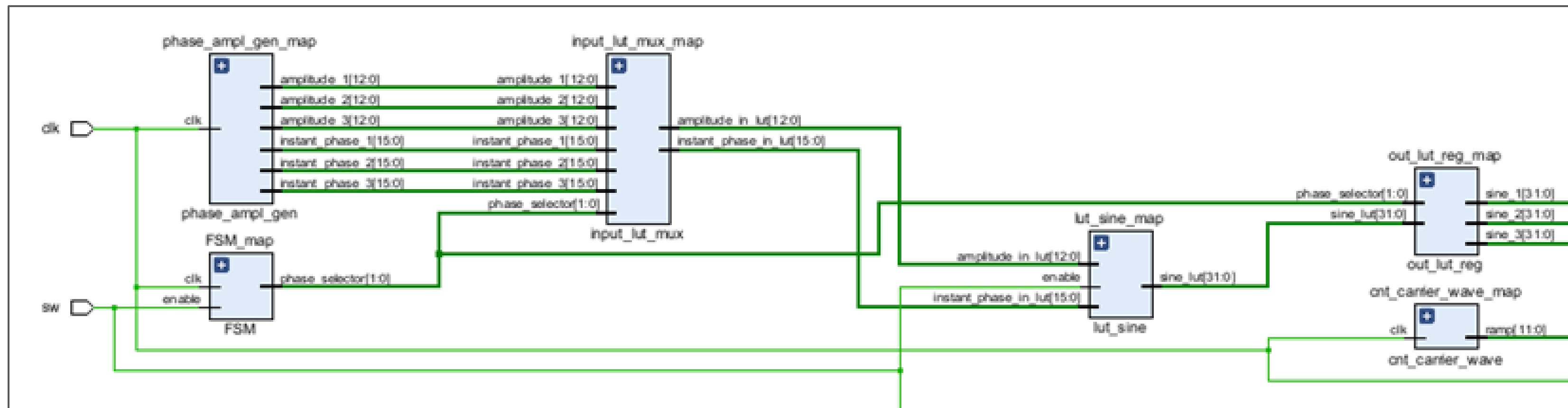
La modulazione di larghezza di impulso (PWM) è una tecnica fondamentale per il controllo dei convertitori di potenza AC-DC trifase. In questo progetto, si utilizza la modulazione PWM unipolare con una frequenza di commutazione di 100 MHz per regolare la tensione in uscita del convertitore.

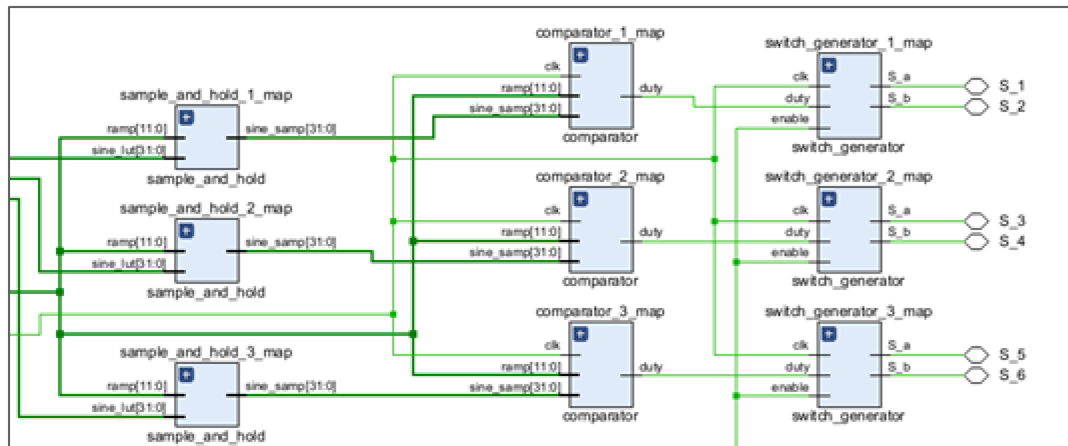
Variando il duty cycle dell'onda rettangolare, si controlla l'energia trasferita ai carichi, ottimizzando l'efficienza energetica del convertitore. L'implementazione su FPGA tramite moduli VHDL permette una gestione veloce delle commutazioni degli interruttori, che tipicamente sono MOSFET o IGBT.



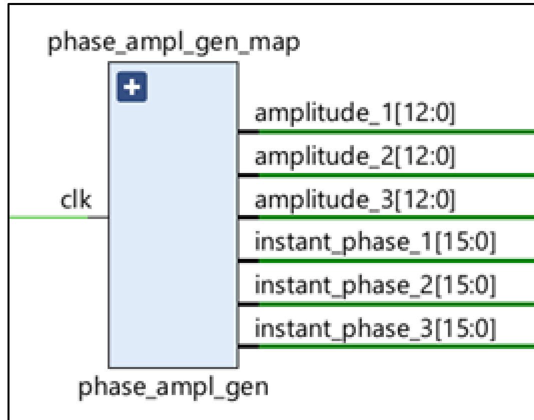
[1].

Schematic





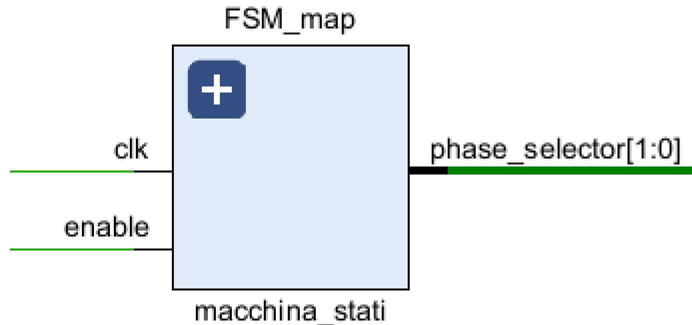
Generatore di fasi e ampiezze




```
phase_1_gen : process(clk)
            begin
            if rising_edge(clk) then
                if (count = update) then
                    if (phase_1 >= two_pi_scaled) then
                        phase_1 <= 0;
                    else
                        phase_1 <= (phase_1 + increase);
                    end if;
                end if;
                instant_phase_1 <= phase_1;
            end if;
            end process;
```

- Questo componente aggiorna continuamente tre segnali di fase all'interno di un intervallo (update), simulando la natura ciclica delle onde sinusoidali, fornendo inoltre in uscita ampiezze costanti.

Macchina a stati finiti



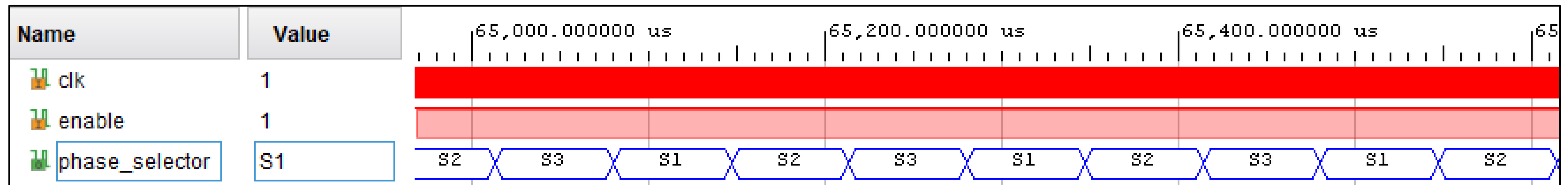
- La macchina a stati finiti (FSM) è un componente che ciclicamente cambia stato per selezionare quale delle tre fasi elaborare, permettendo di calcolare una sinusoide alla volta in intervalli di tempo brevi.

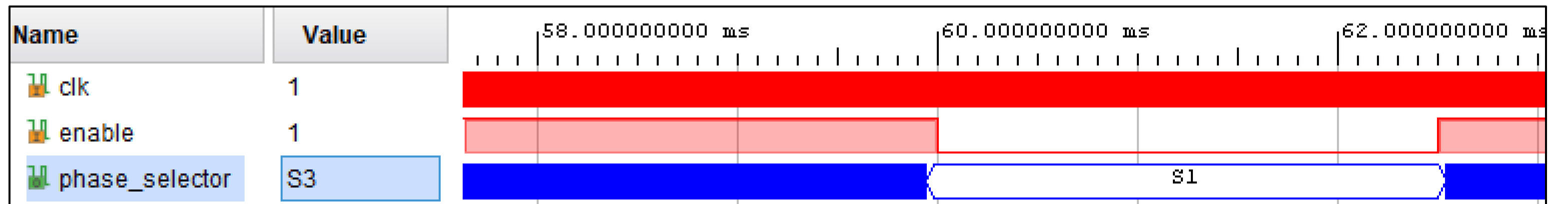
```
NextStateProc : process ( CurrentState , cnt )
    begin
        case CurrentState is
        when S1 =>
            if(cnt=N_cycles) then
                NextState <= S2;
            else
                NextState <= S1;
                end if;
```

```
UpdateStateProc : process ( clk , enable )
     begin
         if (enable = '0') then
            CurrentState <= S1;
          elsif (rising_edge(clk)) then
            CurrentState <= NextState;
            phase_selector <= CurrentState;
     end if ;
```

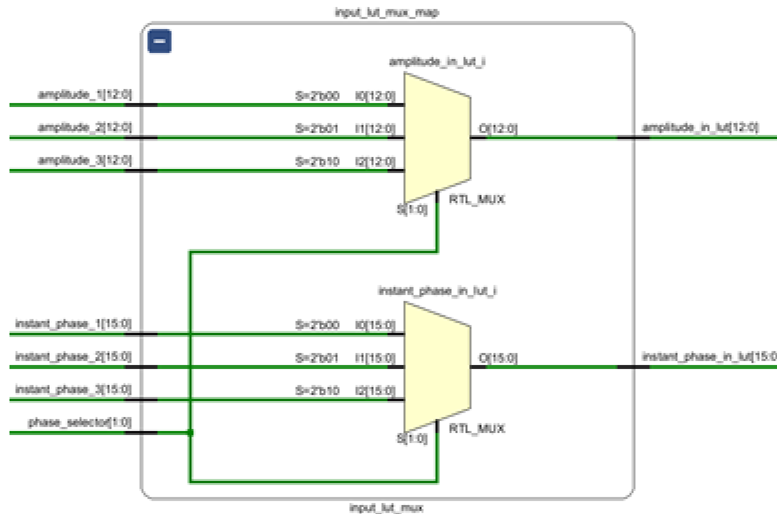
Utilizzo di un contatore per determinare il passaggio tra gli stati. Al raggiungimento di un valore predefinito (N_cycles), la FSM transita al prossimo stato.

Macchina a stati finiti





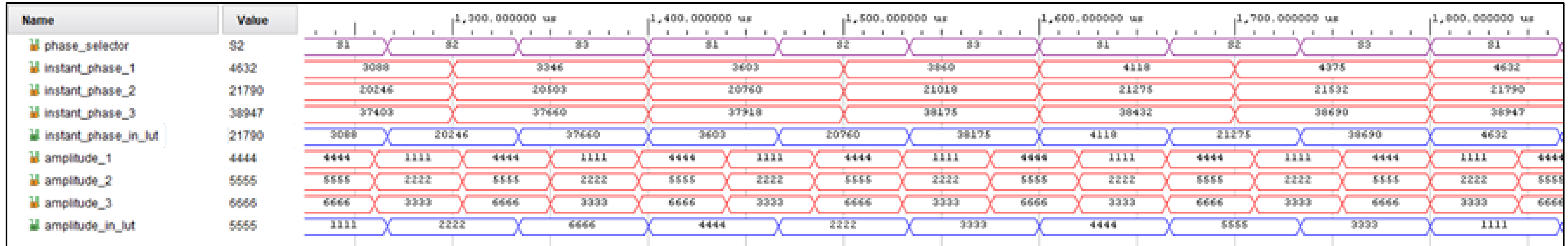
Multiplexer dei segnali alla LUT

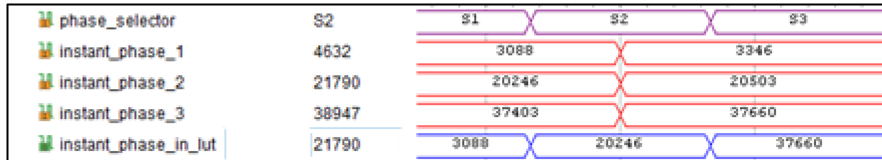


- Questo blocco si occupa di dare in ingresso alla LUT che genera la modulante l'ampiezza e la fase di una delle tre sinusoidi che si vogliono generare. Si fa ciò con due multiplexer pilotati dal segnale `phase_selector`. Questo permette di implementare del time sharing.

Multiplexer dei segnali alla LUT

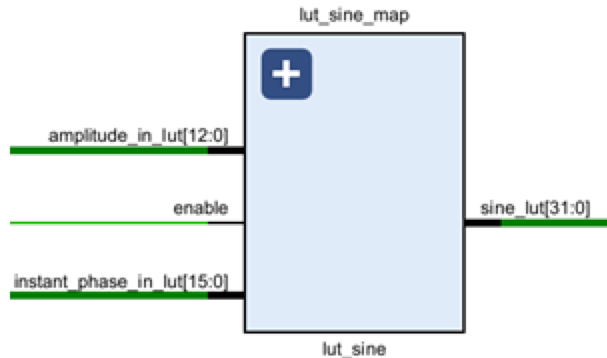
- Simulazioni:





- Si nota che i segnali di uscita (blu) sono quelli corrispondenti alla sinusoide “selezionata” da `phase_selector` in quel momento

LUT per la generazione della modulante



- Questo blocco si occupa di generare la modulante. Nello specifico il componente è composto da una LUT ed effettua alcuni calcoli per dare in uscita un valore del seno corrispondente agli ingressi (fase ed ampiezza)

- Nello specifico, dato il comportamento del MUX (il quale fornisce gli ingressi), in uscita si hanno in successione i valori di tutte e tre le sinusoidi

LUT per la generazione della modulante

```
i_var := ((num_elem_lut-1)*instant_phase_in_lut)/PHASE_MAX_IN;

amplitude_ratio_var := 128*amplitude_in_lut;

scaled_sine_var := TO_SIGNED(amplitude_ratio_var*sine(i_var), scaled_sine_var'length);

offset := shift_left(TO_UNSIGNED(2, offset'length), NUM_BITS-2);

sine_lut <= std_logic_vector( shift_right(scaled_sine_var, scale_factor) + signed(offset) );
```

- Con questa parte di codice si va a calcolare il valore della modulante andando a prelevare il valore contenuto nella LUT corrispondente alla fase in ingresso (calcolando l'indice `i`), per poi "adattare" questo valore all'ampiezza in ingresso.

- Calcoli svolti:

$$\text{scaled_sine} = \frac{A_{in}}{A_{lut}} \sin(i) \cdot 2^{19} = \frac{A_{in}}{2^{12}} \sin(i) \cdot 2^{19} = 2^{7} A_{in} \sin(i)$$

$$\text{offset} = 2 >> NUM_BIT - 2 = 2^{NUM_BIT-1} = \frac{2^{NUM_BIT}}{2}$$

$$\text{sine_lut} = (\text{scaled_sine} >> 19) + \text{offset}$$

- Scelta fattore di scala **`scale_factor`**:

$$\left(\frac{A_{in}}{A_{LUT}}\right)_{max} sin(i)_{max} = 1 \cdot \frac{2^{12}}{2} = 2^{11}$$

  - Dunque per rappresentare questo numero occorrono 11 bit + 1 di segno, dato che si sta lavorando con interi si hanno a disposizione $32 - 12 = 19$ bit

LUT per la generazione della modulante

- Struttura LUT:

```
type memory_type is array (0 to num_elem_lut-1) of integer;
signal sine : memory_type :=(0, 64, 129, 193, 257, 320, 384, . . . .
                             . . . . . . . . . . . . . . . . . . . .
                             . . . . . . . . . . . . . . . . . . . .
                             . . . . . . , -447, -257, -193, -129, -6);
```

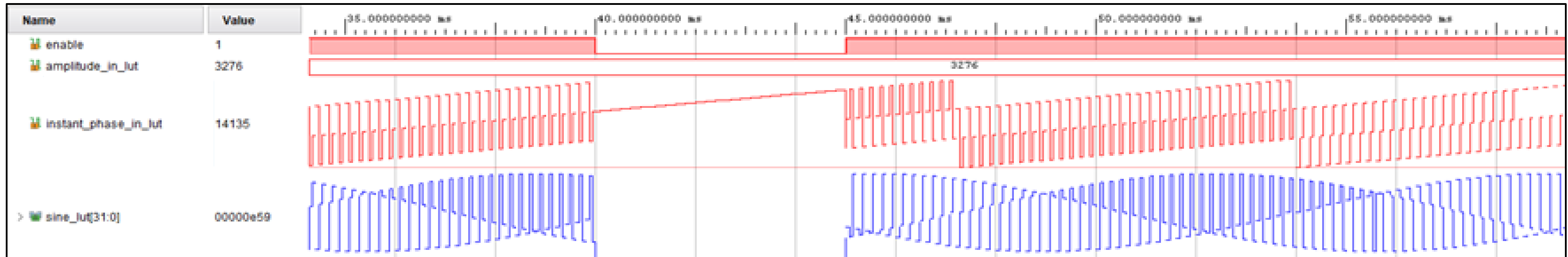
- Scelta numero valori nella LUT:

$$\frac{|sin(\omega t_1) - sin(\omega t_2)|}{sin(\omega t_1)} \approx \frac{|\omega t_1 - \omega t_2|}{\omega t_1} = 0,5\%$$

$$t = t_1 - t_2 = 0,5\% \cdot t_1 = 1 \cdot 10^{-4} s \qquad \rightarrow t_1 = 20ms$$

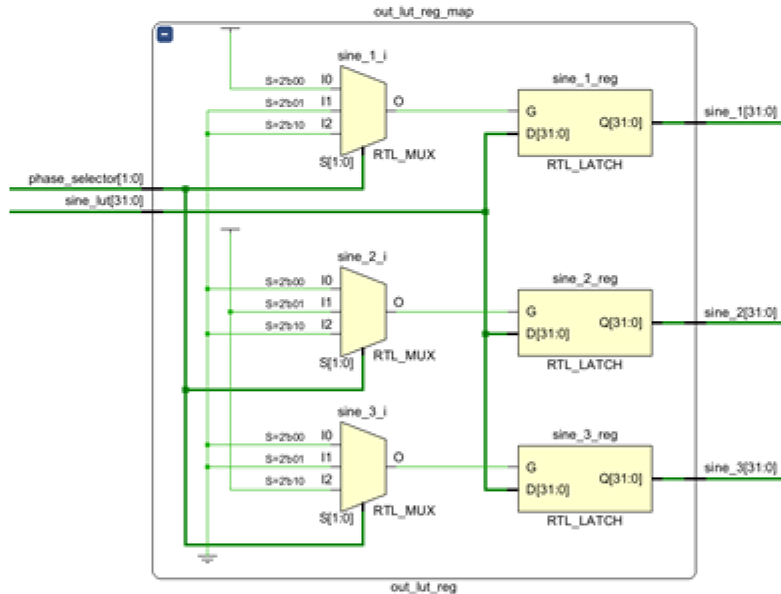
$$\Longrightarrow num_elem_lut = \frac{T_{sin}}{t} = 200$$

- Simulazione:



- Si vede che si generano i valori di tutte e tre le sinusoidi in successione, inoltre si nota bene che anche la fase in ingresso assume, ciclicamente, il valore associato ad una delle tre sinusoidi

Registro in uscita alla LUT

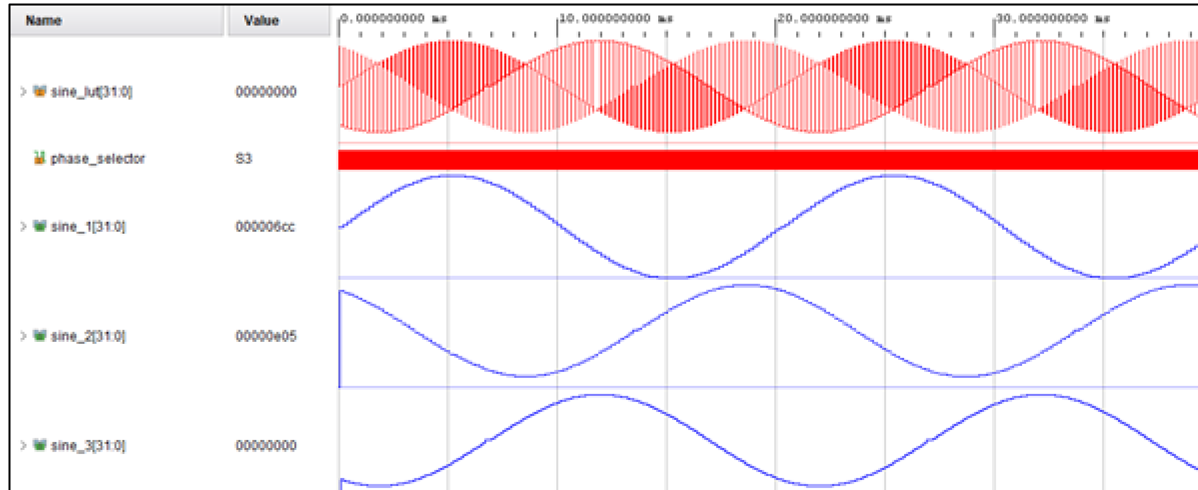


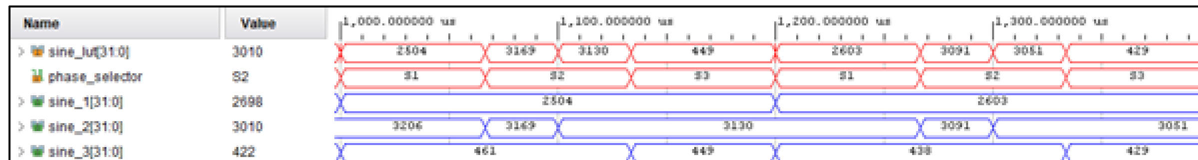
- Questo blocco restituisce in uscita le tre sinusoidi desiderate accettando in ingresso ciò che viene calcolato dalla LUT.

- Per fare ciò si salva in dei registri il valore di una delle tre fasi quando il segnale `phase_selector` assume il valore associato ad una di esse

Registro in uscita alla LUT

- Simulazioni:

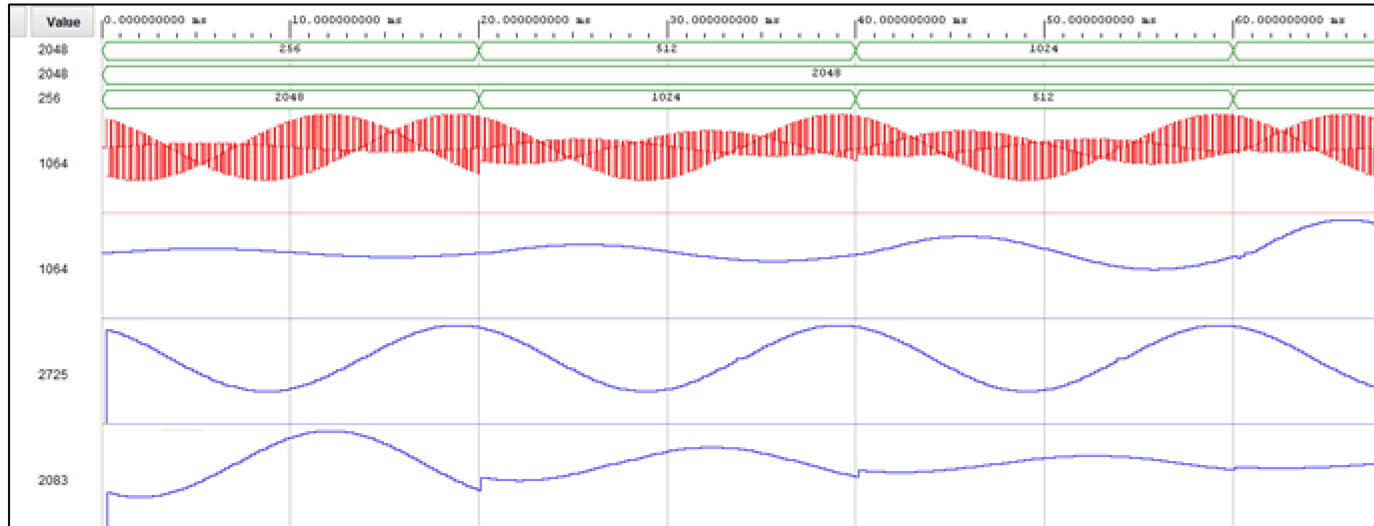




- Si nota come dal segnale `sine_lut` si ottengono le tre sinusoidi desiderate

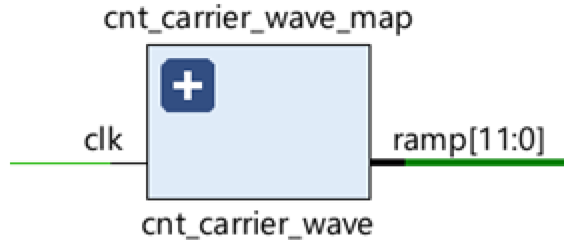
Registro in uscita alla LUT

- Simulazioni:



- Qui si è voluto far notare come si possano generare tre sinusoidi di ampiezza diversa e non costante

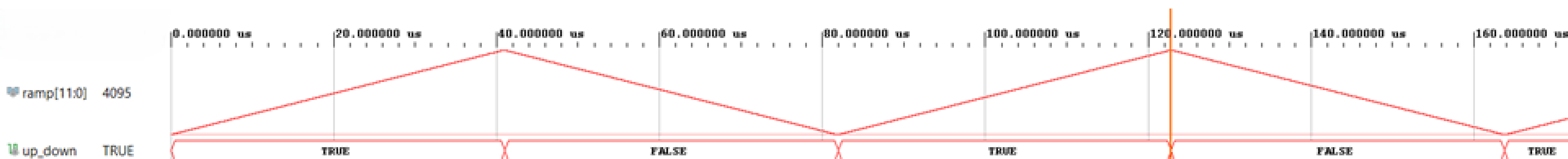
Contatore up-down carrier wave



- All'ingresso del blocco c'è il segnale di clock (`clk`), mentre in uscita si genera una rampa che funge da segnale per la portante.
- Il contatore up-down genera un segnale tringolare, con un valore massimo di conteggio pari a 4095.

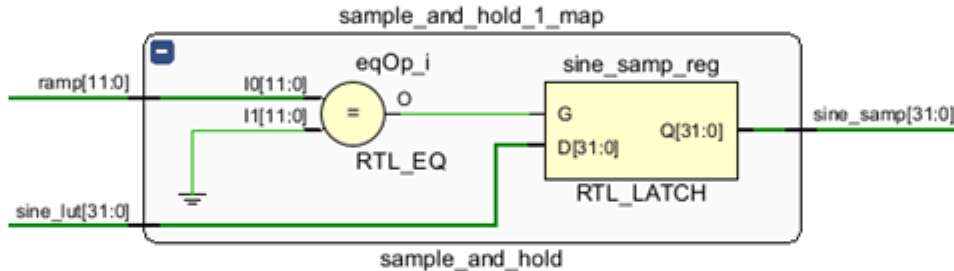
```
process(clk)
begin
    if rising_edge(clk) then
        if up_down then
            if count = MAX_COUNT then
                up_down <= false;
                count <= count - 1;
            else
                count <= count + 1;
            end if;
        else
            if count = (NUM_BITS-1 downto 0 => '0') then
                up_down <= true;
                count <= count + 1;
            else
                count <= count - 1;
            end if;
        end if;
    end if;
end process;

ramp <= count;
```

Contatore up-down carrier wave



- Il grafico mostra l'andamento del contatore up-down utilizzato come carrier wave. Il segnale `ramp` varia tra 0 e 4095, generando un'onda triangolare con un periodo di circa 81.91 µs, quindi una frequenza di 12209 Hz, 244 volte superiore rispetto alla frequenza della modulante. Il segnale up_down indica la direzione del conteggio: TRUE per la fase di incremento e FALSE per la fase di decremento.

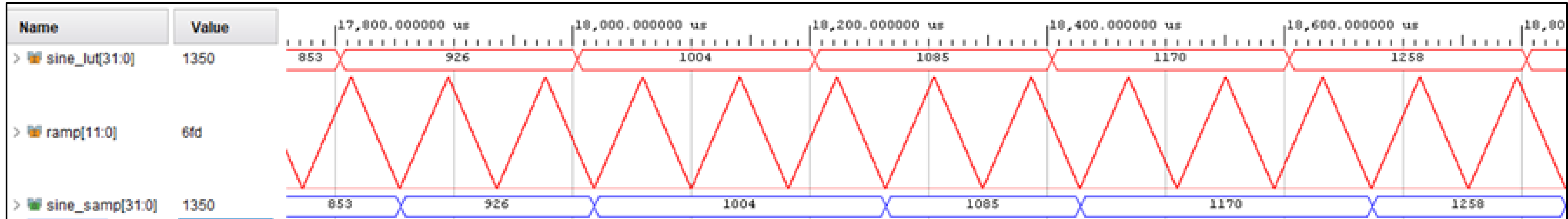
Sample and Hold




```
        begin
            sample_hold : process(ramp)
                begin
                                if ramp = (NUM_BITS-1 downto 0 => '0')
then
                sine_samp <= sine_lut;
                                end if;
    end process;
```

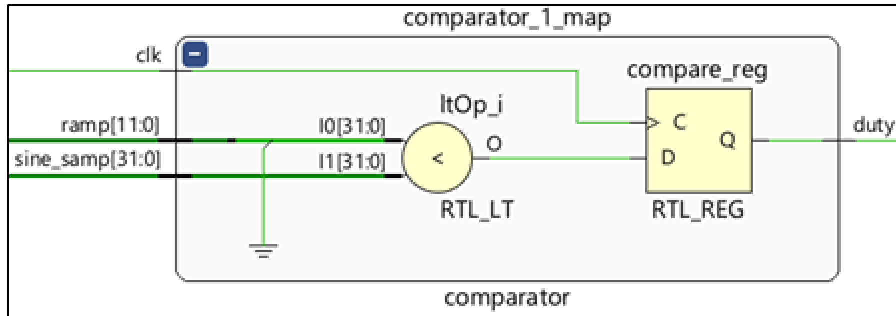
- Il blocco "Sample and Hold" cattura e mantiene il valore della sinusoide generata (sine_lut) quando il segnale di rampa (ramp) raggiunge zero. Questo permette di sincronizzare il campionamento della sinusoide con la frequenza della portante, garantendo un'uscita stabile e precisa.

Sample and Hold

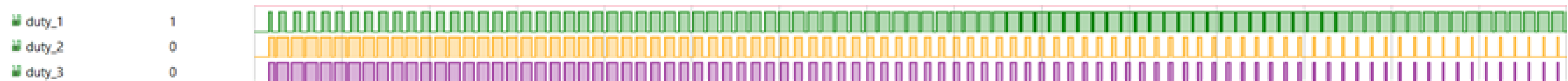


- L'immagine mostra il comportamento temporale del blocco "Sample and Hold". La sinusoide generata (sine_lut) è campionata ogni volta che la rampa (ramp) raggiunge zero. Questo garantisce che l'uscita campionata (sine_samp) corrisponda al valore esatto della sinusoide in un punto specifico e ripetibile del ciclo. La sinusoide campionata (sine_samp) viene quindi utilizzata nel comparatore per generare il segnale di duty del PWM.

Comparatore

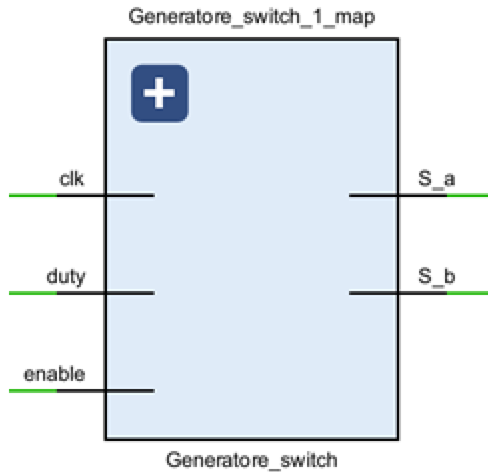


- Lo scopo di questo blocco è generare il duty cycle, determinando i punti di commutazione degli switch.



- In questo grafico si può apprezzare la differenza tra i duty cycle delle 3 fasi

Generatore segnali switches

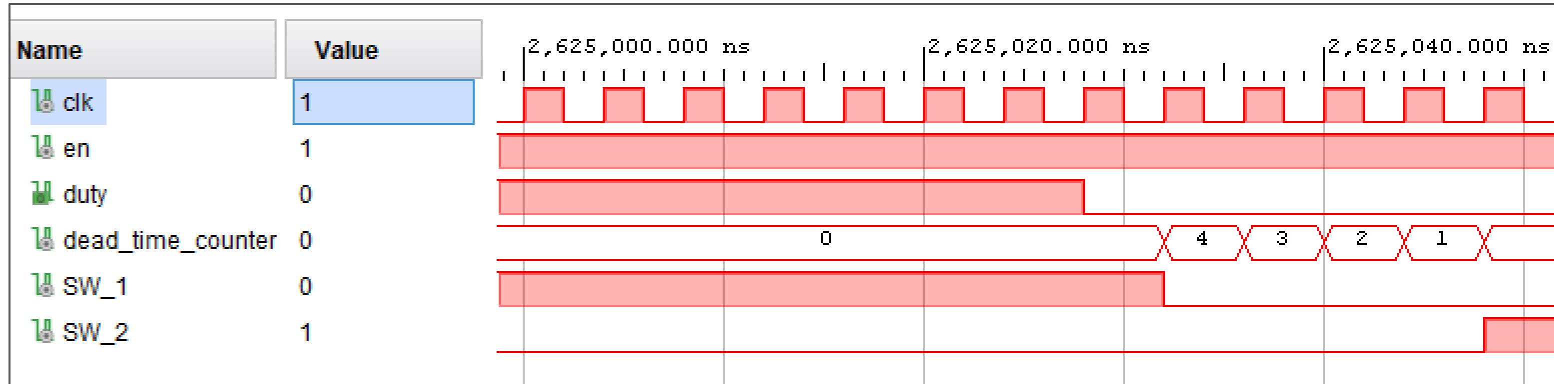



```
dead_time_counter <= dead_time_value;
last_duty <= duty;

if dead_time_counter > 0 then
dead_time_counter <= dead_time_counter - 1;
```

```
if dead_time_counter = 1 and duty = '0' then
        S_b <= '1';
end if;
if dead_time_counter = 1 and duty = '1' then
        S_a <= '1';
end if;
```

- Il blocco gestisce i segnali di commutazione in base al duty cycle, includendo una gestione del tempo morto per prevenire la sovrapposizione dei segnali di comando. Quando il duty cycle cambia, il blocco spegne immediatamente uno dei segnali di uscita e avvia un conteggio del tempo morto. Al termine del tempo morto, il segnale appropriato viene acceso, prevenendo il cortocircuito.

Generatore segnali switches



- Il diagramma temporale illustra come i segnali di uscita (S_a, S_b) vengono gestiti in risposta al cambiamento del duty cycle e durante il tempo morto (impostato a 5 cicli di clock). La gestione del tempo morto assicura che i segnali S_a e S_b non si sovrappongano mai, prevenendo così il cortocircuito.

Libreria my_package

```
package my_package is

    constant NUM_BITS : integer := 12;
    type state is (S1, S2, S3);
    constant PHASE_MAX_IN : integer := 51472;                    -- 2pi*2^13; 13 = scale factor
    constant AMPLITUDE_MAX_IN : integer := 4096;                 -- 2^12, equivale al valore massimo della portante con frequenza di
commutazione minima
    constant SINE_BIT : integer := 31;
    ---- phase_ampl_gen -----
    constant increase : integer := 257;                          -- incremento fase (2pi/200, 200 = valori lut seno)
    constant two_pi_scaled : integer := 51472;                   -- valore costante di due volte pi scalato (2 * pi)*2^13
    constant two_thirds_pi_scaled : integer := 17157;            -- valore costante di due volte pi diviso tre scalato ((2 * pi) / 3)*2^13
    constant four_thirds_pi_scaled : integer := 34315;           -- valore costante di quattro volte pi diviso tre scalato ((4 * pi) /
3)*2^13
    constant update : integer := 10000;                          -- numero di cicli di clock corrispondenti a 100 us,
                                                                 -- tempo dopo il quale si incrementa il valore della fase
    ----- lut_sine -----
    constant scale_factor : integer := 19;                       -- fattore di scala utilizzato per scalare il seno all'ampiezza desiderata
    constant num_elem_lut : integer := 200;                      -- numero elementi contenuti nella lut
    ----- cnt_carrier_wave -----
    constant MAX_COUNT : STD_LOGIC_VECTOR(NUM_BITS-1 downto 0) := (NUM_BITS-1 downto 0 => '1');
    ----- FSM -----
    constant N_cycles : integer := 6667;                         -- Cicli di clk dopo i quali cambiare stato
    ----- switch_generator -----
    constant dead_time_value : integer := 4;

end package my_package;
```

File di constraints: Basys3_Master.xdc

```
set_property PACKAGE_PIN V17 [get_ports {sw}]

set_property IOSTANDARD LVCMOS33 [get_ports {sw}]
```

```
set_property PACKAGE_PIN W5 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports clk]

create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
```

- **LVCMOS** (Low-Voltage Complementary Metal-Oxide-Semiconductor), utilizza una tensione di 3.3V per il segnale alto ‘1’ mentre 0V per il segnale basso ‘0’. Utilizza un singolo cavo per la trasmissione, il che porta vantaggi ma anche degli svantaggi.
- **create_clock**, definisce il segnale di clock specificando le sue proprietà di temporizzazione.
- **set_property**, viene utilizzato per impostare una proprietà specifica su un oggetto di design nel progetto FPGA.

```
set_property PACKAGE_PIN J1 [get_ports {S_1}]

set_property IOSTANDARD LVCMOS33 [get_ports {S_1}]

set_property PACKAGE_PIN L2 [get_ports {S_2}]

set_property IOSTANDARD LVCMOS33 [get_ports {S_2}]

set_property PACKAGE_PIN J2 [get_ports {S_3}]

set_property IOSTANDARD LVCMOS33 [get_ports {S_3}]

set_property PACKAGE_PIN G2 [get_ports {S_4}]

set_property IOSTANDARD LVCMOS33 [get_ports {S_4}]

set_property PACKAGE_PIN H1 [get_ports {S_5}]

set_property IOSTANDARD LVCMOS33 [get_ports {S_5}]

set_property PACKAGE_PIN K2 [get_ports {S_6}]

set_property IOSTANDARD LVCMOS33 [get_ports {S_6}]
```

Filtro in simulazione

```
switch_selection : process(clk)
    begin

        if rising_edge(clk) then
            if timer < TIMER_THRESHOLD  then
                timer <= timer+1;
                case SW_active is
                   when 1 =>
                    SW <= SW_1;
                   when 2 =>
                    SW <= SW_3;
                   when 3 =>
                    SW <= SW_5;
                   when others =>
                    SW <= SW;
                end case;
            else
```

- Processo che alterna periodicamente gli switch dei segnali da filtrare

```
case SW_active is
                    when 1 =>
                        SW <= SW_3;
                        SW_active <= 2;
                        timer <= 0;
                        TIMER_THRESHOLD <= 4500000;
                    when 2 =>
                        SW <= SW_5;
                        SW_active <= 3;
                        timer <= 0;
                        TIMER_THRESHOLD <= 4500000;
                    when 3 =>
                        SW <= SW_1;
                        SW_active <= 1;
                        timer <= 0;
                        TIMER_THRESHOLD <= 4500000;
                    when others =>
                        timer <= 0;
                        SW_active <= 1;
                        SW <= SW;
                end case;
            end if;
        end if;
    end process;
```

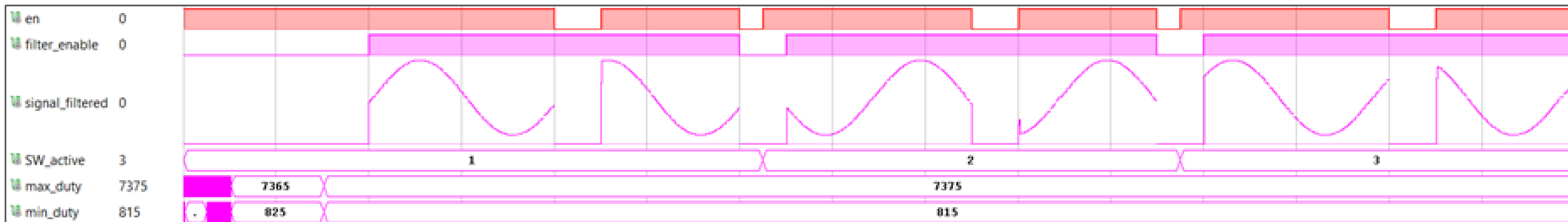
Filtro in simulazione

```
filtro_1 : process(clk, en)
    begin
        if en = '0' then
            pwm_state <= '0';
            signal_filtered <= 0;
            pwm_high_count <= 1;

        elsif rising_edge(clk) then

            clock_count <= clock_count + 1;


            if pwm_state /= SW then

                pwm_state <= SW;

                if pwm_state = '1' then
                    pwm_period_state <= pwm_period_state + 1;

-- identifies two different periods of the PWM signal
                    if pwm_period_state > 0 then

-- system to avoid overflow
                        pwm_period_state <= 0;
                    end if;
                end if;

                pwm_high_count <= pwm_high_count + 1;


                if pwm_period_state > 0 and pwm_state /= 'U' then

                    if clock_count > max_duty then
                        max_duty <= clock_count;
                    end if;
                    if clock_count < min_duty then
                        min_duty <= clock_count;
                    end if;
```

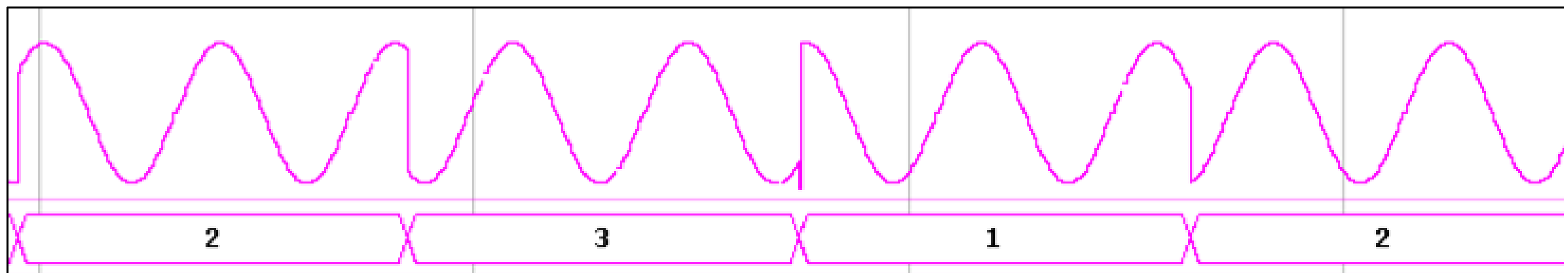
```
-- converts the duty cycle to an analog value

                if pwm_high_count = 2 then
                    if filter_enable = '1' then
                        clock_count <= clock_count - min_duty;
-- rearranges the obtained value based on the min_duty value obtained
                        signal_filtered <= (2**(NUM_BITS) * clock_count) / max_duty;
-- calculates the value of the demodulated signal based on the range between max_duty and min_duty

                    else
                        signal_filtered <= 0;
                    end if;
                    pwm_high_count <= 1;
                end if;
                clock_count <= 1;
            end if;
        end if;

    end process;
```

- Il processo elabora il duty cycle di una fase alla volta, riarrangia il valore ottenuto in base ai valori minimi e massimi del duty cycle, e calcola il segnale demodulato risultante in funzione dell'intervallo tra `max_duty` e `min_duty`.

Filtro in simulazione

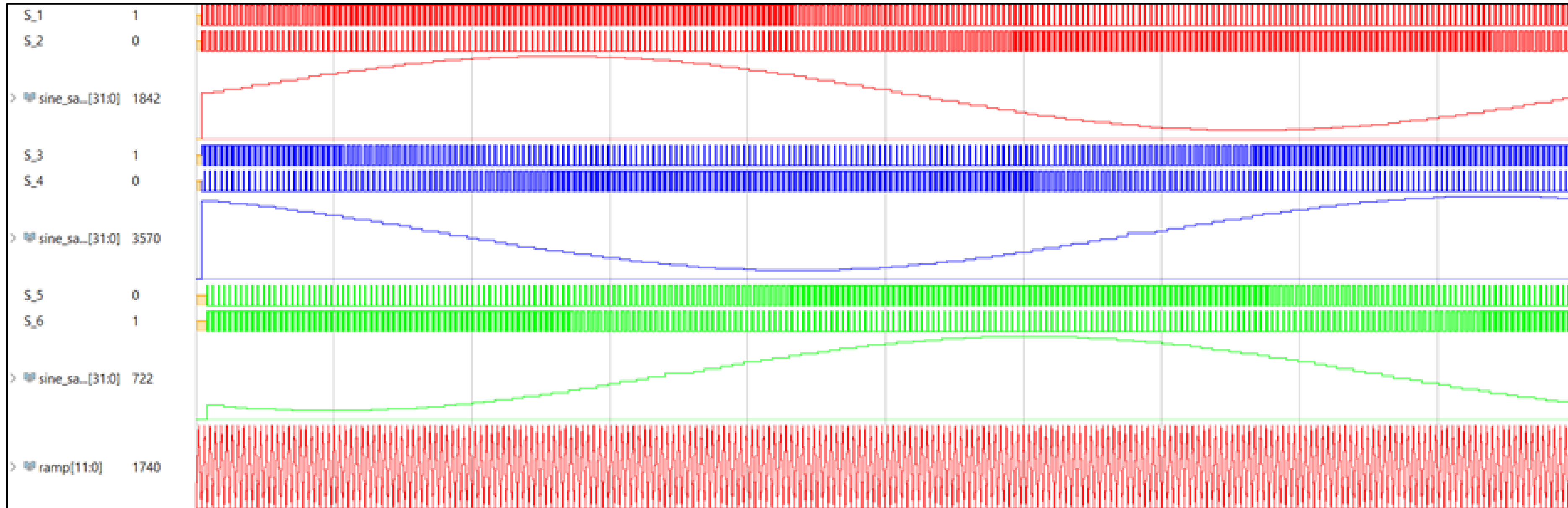


- funzionamento segnale di `filter_enable` ed enable (`en`) sulle tre fasi, con calcolo dei valori di `max_duty` e `min_duty` nei primi 20 ms della simulazione.



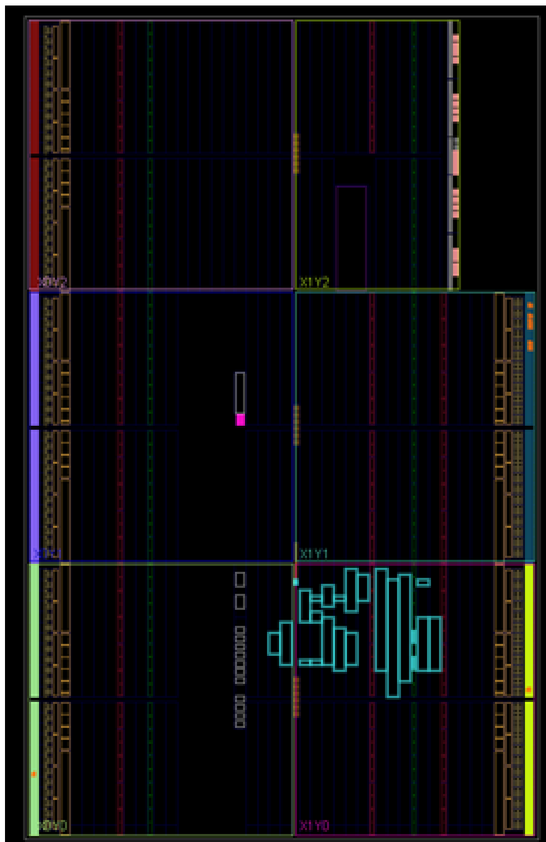
- simulazione del filtro per le singole fasi senza interruzioni

Simulazioni



- Il grafico rappresenta la simulazione delle tre sinusoidi che fungono da segnali modulanti, mentre il segnale a rampa rossa rappresenta la portante PWM. I segnali etichettati come S_1, S_2, S_3, S_4, S_5 e S_6 indicano i duty cycle degli switch dell'inverter.

Utilizzo

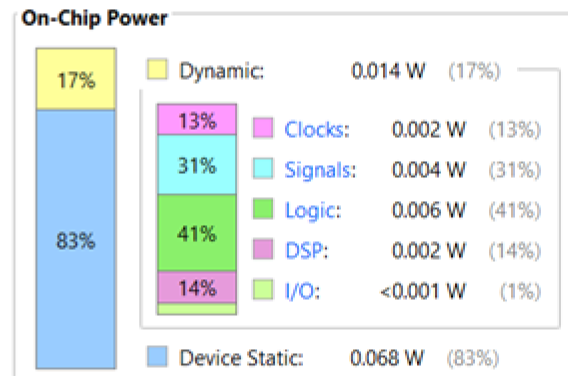


- L’immagine rappresenta il design implementato, mostra come il progetto è stato mappato sull'hardware dell'FPGA.

- Questa visualizzazione aiuta a comprendere la distribuzione e le interconnessioni degli elementi logici

Utilizzo

Resource	Utilization	Available	Utilization %
LUT	558	20800	2.68
FF	296	41600	0.71
DSP	2	90	2.22
IO	8	106	7.55
BUFG	4	32	12.50

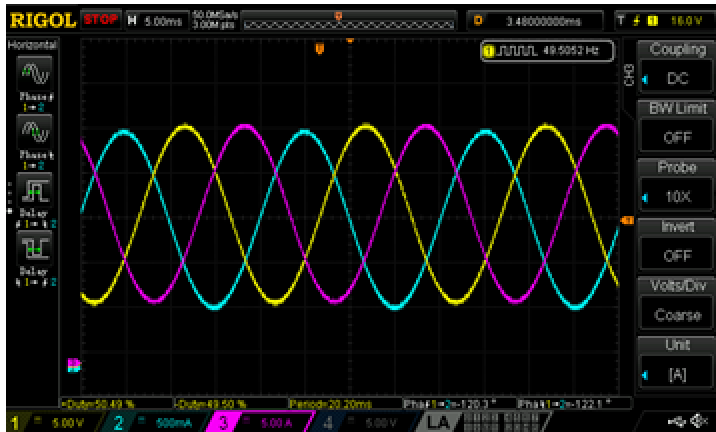


- Questa tabella mostra l'utilizzo delle risorse hardware di un FPGA, indicando quante unità di ciascuna risorsa sono utilizzate, quante sono disponibili e la percentuale di utilizzo.

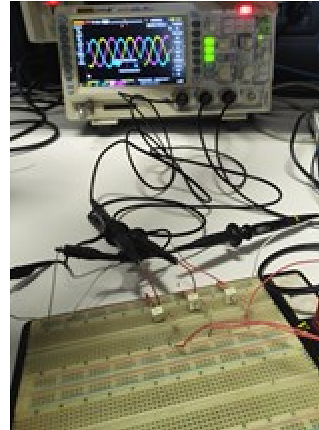
- Questo grafico illustra la distribuzione del consumo di potenza on-chip di un FPGA, suddivisa in potenza dinamica e statica.

Risultati di Laboratorio

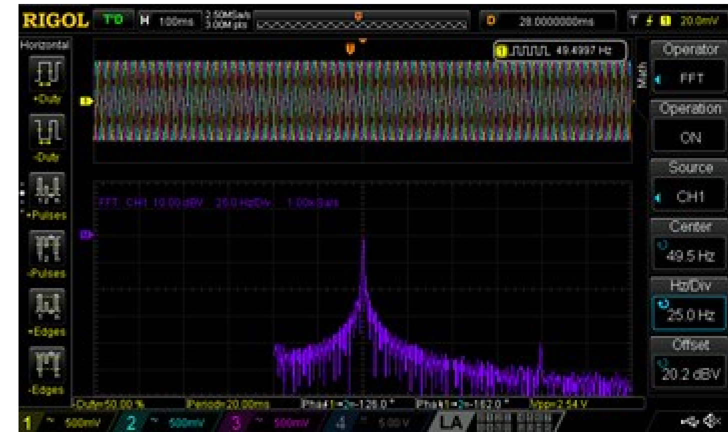
- Forme d'onda:



- Circuito RC:



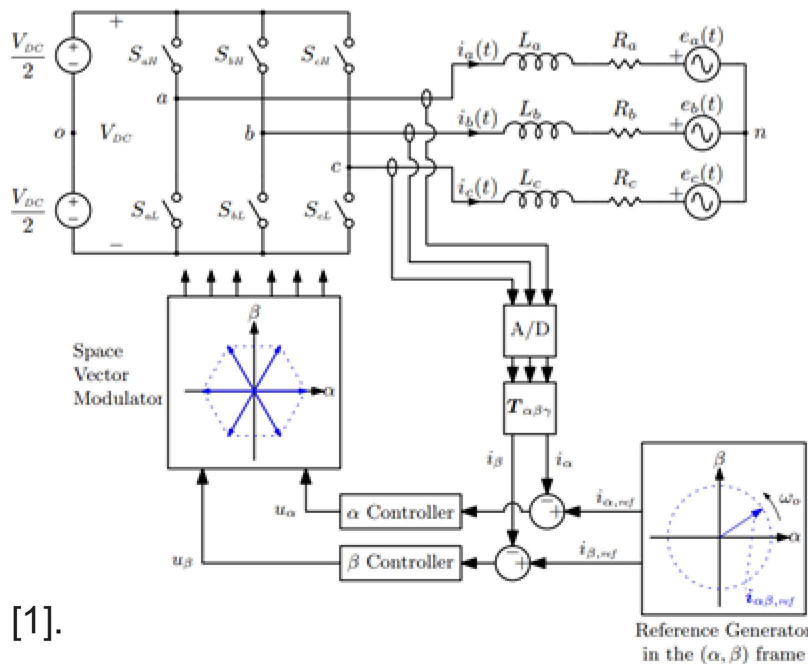
- Analisi in frequenza (FFT):



- Si riporta quanto ottenuto in laboratorio andando a filtrare i segnali di comando in uscita alla scheda con un opportuno filtro passa basso.
- Si nota che le forme d'onda generate, ottenute in laboratorio tramite l'ausilio di un oscilloscopio, hanno andamento corrispondente a quanto desiderato, ovvero una terna sinusoidale con componenti sfasate di 120 gradi e di periodo 20ms.
- Inoltre osservando il contenuto armonico si vede che si ha principalmente un picco centrato a 50Hz, ossia si ha effettivamente un periodo delle onde di 20ms.

SVM MODULATION

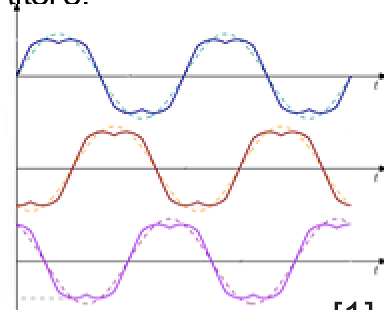
Questa parte del progetto mira a realizzare un modulatore per un convertitore DC-AC trifase utilizzando la Space Vector Modulation (SVM). La SVM è una tecnica di modulazione adatta ai sistemi digitali, in quanto consente un controllo più preciso e flessibile dei vettori di tensione applicati al motore o al carico.



[1].

Uno dei principali vantaggi della SVM è la sua capacità di generare una tensione di uscita con un contenuto armonico ridotto rispetto alla PWM tradizionale. Questo si traduce in una minore distorsione della forma d'onda di corrente, una migliore efficienza energetica e una riduzione delle perdite nel convertitore.

Inoltre, la SVM offre la possibilità di controllare direttamente l'ampiezza e la fase della tensione di uscita, consentendo una maggiore flessibilità nella gestione del sistema.



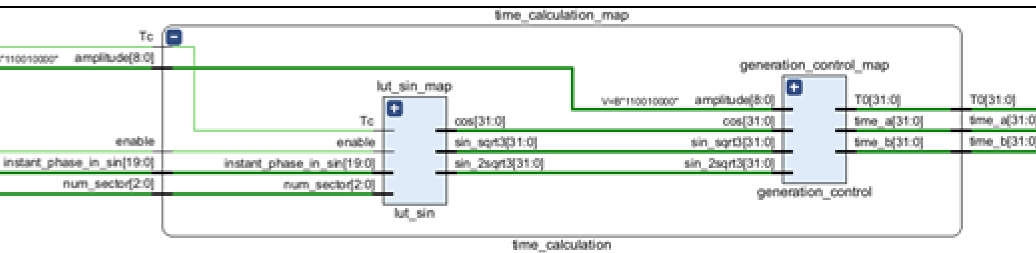
[1].

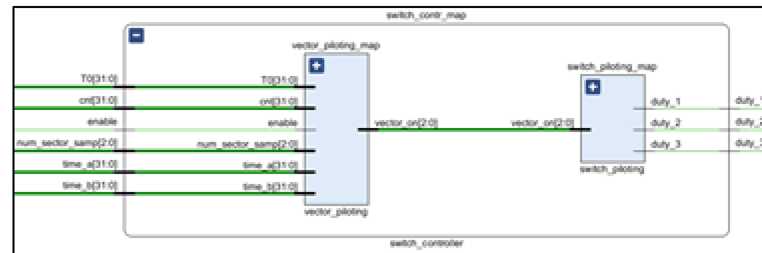
Università degli Studi di Padova

800 anni 1222-2022

Schematic

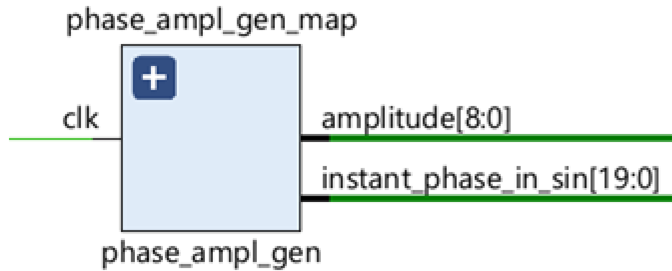
clk
sw
phase_ampl_gen_map
phase_ampl_gen
clk
amplitude[8:0]
instant_phase_in_sin[19:0]
vecotor_gen_time_map
vecotor_gen_time
enable
Tc
cnt[31:0]
sector_identifier_map
sector_identifier
num_sector[2:0]
time_calculation_map
time_calculation
T0[31:0]
time_a[31:0]
time_b[31:0]
num_sect_sampl_map
num_sector_sampler
num_sector_samp[2:0]
switch_contr_map
switch_controller
duty_1
duty_2
duty_3
switch_generation_1_map
switch_generation_2_map
switch_generation_3_map
switch_generation
duty
S_a
S_b
S_1
S_2
S_3
S_4
S_5
S_6





UNIVERSITÀ DEGLI STUDI DI PADOVA

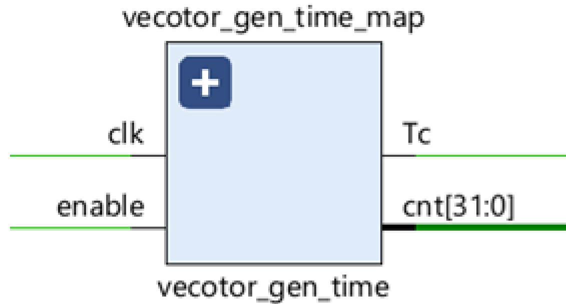
Generatore di fasi e ampiezze




```
phase_1_gen : process(clk)
        begin
            if rising_edge(clk) then
                if (count = update) then
                    if (phase_1 >= two_pi_scaled) then
                        phase_1 <= 0;
                    else
                        phase_1 <= (phase_1 + pi_scaled);
                    end if;
                instant_phase_in_sin <= phase_1;
                end if;
            end if;
        end process;
```

- Questo componente aggiorna continuamente il segnale di fase all'interno di un intervallo (update), simulando la natura ciclica delle onde sinusoidali, fornendo inoltre in uscita un' ampiezza costante.

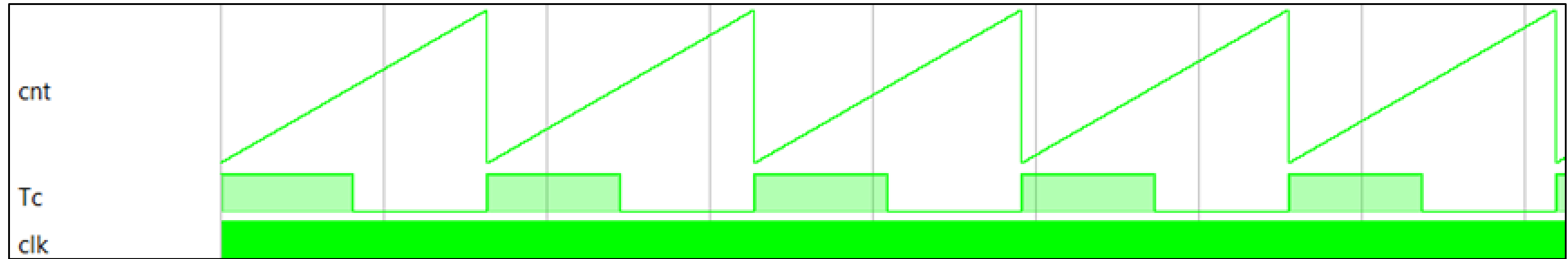
Generatore segnale “Tc”




```
cntProc : process(clk, enable)
        begin
            if (enable = '0') then
                cnt <= 0;
                Tc <= '1';
            elsif rising_edge(clk) then
                if cnt = Tc_count then
                    cnt <= 0;
                else
                    cnt <= cnt + 1;
                end if;

                if cnt = Tc_count/2 OR cnt = Tc_count then
                    Tc <= NOT Tc;
                end if;
            end if;
        end process;
```

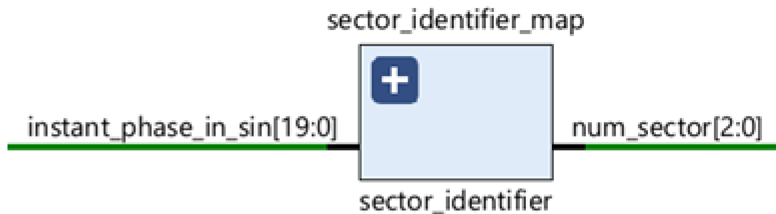
- Questo componente genera un clock secondario (Tc) necessario per sincronizzare i blocchi time_calculation e num_sect_sampler

Generatore segnale “Tc”



- In questa immagine si nota la differenza tra la durata di Tc, con relativo contatore, e il clock normale

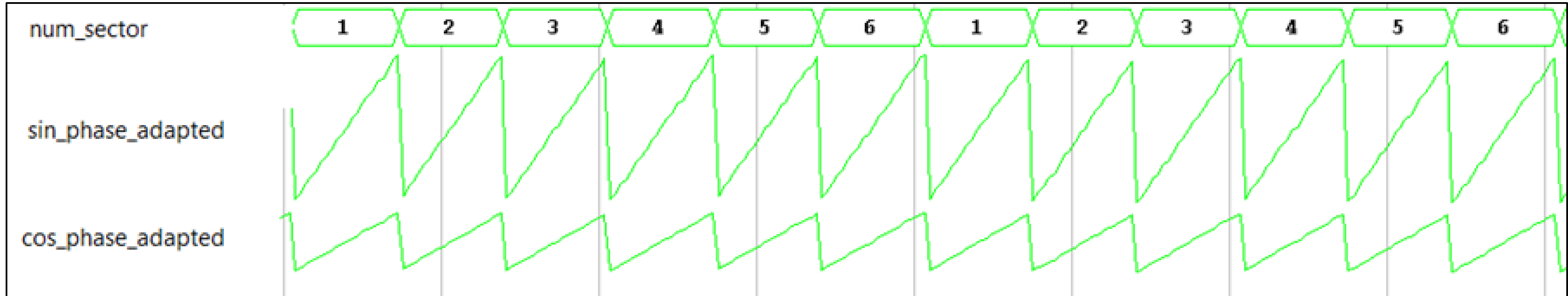
Identificazione del settore



- Questo componente determina il settore corrente della fase istantanea del segnale sinusoidale, suddividendo il ciclo in sei settori da $\pi/3$ radianti ciascuno.

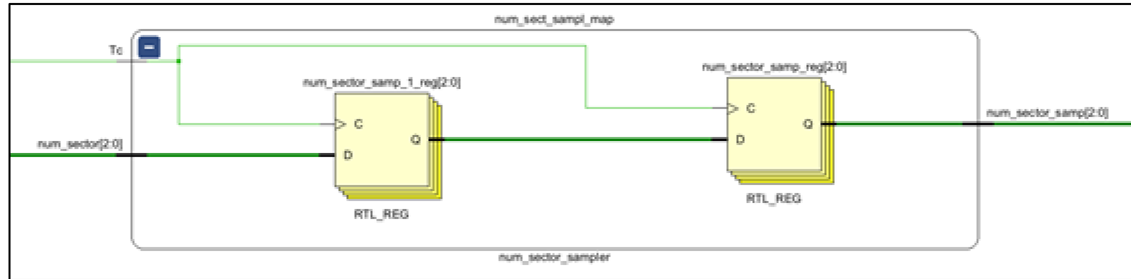
```
sector_ident : process(instant_phase_in_sin)
       begin
            -- primo settore
            if ( instant_phase_in_sin >= 0 and instant_phase_in_sin <= pi_over_three ) then
                    num_sector <= 1;
            end if;

            -- secondo settore
            if ( instant_phase_in_sin >  pi_over_three and instant_phase_in_sin <= 2*pi_over_three ) then
                    num_sector <= 2;
            end if;

            -- terzo settore
            if ( instant_phase_in_sin > 2*pi_over_three and instant_phase_in_sin <= 3*pi_over_three ) then
                    num_sector <= 3;
            end if;

            -- quarto settore
            if ( instant_phase_in_sin > 3*pi_over_three and instant_phase_in_sin <= 4*pi_over_three ) then
                    num_sector <= 4;
            end if;

            -- quinto settore
            if ( instant_phase_in_sin > 4*pi_over_three and instant_phase_in_sin <= 5*pi_over_three ) then
                    num_sector <= 5;
            end if;

            -- sesto settore
            if ( instant_phase_in_sin > 5*pi_over_three and instant_phase_in_sin <= 6*pi_over_three ) then
                    num_sector <= 6;
            end if;
        end process;
```

Identificazione del settore



- Questa immagine illustra come il blocco divide il ciclo in sei settori, visualizzando il numero di settore e le fasi adattate dei segnali sinusoidale e cosinusoidale per ciascun settore.

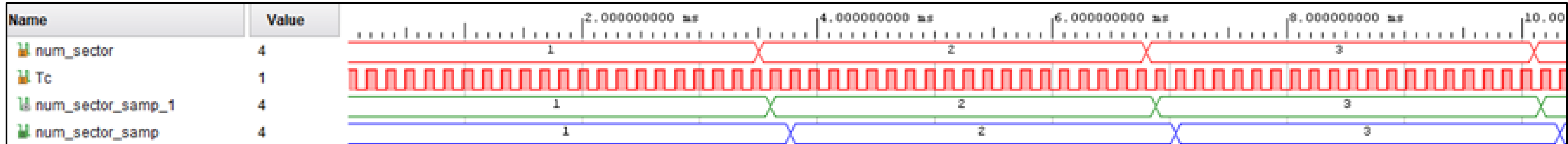
Sincronizzazione segnale “num_sector”

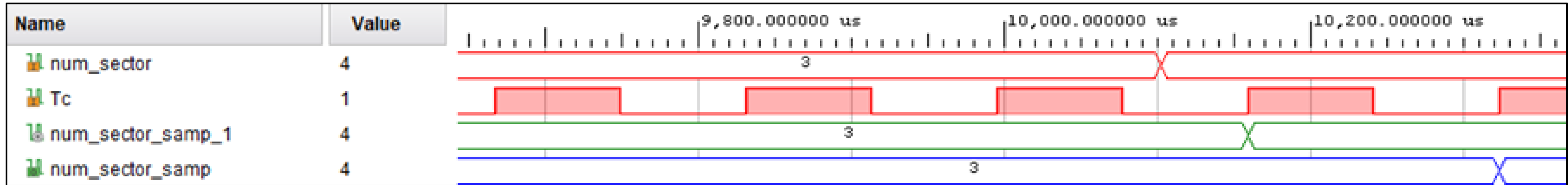


- Questo blocco si occupa di sincronizzare il segale `num_sector` con il segnale `Tc` (segnale a onda quadra simile ad un clock). Come si vede dalla figura Il blocco è composto da 2 FF delay. Il primo si occupa effettivamente di sincronizzare `num_sector` con `Tc`. Il secondo è stato inserito in quanto i blocchi che si occupano dei calcoli sono sincronizzati anch'essi con `Tc`, ciò provoca un ritardo intrinseco in questi dati. Per ovviare a ciò si ritarda di un ciclo di `Tc` il segnale `num_sector`, in modo da non creare problemi di sincronizzazione nei blocchi successivi.

Sincronizzazione segnale “num_sector”

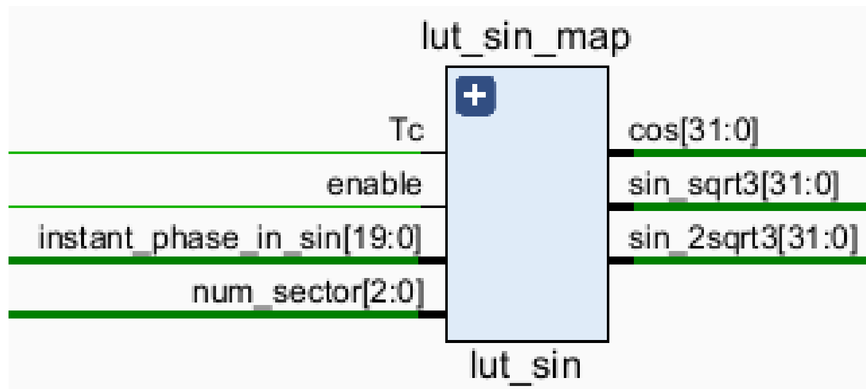
- Simulazioni:





- Si vede come `num_sector` venga prima sincronizzato con `Tc` (`num_sector_samp_1`) e successivamente ritardato di `Tc` (`num_sector_samp`)

LUT per la generazione di funzioni trigonometriche



- Scelta numero valori nella LUT:

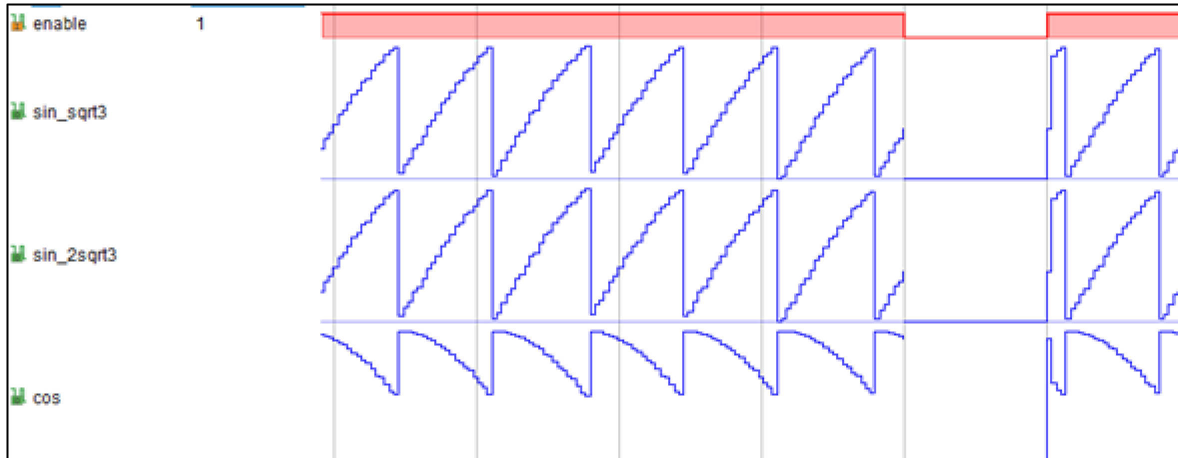
Sfruttando le proprietà trigonometriche è possibile utilizzare un'unica LUT che fornisca i valori per le funzioni seno e coseno. Inoltre sfruttando la periodicità delle funzioni e conoscendo il settore della fase in ingresso i valori sono compresi tra 0 e 5π/6, rispettivamente valore minimo del seno e massimo del coseno

- Struttura LUT:

```
type memory_type is array (0 to num_elem_lut-1) of integer;
signal sine_lut : memory_type :=(0,54,107,161,215,269,322,376, . . .
                                  . . . . . . . . . . . . . . . . . . .
                                  . . . . . . . . . . . . . . . . . . .
                                  . . . . . . , 4234,4188,4142,4095);
```

LUT per la generazione di funzioni trigonometriche

- Simulazioni:

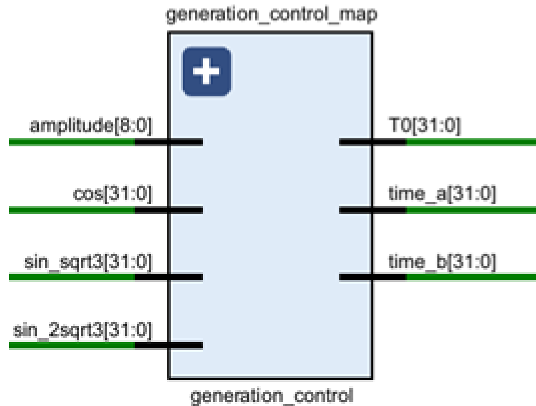


79,000.000000 us | 79,500.000000 us | 80

Name	Value							
Tc	0							
sin_sqrt3	1974024	1256224	1376400	1440928	1553704	1672400	1728936	1826320
sin_2sqrt3	3941379	2508204	2748150	2876988	3102159	3339150	3452031	3646470
cos	2443776	3592704	3443200	3379200	3209216	3044352	2948096	2745856

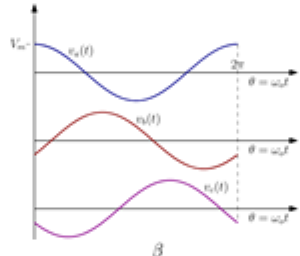
- Le simulazioni mostrano il comportamento dei segnali generati dalla LUT per funzioni trigonometriche. In particolare, osserviamo i segnali di seno e coseno per verificare la correttezza dell'implementazione della LUT.

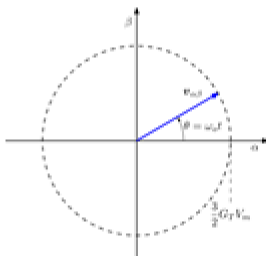
Calcolo tempi per la generazione dei vettori



- Questo blocco si occupa di calcolare i tempi per cui accendere i vari vettori ottenibili dalle combinazioni degli switches, in modo da generare mediamente un vettore specifico. Così facendo si riesce a creare la terna sinusoidale desiderata.

Calcolo tempi per la generazione dei vettori

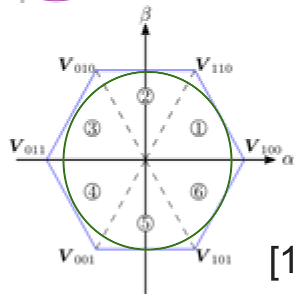




[1].

$$\begin{cases} e_1(t) = V_m \cos(\omega t) \\ e_2(t) = V_m \cos(\omega t - \frac{2}{3}\pi) \\ e_3(t) = V_m \cos(\omega t - \frac{4}{3}\pi) \end{cases} \overset{\alpha\beta\gamma}{\Rightarrow} \begin{cases} v_\alpha(t) = \frac{3}{2} G_T V_m \cos(\omega_o t) \\ v_\beta(t) = \frac{3}{2} G_T V_m \sin(\omega_o t) \\ v_\gamma(t) = 0 \end{cases}$$

$$\Rightarrow \|\vec{v}_{\alpha\beta}\| = \frac{3}{2} G_T V_m$$

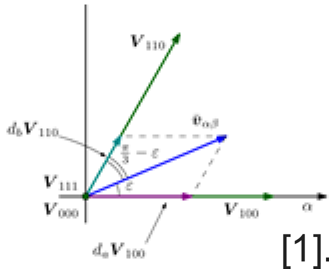


[1].

$$V_{m,max} = \frac{2}{\sqrt{3}} \frac{V_{DC}}{2} \approx 1.155 \frac{V_{DC}}{2} \quad \rightarrow$$

Ampiezza massima per generare una terna di sinusoidi

$$V_m = \sqrt{2} E_{rms} = \sqrt{2} \cdot 220V = 325V \Rightarrow V_m = 325V \approx 1.155 \frac{V_{DC}}{2} \Rightarrow V_{DC} = 565V$$



[1].

$$\begin{cases} d_a = \frac{\|\vec{v_{\alpha\beta}}\|}{V_{DC} G_T} \left(\cos(\varepsilon) - \frac{\sqrt{3}}{3} \sin(\varepsilon) \right) \\ d_b = \frac{\|\vec{v_{\alpha\beta}}\|}{V_{DC} G_T} \frac{2\sqrt{3}}{3} \sin(\varepsilon) \end{cases} \rightarrow$$

Questi valori sono duty cycles, ossia frazioni del tempo in cui si genera un vettore specifico

$$\rightarrow G_T = \sqrt{\frac{2}{3}}$$

Formule estratte da [1].

Calcolo tempi per la generazione dei vettori

```
vector_ampl := almpl_tranform*internal_ampl;
ratio_voltage := vector_ampl / max_voltage_Gt;

scale_ta := TO_UNSIGNED( ratio_voltage*( cos - sin_sqrt3) , 32 );
scale_tb := TO_UNSIGNED( ratio_voltage * sin_2sqrt3, 32);

time_a <= TO_INTEGER( shift_right(scale_ta , 17) );
time_b <= TO_INTEGER( shift_right(scale_tb , 17) );
T0 <=  Tc_count - (time_a + time_b);
```

- Calcoli svolti nel codice e fattori di scala definiti:

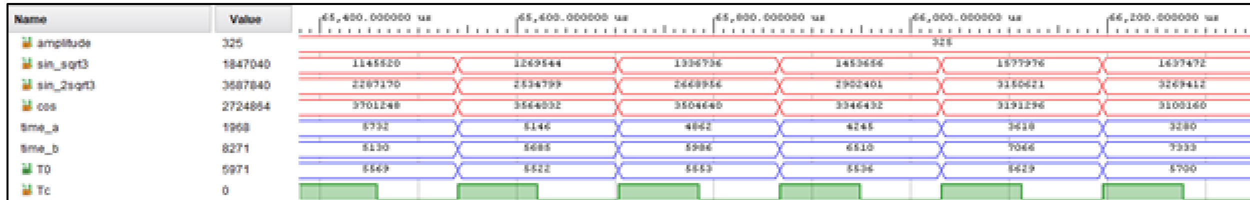
$$\begin{cases} \tilde{d}_a = \left(\frac{\|\vec{v}_{\alpha\beta}\|}{V_{DC}G_T}\, 2^9\right) \cdot A_{LUT} \cdot \left(2^9 \cos(\varepsilon) - \frac{\sqrt{3}}{3}\, 2^9\, sin(\varepsilon)\right) \\ \tilde{d}_b = \left(\frac{\|\vec{v}_{\alpha\beta}\|}{V_{DC}G_T}\, 2^9\right) \cdot A_{LUT} \cdot \left(\frac{2\sqrt{3}}{3}\, 2^9 \sin(\varepsilon)\right) \end{cases} \Rightarrow \begin{cases} \tilde{d}_a = \left(\frac{\|\vec{v_{\alpha\beta}}\|}{V_{DC}G_T}\left(\cos(\varepsilon) - \frac{\sqrt{3}}{3}\sin(\varepsilon)\right)\right) \cdot 2^{31} \\ \tilde{d}_b = \left(\frac{\|\vec{v_{\alpha\beta}}\|}{V_{DC}G_T}\frac{2\sqrt{3}}{3}\sin(\varepsilon)\right) \cdot 2^{31} \end{cases}$$

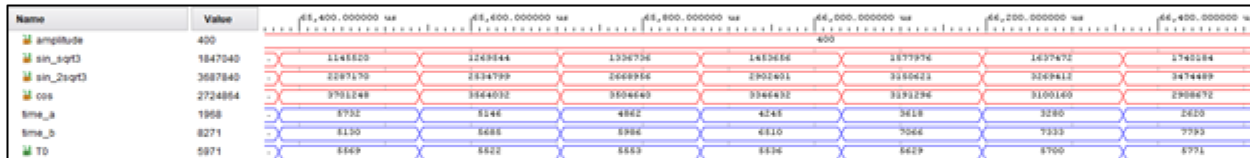
$$t_a = d_a \cdot T_c \rightarrow \text{num_cicli_ta} = \frac{\tilde{d}_a}{2^{31}}\,\text{num_cicli_Tc} = \frac{\tilde{d}_a}{2^{31}}\,2^{14} = \frac{\tilde{d}_a}{2^{17}} = \tilde{d}_a >> 17 \quad \rightarrow \text{num_cicli_Tc} = 2^{14}$$

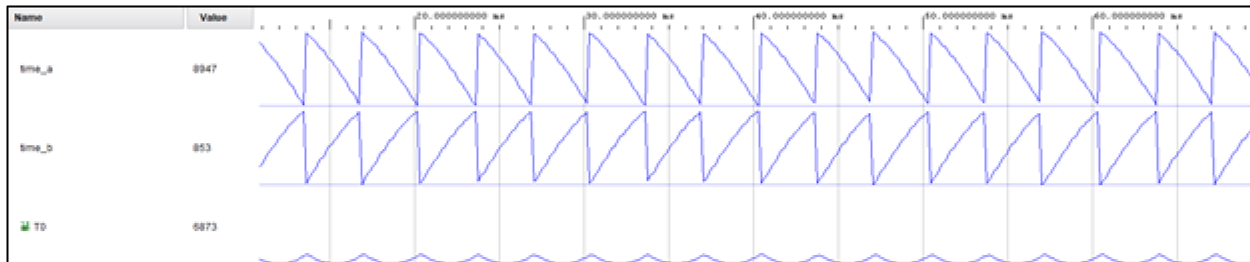
$$\rightarrow A_{lut} = 2^{13}$$

Calcolo tempi per la generazione dei vettori

- Simulazioni:







- Si nota come i valori calcolati siano sincronizzati con Tc

- Andamento analogico valori calcolati

Sistema di pilotaggio dei vettori

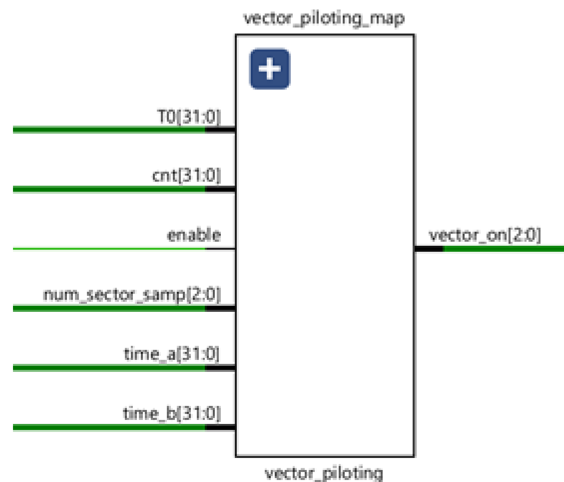
```
vector_gen : process(cnt, enable, num_sector_samp)

  variable T0_4 : integer;
  variable Ta_2 : integer;
  variable Tb_2 : integer;

  variable vect_change_1 : integer;
  variable vect_change_2 : integer;
  variable vect_change_3 : integer;
  variable vect_change_4 : integer;
  variable vect_change_5 : integer;
  variable vect_change_6 : integer;
  variable vect_change_7 : integer;

  variable vect_change_1_2 : integer;
  variable vect_change_2_2 : integer;
  variable vect_change_3_2 : integer;
  variable vect_change_4_2 : integer;
  variable vect_change_5_2 : integer;
  variable vect_change_6_2 : integer;
  variable vect_change_7_2 : integer;
```

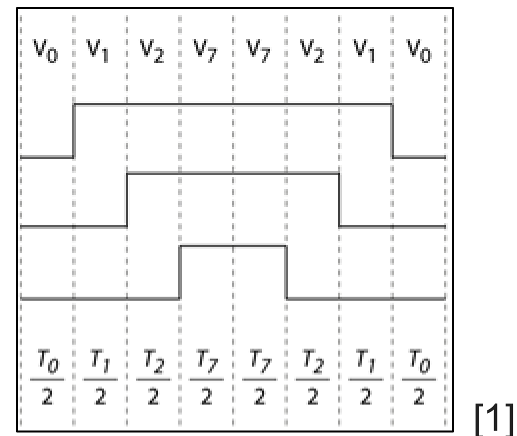
- La modulazione SVM è basata su un approccio di simmetria spaziale dei vettori. I settori pari e dispari sono trattati in modo diverso per mantenere questa simmetria, assicurando che la modulazione risulti bilanciata.

```
if(cnt = 0)then

  T0_4 := T0/4;
  Ta_2 := time_a/2;
  Tb_2 := time_b/2;

  vect_change_1 := T0_4;
  vect_change_2 := T0_4 + Ta_2;
  vect_change_3 := T0_4 + Ta_2 + Tb_2;
  vect_change_4 := T0_4 + Ta_2 + Tb_2 + 2*T0_4;
  vect_change_5 := T0_4 + Ta_2 + 2*T0_4 + time_b;
  vect_change_6 := 3*T0_4 + time_b + time_a;
  vect_change_7 := T0 + time_b + time_a;

  vect_change_1_2 := T0_4;
  vect_change_2_2 := T0_4 + Tb_2;
  vect_change_3_2 := T0_4 + Tb_2 + Ta_2;
  vect_change_4_2 := T0_4 + Tb_2 + Ta_2 + 2*T0_4;
  vect_change_5_2 := T0_4 + Tb_2 + 2*T0_4 + time_a;
  vect_change_6_2 := 3*T0_4 + time_b + time_a;
  vect_change_7_2 := T0 + time_a + time_b;
```

Sistema di pilotaggio dei vettori

```
case num_sector_samp is
  when 1 =>
    if cnt <= vect_change_1 then
      vector_on <= v0;
    end if;
    if cnt > vect_change_1 and cnt <= vect_change_2 then
      vector_on <= v1;
    end if;
    if cnt > vect_change_2 and cnt <= vect_change_3 then
      vector_on <= v2;
    end if;
    if cnt > vect_change_3 and cnt <= vect_change_4 then
      vector_on <= v7;
    end if;
    if cnt > vect_change_4 and cnt <= vect_change_5 then
      vector_on <= v2;
    end if;
    if cnt > vect_change_5 and cnt <= vect_change_6 then
      vector_on <= v1;
    end if;
    if cnt > vect_change_6 and cnt <= vect_change_7 then
      vector_on <= v0;
    end if;

    . . .
```





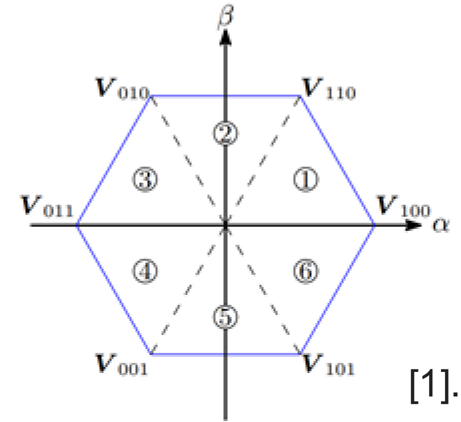
[1].

- Il codice VHDL mostra come vengono selezionati i vettori in base al conteggio (**`cnt`**) e ai valori degli intervalli di tempo **`vect_change`**, calcolati ad ogni cambio settore.
- Il diagramma a destra rappresenta un esempio di sequenza temporale dei vettori (V0, V1, V2, V7) e i relativi intervalli di tempo (T0, T1, T2, T7), evidenziando come i vettori vengono applicati nel tempo per ottenere la modulazione desiderata.

Sistema di pilotaggio degli switch

```
switch_gen : process(vector_on)
  begin

    case vector_on is
      when v0 =>
         duty_1 <= '0';
         duty_2 <= '0';
        duty_3 <= '0';
      when v1 =>
        duty_1 <= '1';
        duty_2 <= '0';
        duty_3 <= '0';
      when v2 =>
        duty_1 <= '1';
        duty_2 <= '1';
        duty_3 <= '0';
      when v3 =>
        duty_1 <= '0';
        duty_2 <= '1';
        duty_3 <= '0';
```

```
      when v4 =>
        duty_1 <= '0';
        duty_2 <= '1';
        duty_3 <= '1';
      when v5 =>
        duty_1 <= '0';
        duty_2 <= '0';
        duty_3 <= '1';
      when v6 =>
        duty_1 <= '1';
        duty_2 <= '0';
        duty_3 <= '1';
      when v7 =>
        duty_1 <= '1';
        duty_2 <= '1';
        duty_3 <= '1';
      end case;
  end process;
```

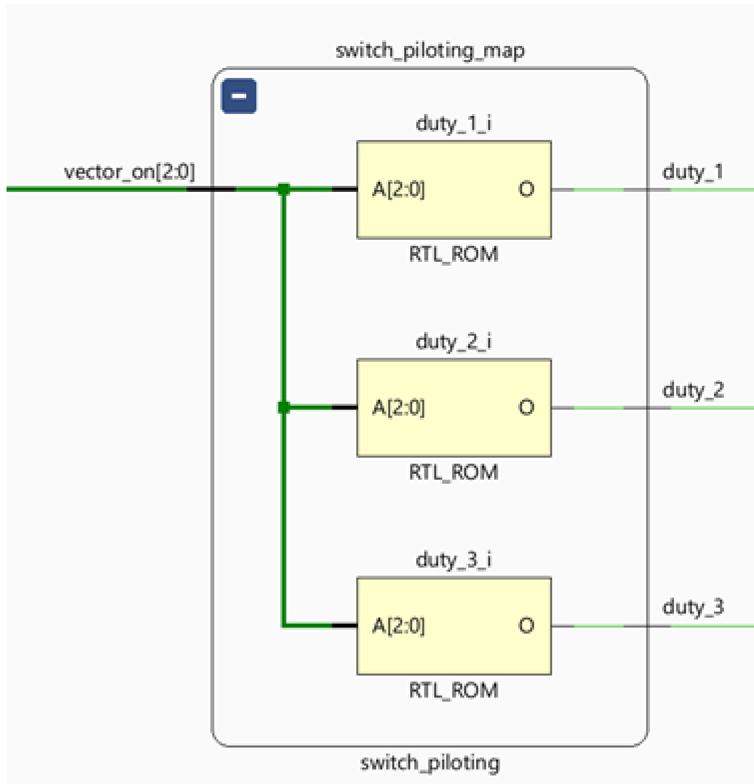


[1].

- Il codice VHDL mostra come vengono attivati gli switch in base al vettore che si vuole generare (**vector_on**), con la configurazione del duty cycle per ciascun vettore. Ogni stato del vettore (da **v0** a **v7**) corrisponde a una configurazione specifica dei duty cycle (**duty_1**, **duty_2**, **duty_3**) per le tre coppie di switch.

- Il diagramma a destra rappresenta il piano α-β con i vettori spaziali e la loro disposizione nei vari settori, evidenziando come i vettori vengono utilizzati per controllare il funzionamento degli switch nell'inverter.

Sistema di pilotaggio degli switch

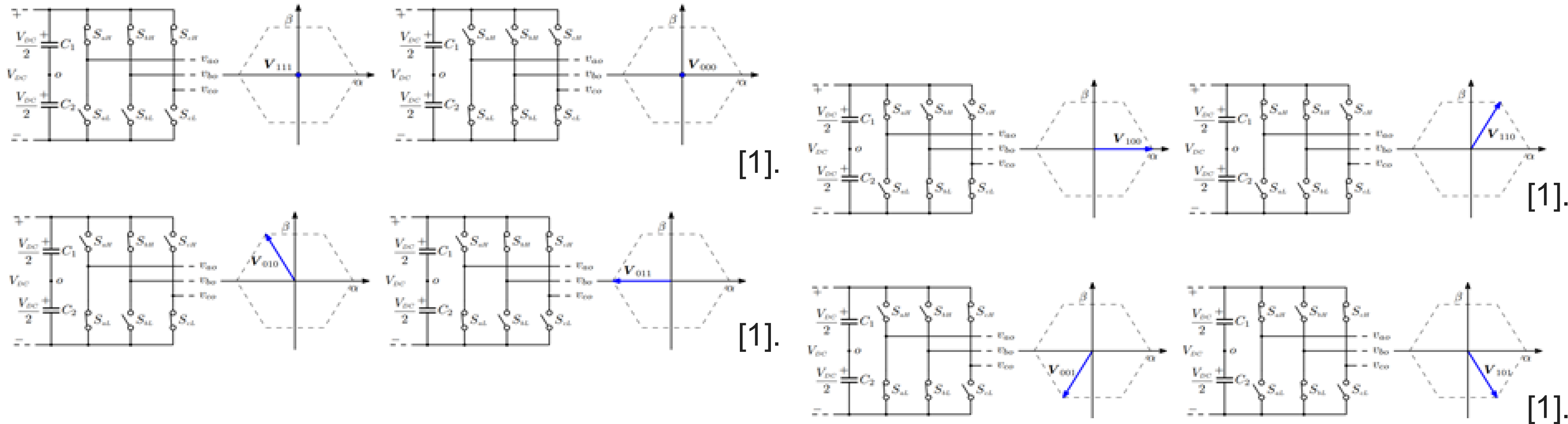


- Lo schema del blocco presenta tre blocchi ***RTL_ROM***, sono memorie a sola lettura (ROM) configurate con tabelle di lookup per generare i segnali di duty cycle desiderati in base al valore dell'ingresso **vector_on[2:0]**. Le uscite **duty_1**, **duty_2**, e **duty_3** forniscono i segnali di duty cycle utilizzati per pilotare switch.

1222-2022 800 ANNI

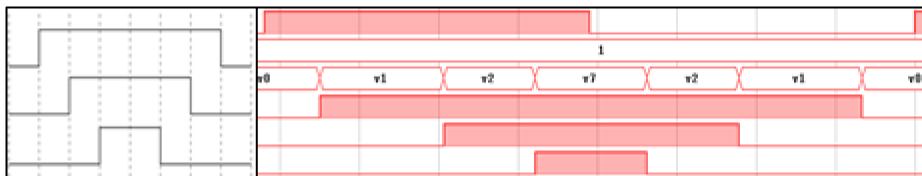
UNIVERSITÀ DEGLI STUDI DI PADOVA

Sistema di pilotaggio degli switch

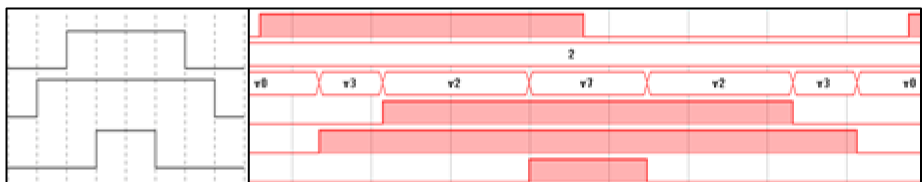


- L'immagine mostra vari stati di commutazione dei transistor (`S_1`, `S_2`, ..., `S_6`). Ogni schema è accompagnato da un diagramma di spazio vettoriale che rappresenta le transizioni dei vettori di tensione corrispondenti (`V_000`, `V_111`, ecc.). Le frecce blu nei diagrammi indicano la direzione e la magnitudine dei vettori di tensione risultanti per ciascuna configurazione di commutazione.

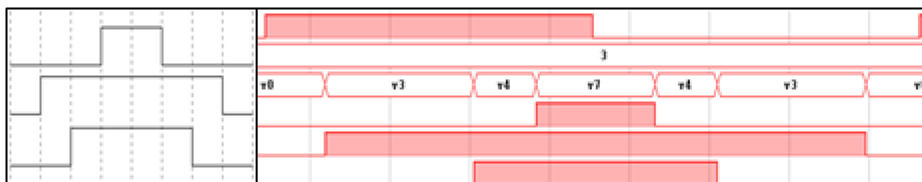
- Confronto tra il diagramma teorico e la simulazione reale nella generazione dei vettori



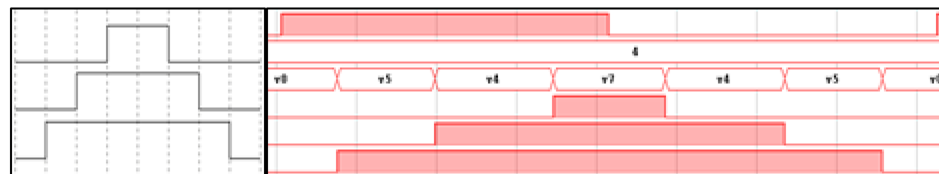
1° quadrante



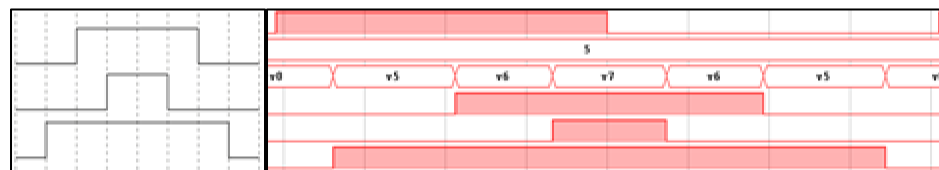
2° quadrante



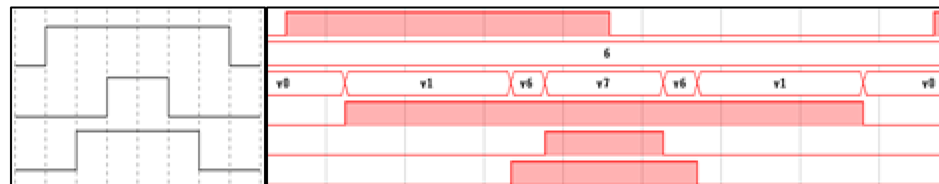
3° quadrante



4° quadrante

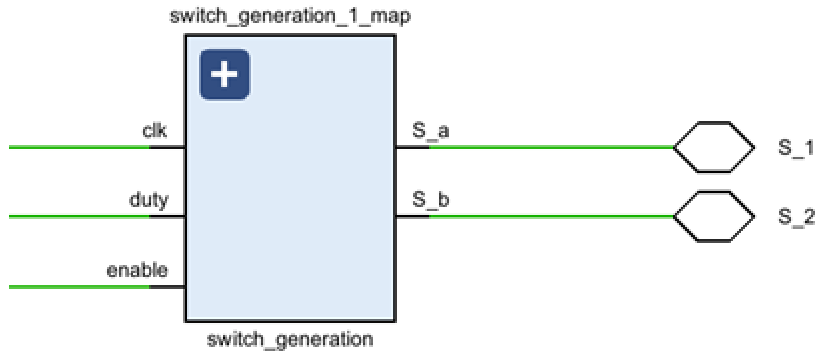


5° quadrante



6° quadrante

Generatore segnali switches

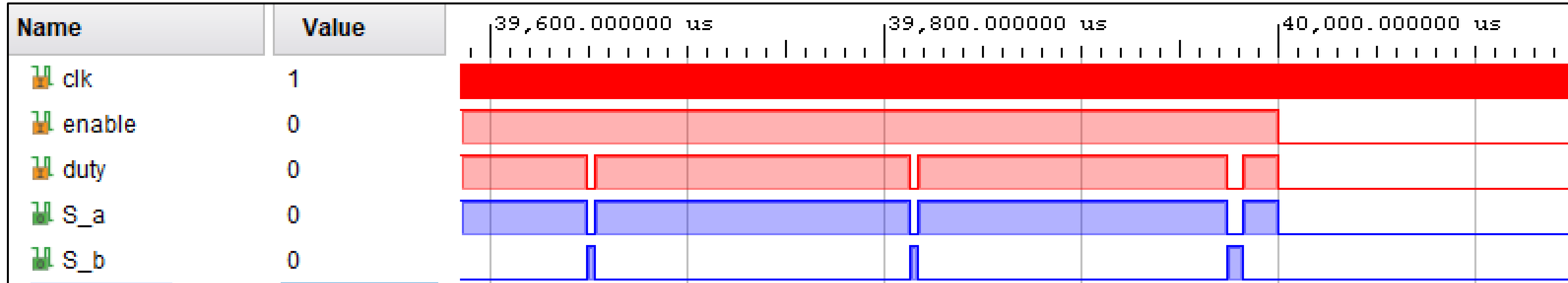


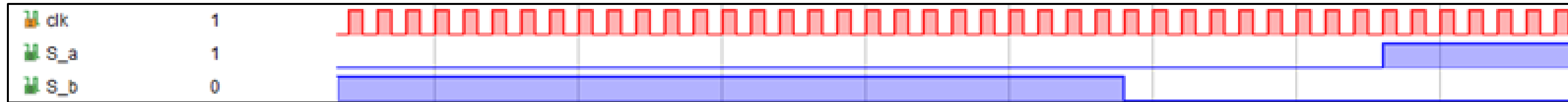
- Il blocco "switch_generation" prende in ingresso il segnale di duty cycle e il clock, generando i segnali di commutazione `S_a` e `S_b` con una gestione accurata dei tempi morti per prevenire cortocircuiti.

- Quando il duty cycle cambia, il blocco spegne immediatamente uno dei segnali di uscita e avvia un conteggio del tempo morto. Al termine del tempo morto, il segnale appropriato viene acceso, garantendo che i segnali S_a e S_b non si sovrappongano mai.

Generatore segnali switches

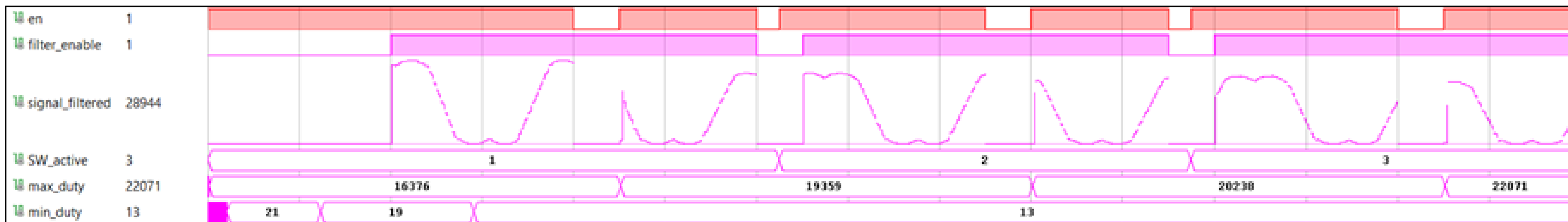
- Simulazioni:



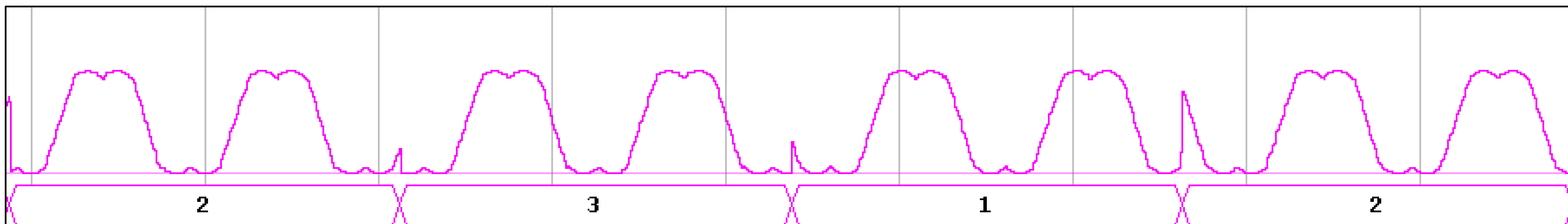


- Le simulazioni confermano il corretto funzionamento del blocco di generazione dei segnali di commutazione. Le implementazioni della modulazione SVM offrono vantaggi significativi rispetto alla modulazione PWM tradizionale, migliorando l'efficienza del sistema.

Filtro in simulazione

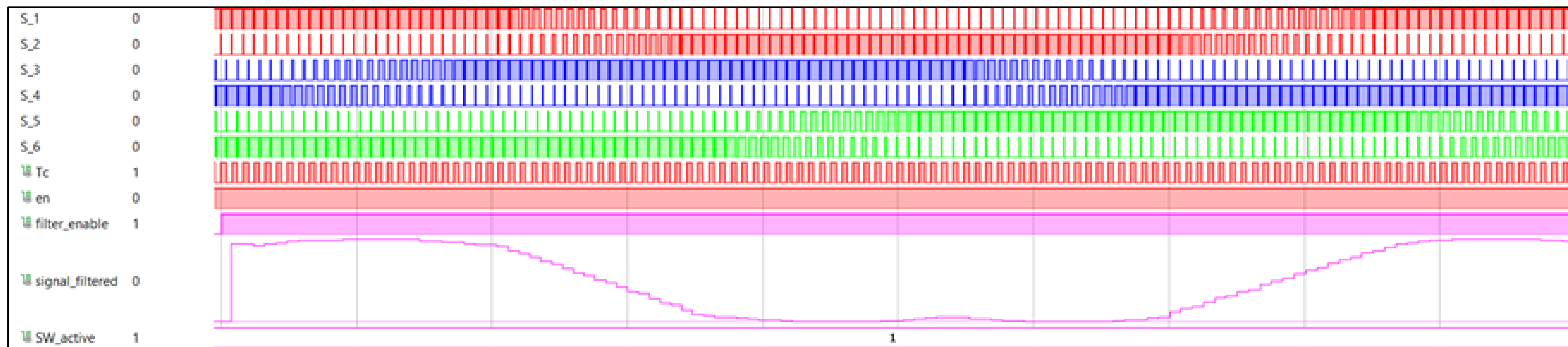


- funzionamento segnale di **`filter_enable`** ed enable (**`en`**) sulle tre fasi, con calcolo dei valori di **`max_duty`** e **`min_duty`** nei primi 20 ms della simulazione.



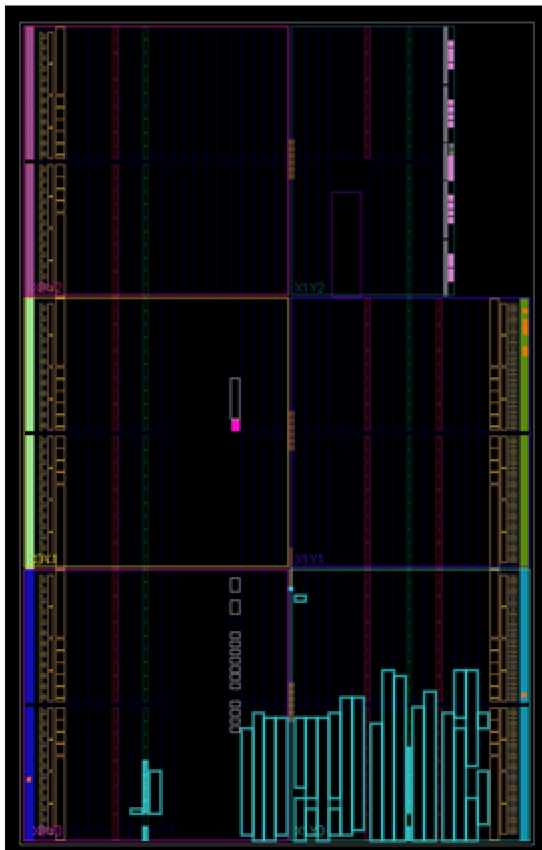
- simulazione del filtro per le singole fasi senza interruzioni

Simulazioni



- Il grafico mostra la simulazione attraverso i segnali di controllo e commutazione dell'inverter. Viene evidenziato il duty cycle degli switch S_1, S_2, S_3, S_4, S_5 e S_6, con particolare attenzione al comportamento della prima coppia di switch filtrati.

Utilizzo

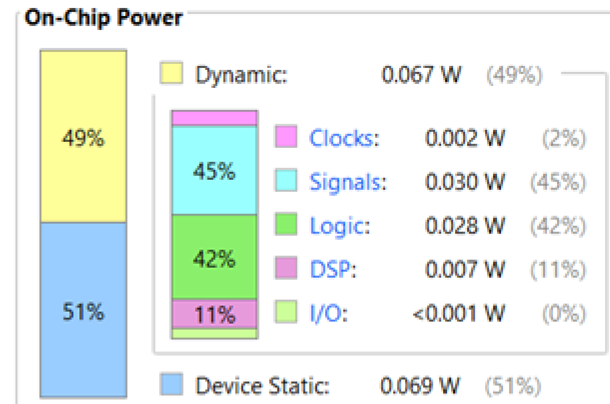


- L'immagine rappresenta il design implementato, mostra come il progetto è stato mappato sull'hardware dell'FPGA.

- Questa visualizzazione aiuta a comprendere la distribuzione e le interconnessioni degli elementi logici

Utilizzo

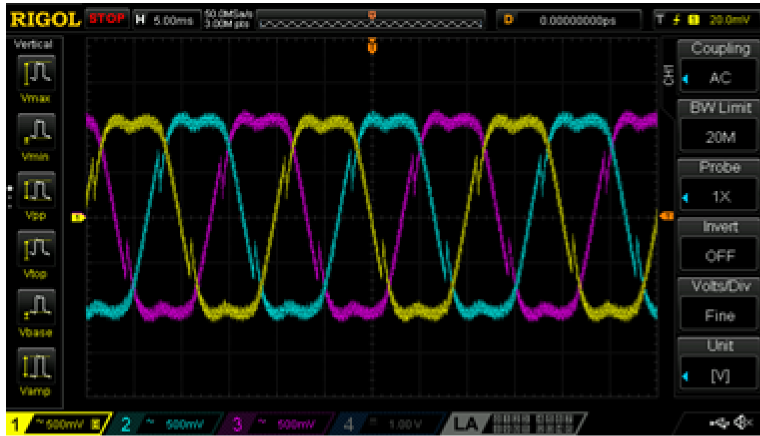
Resource	Utilization	Available	Utilization %
LUT	2574	20800	12.38
FF	208	41600	0.50
DSP	11	90	12.22
IO	8	106	7.55
BUFG	2	32	6.25



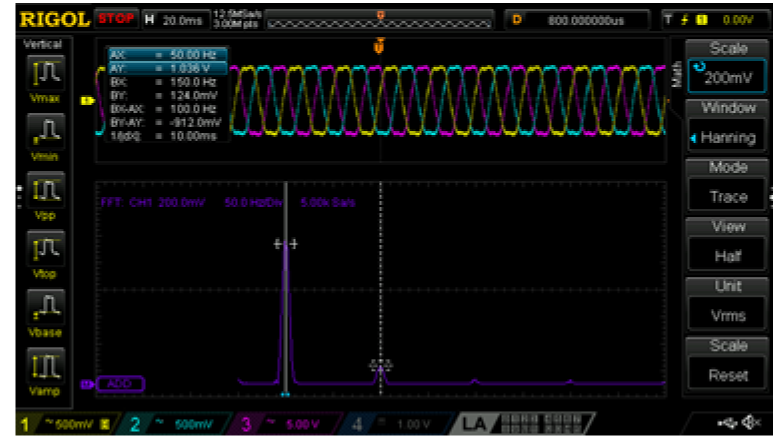
- Questa tabella mostra l'utilizzo delle risorse hardware di un FPGA, indicando quante unità di ciascuna risorsa sono utilizzate, quante sono disponibili e la percentuale di utilizzo.
- Questo grafico illustra la distribuzione del consumo di potenza on-chip di un FPGA, suddivisa in potenza dinamica e statica.

Risultati di Laboratorio

- Forme d'onda:



- Analisi in frequenza (FFT):



- Si riporta quanto ottenuto in laboratorio con un set-up del tutto analogo al caso della modulazione PWM
- Si nota che le forme d'onda generate hanno andamento corrispondente a quanto desiderato, ovvero la terna che si otterrebbe tramite la tecnica PWM con Iniezione di Terza Armonica.
- Osservando il contenuto armonico si hanno due picchi: uno centrato a 50Hz ed uno centrato a 150Hz. Quest'ultimo corrisponde alla terza armonica

Bibliografia

[1]. Buso, Simone, and Paolo Mattavelli. *Digital control in power electronics*. Morgan & Claypool Publishers, 2015.

UNIVERSITÀ DEGLI STUDI DI PADOVA

GRAZIE PER L’ATTENZIONE