



**POLITECNICO
DI TORINO**

*Corso di Laurea Magistrale
in
Ingegneria Elettronica*

Anno Accademico 2017/2018

Sistemi Digitali Integrati

*Relazione di laboratorio
Progetto di una butterfly*

Studente: Barezzi Mattia

Matricola: 252967

Studente: Borello Gabriele

Matricola: 252845

Studente: Canzonieri Gianmarco

Matricola: 244123

Introduzione

Lo scopo di questo progetto è creare una unità di elaborazione chiamata butterfly che costituisce l'elemento base per realizzare una FFT classica.

Il sistema è stato realizzato seguendo tutti i passi richiesti: dalla creazione del dataflow graph, sviluppo di datapath e control unit fino alla scrittura del codice VHDL e test usando sensate variabili di ingresso.

Date le specifiche di progetto, si hanno i dati ricevuti dall'esterno A_r , A_i , B_r , B_i , W_r , W_i che sono le parti reali (dove abbiamo r) e le parti immaginarie (dove abbiamo i) delle variabili A , B , W definite in forma frazionaria in complemento a due su 24 bit in un intervallo compreso da -1 e 1 esclusi.

Il sistema progettato prevede che il campionamento dei dati sia sfasato in modo tale da ottimizzare il flusso di lavoro quando viene eseguito in modalità "continua".

Si può vedere graficamente: abbiamo due butterfly (una in blu e una in arancione) in cascata dove quando vengono campionati i primi dati dalla seconda butterfly, la prima butterfly sta elaborando il secondo blocco di dati della prima operazione (che verranno campionati nel colpo di clock successivo dalla seconda butterfly).

/	A_{r_1}	MEM A_{r_2}	/
/	B_{r_1}	MEM B_{r_2}	/
/	/	A_{i_1}	MEM A_{i_2}
/	/	B_{i_1}	MEM B_{i_2}

Nella relazione di laboratorio sono presenti i seguenti paragrafi:

- Dataflow graph
- Datapath
- Control unit
- ASM chart
- Control ASM chart
- Simulazioni
- Codice VHDL

Dataflow graph

La prima fase del progetto è dedicata allo sviluppo del dataflow graph che permette lo studio del miglior scheduling che ottimizzi timing, uso degli operatori e uso delle variabili.

Le specifiche del progetto richiedono l'uso di un singolo moltiplicatore con due stadi di pipeline e l'uso di due sommatore con un singolo livello di pipeline. Inoltre, come vedremo nel dettaglio nelle prossime sezioni, si

utilizza un blocco di troncamento per adattare i dati in uscita dalla butterfly all'eventuale successivo stadio di butterfly. Si nota inoltre che ci sono due operazioni che sono semplici moltiplicazioni per due del dato in ingresso: in questo caso non è necessario l'uso del moltiplicatore ma si può usare un semplice shifter aritmetico che non necessita di gate per la sua realizzazione.

L'uso dei colori permette una facile lettura del flusso di operazioni fatte:

- Verde: moltiplicatore da 24 bit
- Arancione: primo sommatore da 50 bit
- Viola: secondo sommatore da 50 bit
- Rosso: primo modulo di troncamento
- Bianco: secondo modulo di troncamento

Le operazioni fatte sono:

$$\text{MPY1} = \text{Br} * \text{Wr}$$

$$\text{MPY2} = \text{Bi} * \text{Wi}$$

$$\text{MPY3} = \text{Br} * \text{Wi}$$

$$\text{MPY4} = \text{Bi} * \text{Wr}$$

$$\text{SFT5} = 2 * \text{Ar}$$

$$\text{SFT6} = 2 * \text{Ai}$$

$$\text{SUM1} = \text{Ar} + \text{MPY1}$$

$$\text{SUM2} = \text{SUM1} - \text{MPY2} = \text{Ar}$$

$$\text{SUM3} = \text{Ai} + \text{MPY3}$$

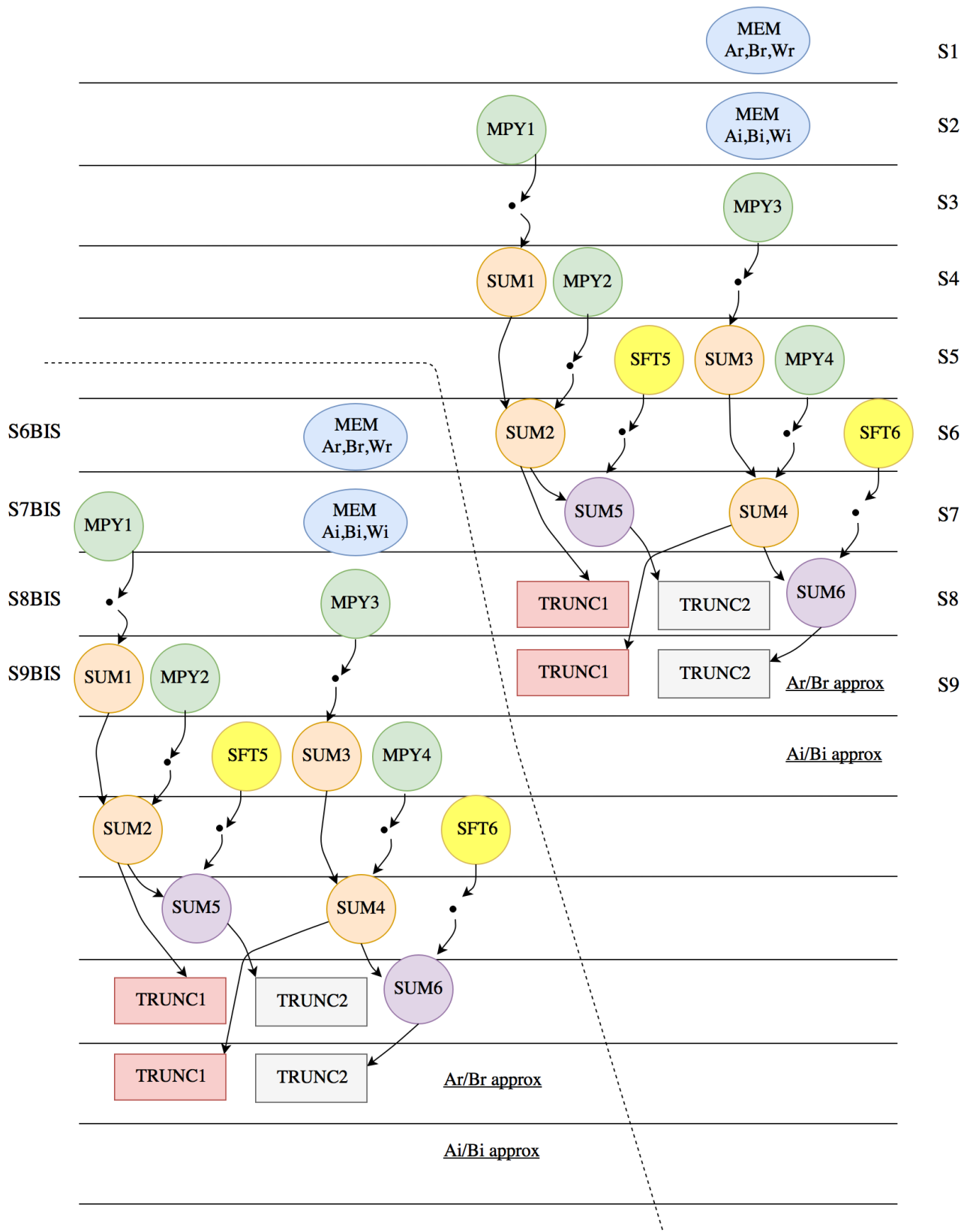
$$\text{SUM4} = \text{SUM3} + \text{MPY4} = \text{Ai}$$

$$\text{SUM5} = \text{MPY5} - \text{SUM2} = \text{Br}$$

$$\text{SUM6} = \text{MPY6} - \text{SUM4} = \text{Bi}$$

Dall'immagine, mostrata nella pagina successiva, abbiamo in alto a destra la sequenza di avvio. Nel caso l'elaborazione fosse di tipo continua, il sistema progettato effettua, in aggiunta alle operazioni a destra, anche le operazioni a sinistra e così via finché l'esecuzione rimane di tipo "continuo".

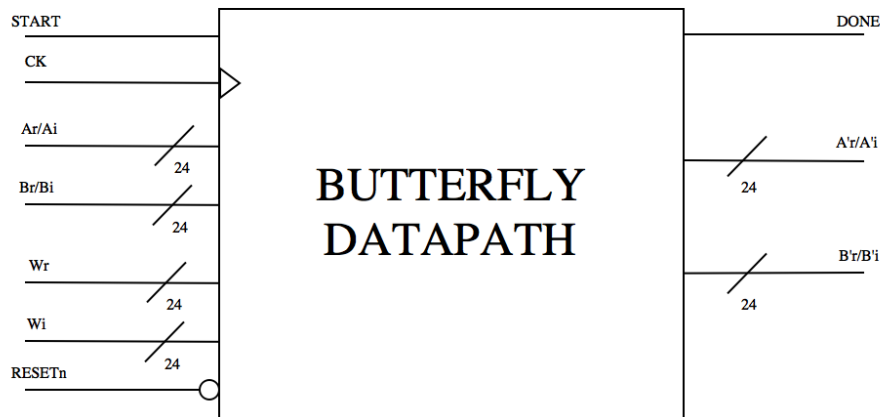
Abbiamo fatto delle scelte progettuali mirate a throughput e latenza: il throughput migliora all'aumentare della frequenza di funzionamento. Nel caso del nostro progetto, aumentare il numero di registri temporanei, aumenterebbe i livelli di pipe, potendo migliorare dunque la frequenza di funzionamento. Però aumentando i livelli di pipe, aumenta la latenza.



Dataflow graph

Datapath

Il blocco datapath è sintetizzabile come segue.



Blocco datapath

Si ricorda che le variabili Ar e Ai e Br e Bi condividono la stessa linea avendo scelto l'ottimizzazione descritta nel capitolo precedente.

All'interno del blocco si utilizza il metodo del “Unconditional Block Floating Point Scaling” che consiste nel non inserire alcun bit di guardia in ingresso per evitare l'overflow dei dati.

Abbiamo derivato l'architettura del datapath e valutato il ritardo combinatorio dei singoli componenti tenendo conto delle specifiche date (2 liv. pipe per il moltiplicatore e 1 liv. di pipe per i sommatore). Il moltiplicatore è a 24 bit, avendo due livelli di pipe, bisogna dividere in 3 il circuito e quindi abbiamo 3 moltiplicatori da 8 bit. In seguito abbiamo analizzato il sommatore (anche mantenendo l'intero dato) che ha il percorso critico contenente una metà del primo sommatore e una metà del secondo sommatore: il totale dei full adder è $25+25 = 50$.

Aggiungendo un livello di pipe all'uscita del moltiplicatore si migliorerebbe il throughput ma peggiorerebbe la latenza quindi questa scelta non è stata adottata. L'altra idea è piazzare il registro all'uscita del sommatore alpha verso il sommatore beta per spezzare la somma da 50 a 25 ma questo non migliorerebbe la frequenza di funzionamento. Con ogni probabilità, infatti, il percorso critico è quello che comprende l'ultimo terzo del moltiplicatore e la prima metà del sommatore alpha.

Si è quindi ottimizzata la latenza a discapito del throughput.

A seguito di considerazioni progettuali legate all'intero ciclo di vita delle variabili del sistema si è scelto un parallelismo di 50 bit per i due sommatore e 24 bit per il moltiplicatore.

L'ultimo blocco prima dell'uscita è quello del troncamento che svolge l'operazione di arrotondamento solo sui dati in uscita dalla butterfly usando la tecnica del “Rounding To Half Up”. Questo richiede un blocco aritmetico con un livello di pipe.

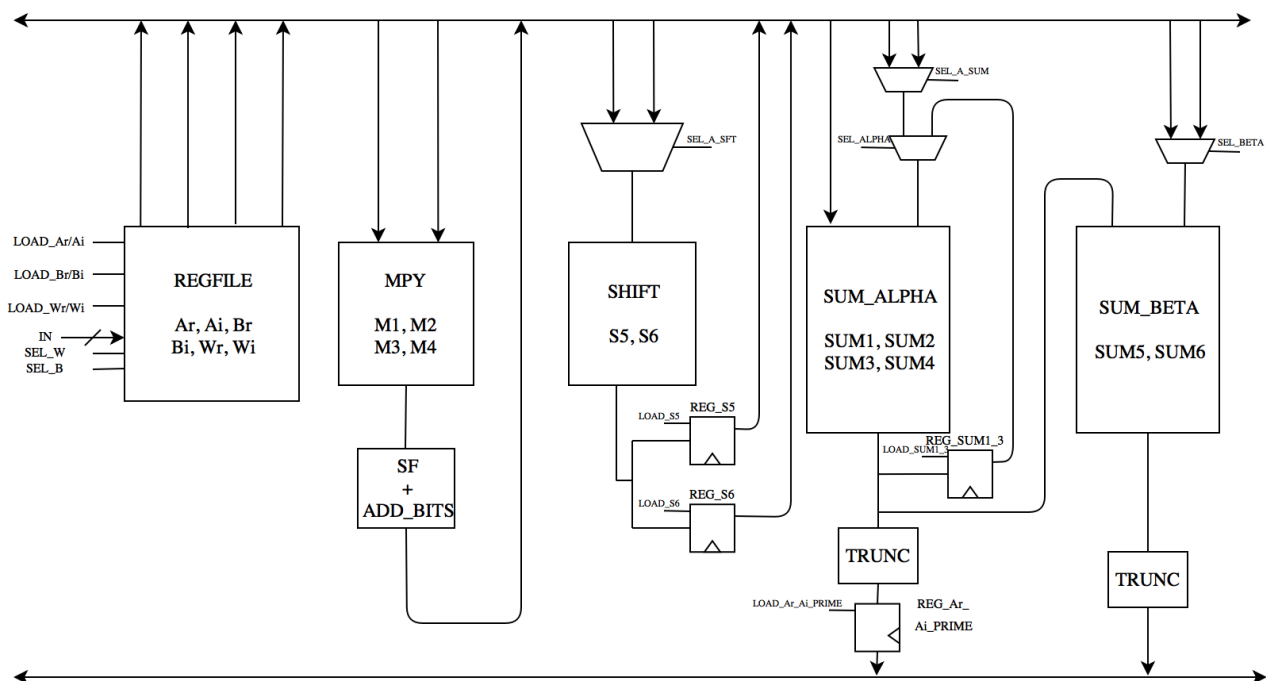
Per la sincronizzazione dei dati in ingresso e in uscita tra butterfly in cascata si è pensato di utilizzare il segnale di DONE attivato quando il primo dei due blocchi di dati è disponibile: questo attiva la seconda butterfly avendo collegato tra loro il segnale di DONE della prima butterfly e lo START della seconda butterfly.

In questo modo si può discriminare il caso in cui ho un'elaborazione isolata o un'elaborazione continua.

Si è ottimizzato il numero di bus globali utilizzando solo bus locali che da un punto di vista di economia del sistema costano come il bus globale, ma hanno il vantaggio di permettere più operazioni in parallelo. Quindi l'uso dei bus locali aumenta la microconcorrenza del sistema e questo ci permette di massimizzare le prestazioni.

Datapath (derivazione architetturale)

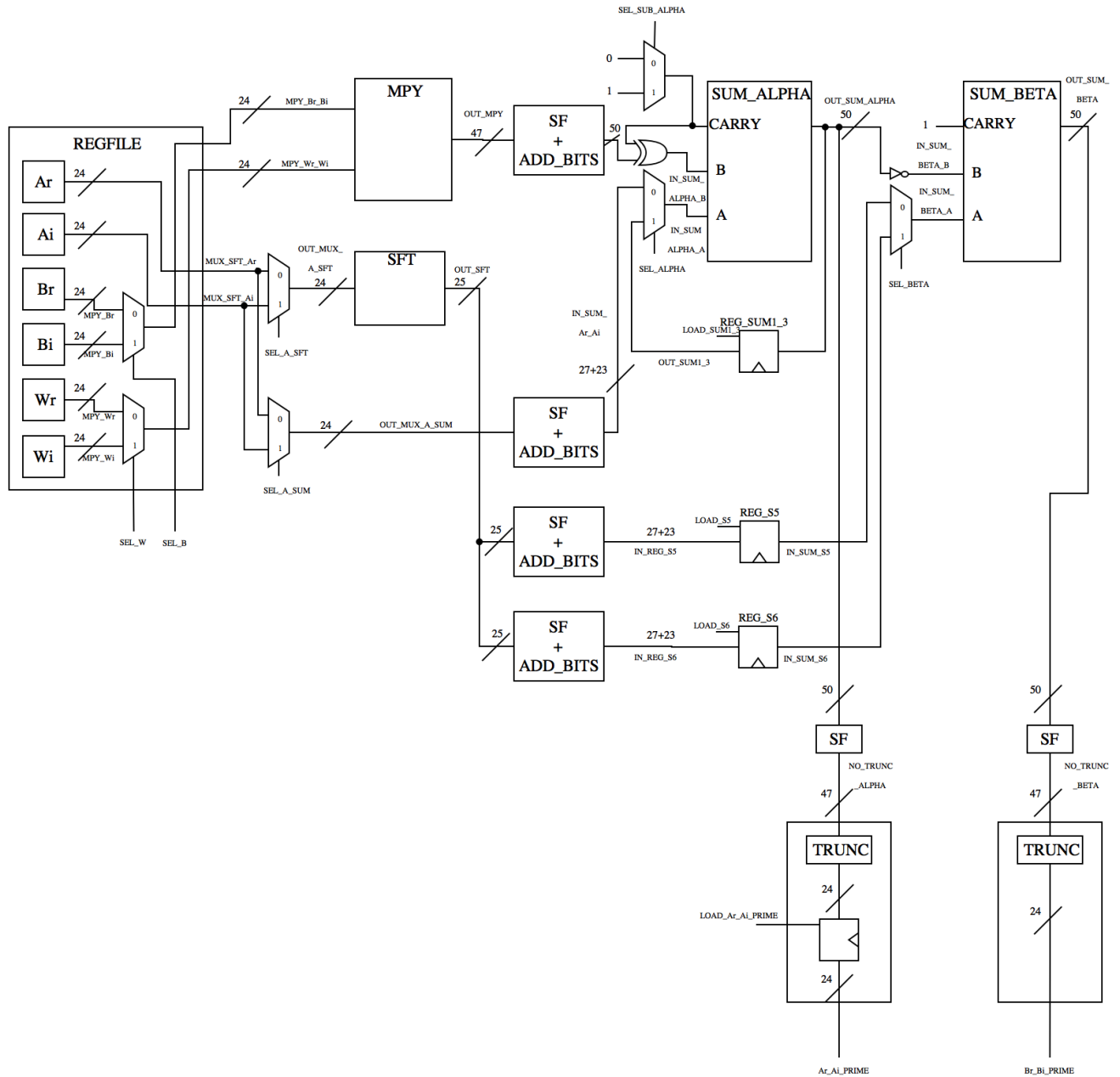
Si è realizzato quindi una prima derivazione architetturale (più semplice anche da studiare per capire il funzionamento della macchina) senza curarci dei singoli collegamenti tra i vari componenti.



Datapath (derivazione architetturale)

Datapath fisico

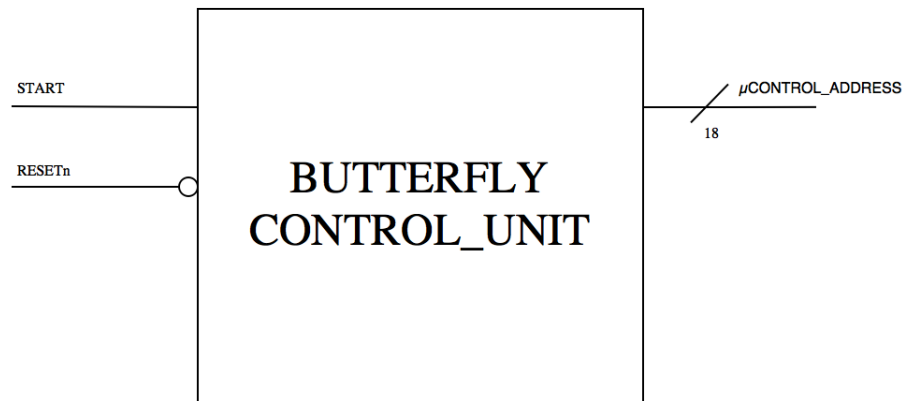
Quando è stata definita la prima derivazione architetturale si è passato al vero datapath che descrive tutti i collegamenti. Per ogni linea c'è il numero di bit di parallelismo e il nome della linea che viene poi utilizzato nel codice VHDL.



Datapath fisico

Control unit

Il blocco control unit è sintetizzabile come segue.

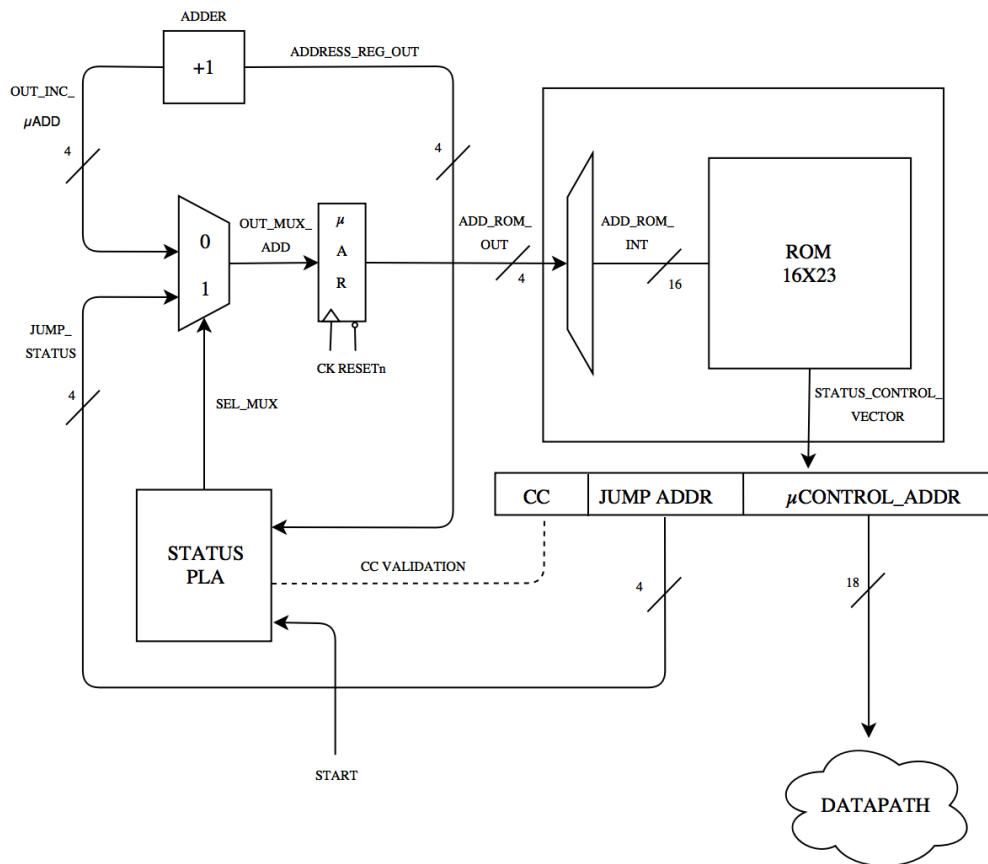


Blocco control unit

Struttura della control unit

La control unit è realizzata mediante la tecnica della microprogrammazione utilizzando un sequenziatore con indirizzamento implicito utilizzando la tecnica del “Conditional Sequencing PLA” per la gestione degli indirizzi di salto. Nella control unit, utilizzando l’indirizzamento implicito, il percorso critico deriva dal sommatore legato al μ address register (che consiste in una catena di full adder) per cui tramite la tecnica della “Design by Contraction”, utilizzabile perché viene sommato sempre lo stesso valore + 1, si potrebbe rendere il percorso combinatorio più veloce migliorando le prestazioni del sistema.

Anche qui abbiamo per ogni linea il numero di bit del parallelismo e il nome utilizzato nel codice VHDL.

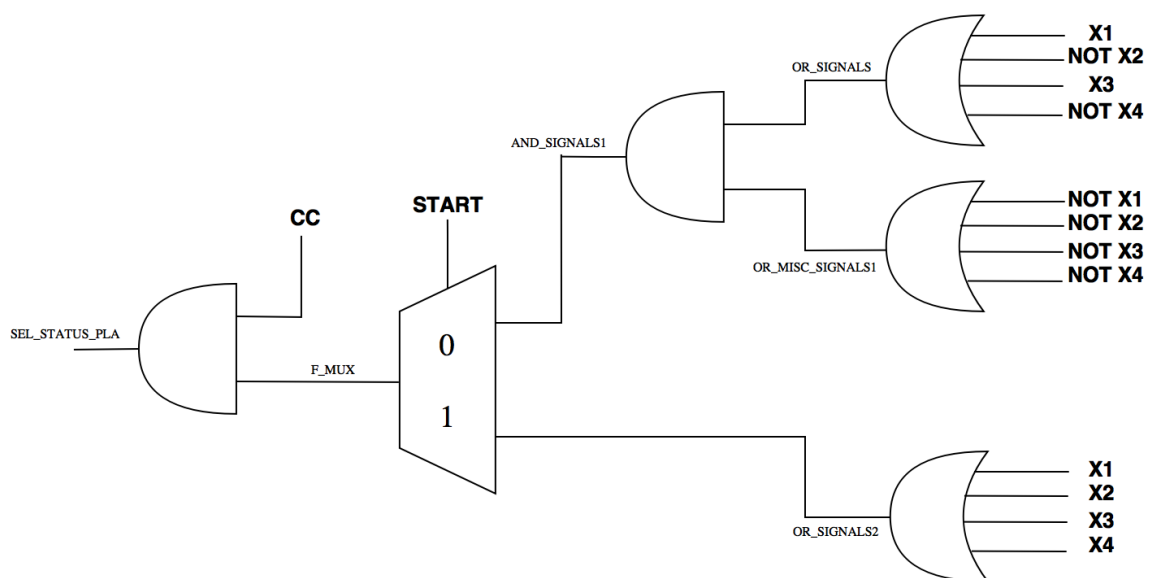


Datapath della control unit

Descrizione del blocco Status PLA

Si è scelto di realizzare il blocco PLA in modo combinatorio.

Il circuito logico della status PLA è stata realizzata come sotto.



Circuito logico della status PLA

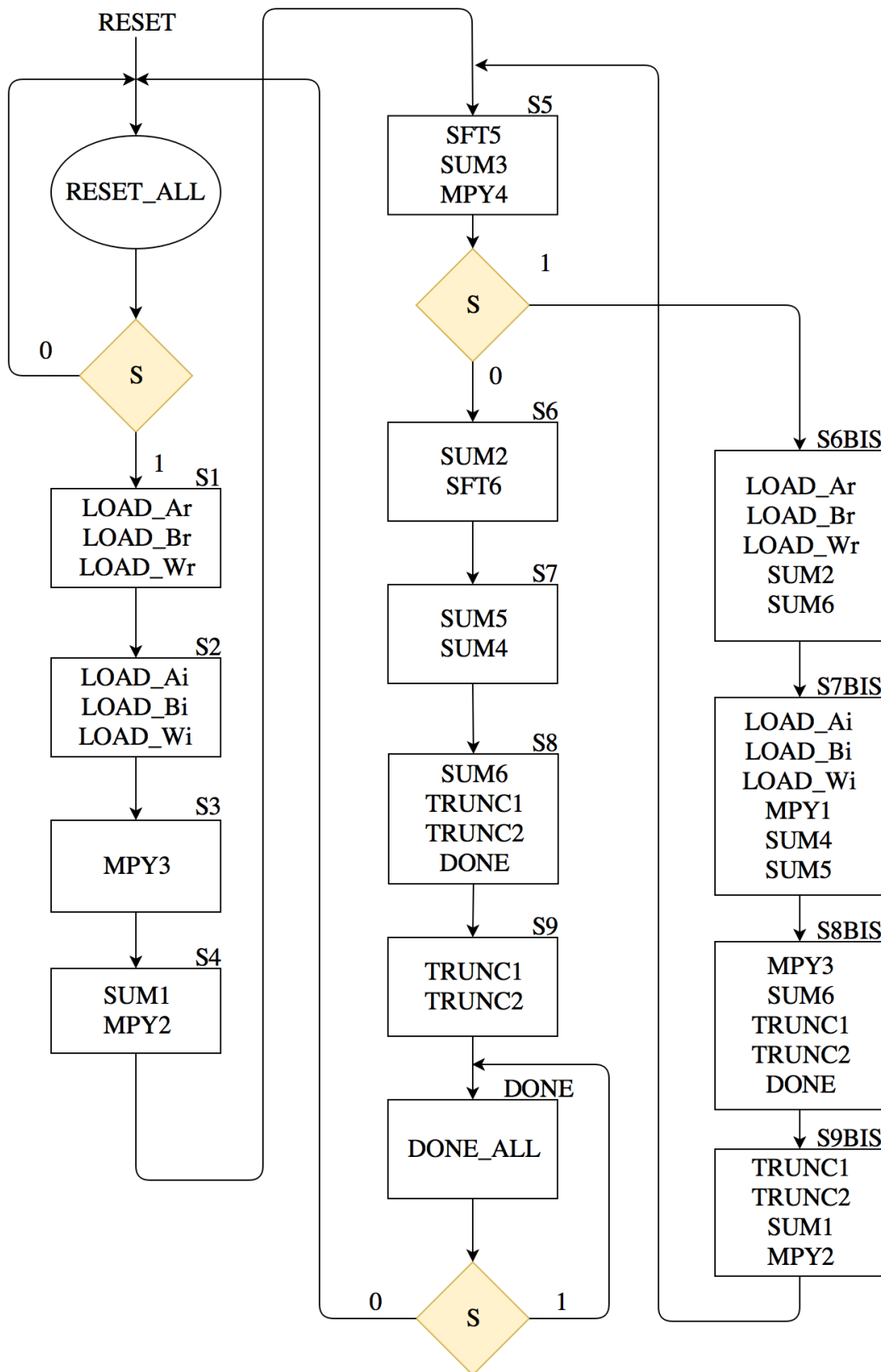
Codici del sequenziatore e del command generator

Questi sono i codici per il sequenziatore e il command generator contenuti nella μ ROM.

Stato	μAR				Indirizzo dello stato successivo					LOAD_Ar	LOAD_Ai	LOAD_Br	LOAD_Bi	LOAD_Wr	LOAD_Wi	LOAD_S5	LOAD_S6	SEL_ALPHA	SEL_BETA	LOAD_SUM1_3	SEL_W	SEL_B	SEL_A_SFT	SEL_A_SUM	SEL_SUB_ALPHA	DONE	LOAD_Ar_Ai_PRIME
RESET	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S1	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
S2	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
S3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
S4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
S5	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0
S6	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0	1	0	0
S7	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
S8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	1
S9	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
S6B	1	0	1	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	1	1	0	1	0	0
S7B	1	0	1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0
S8B	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1
S9B	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
RESET	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DONE	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

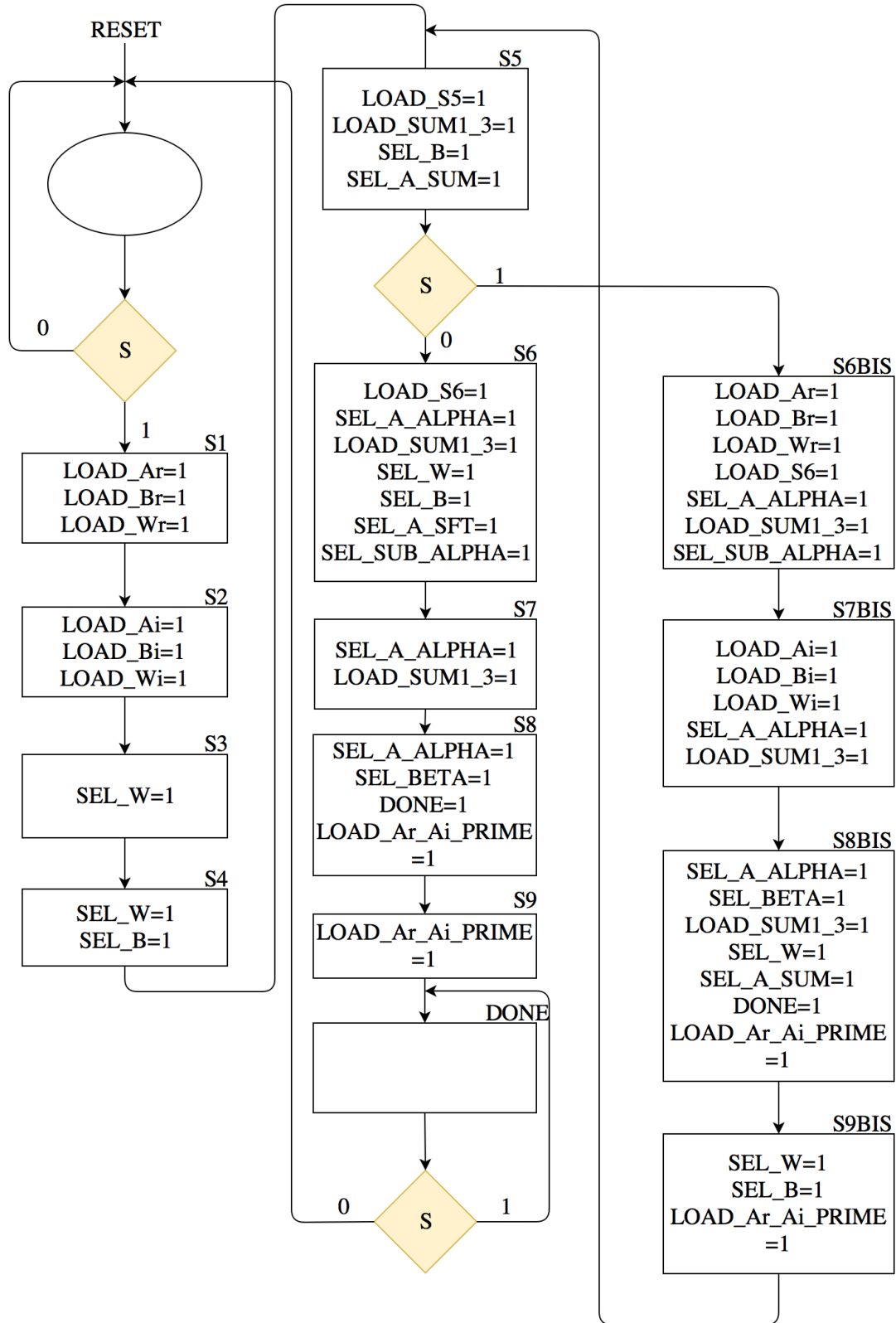
ASM chart

L'ASM chart del circuito sintetizza graficamente le istruzioni dello pseudocodice.

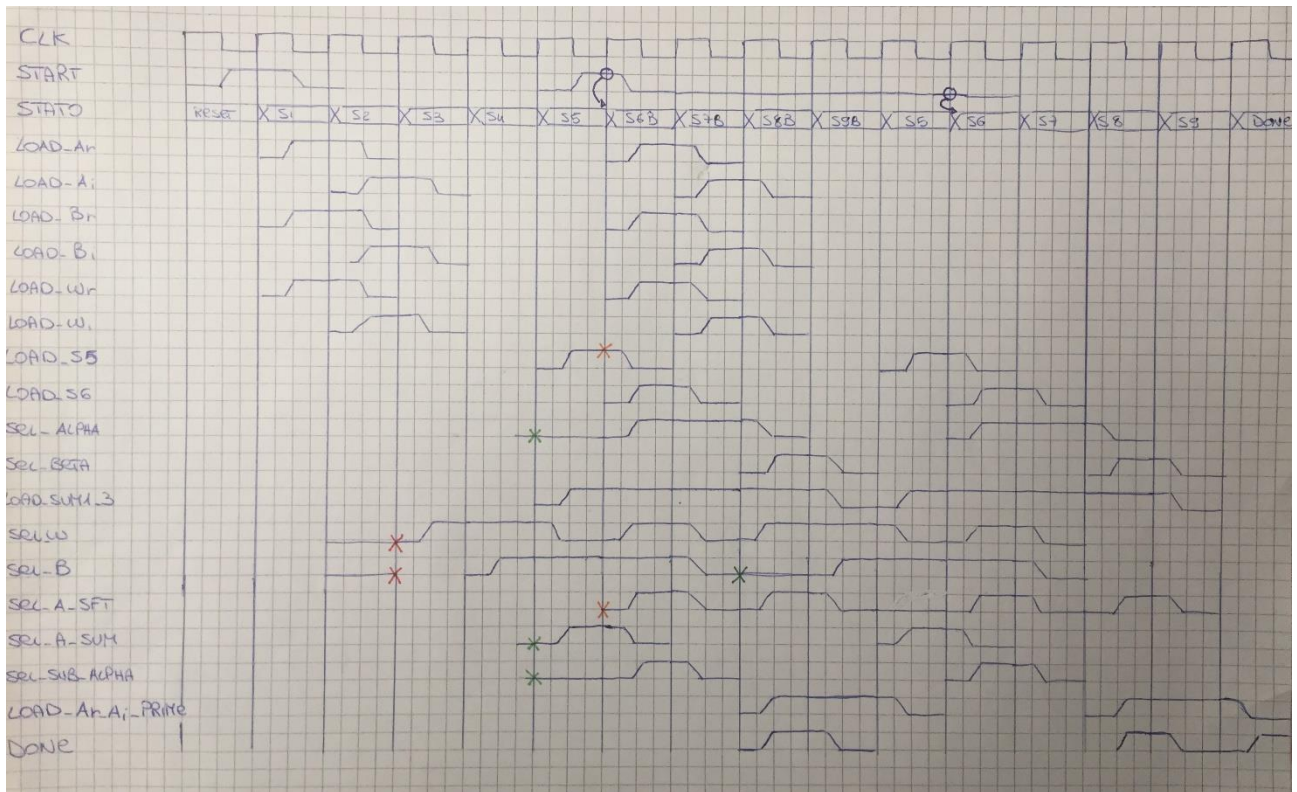


Control ASM chart

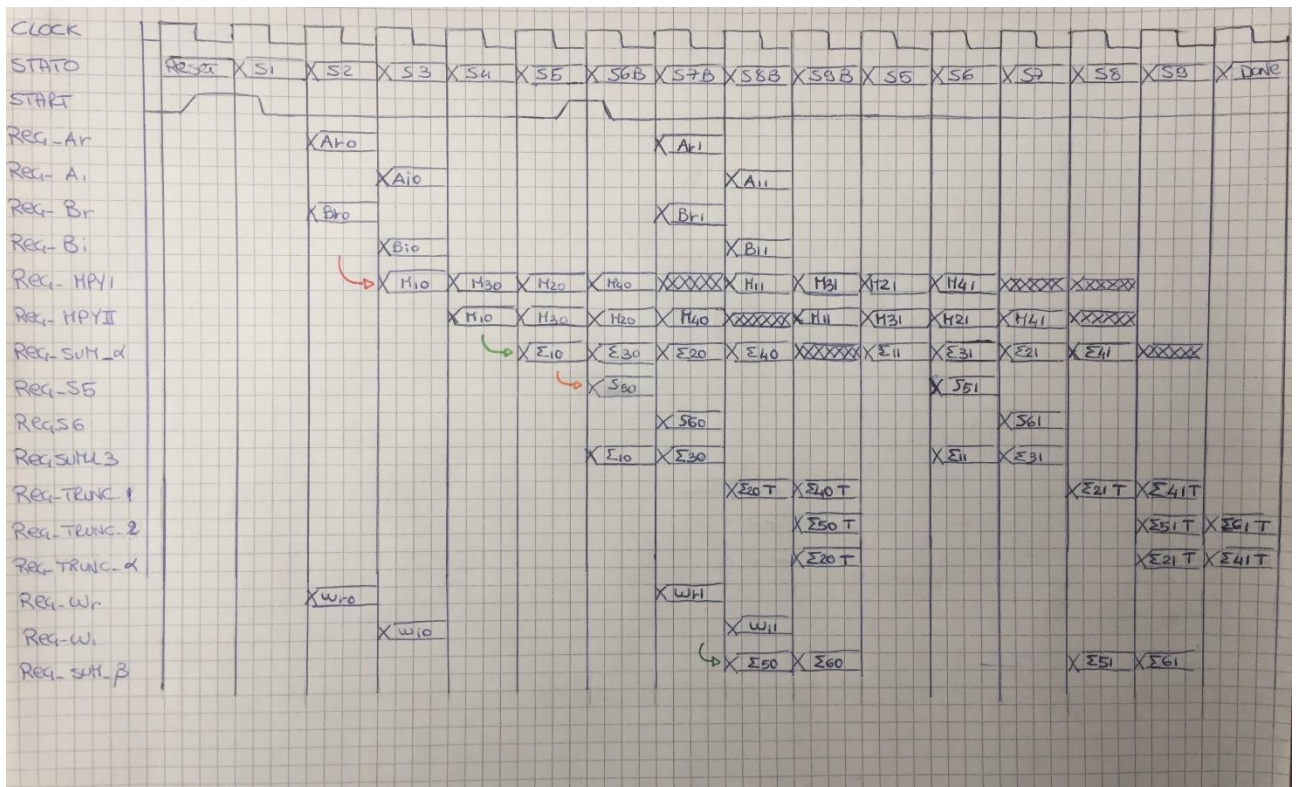
La control ASM chart si basa sullo stesso schema del grafico precedente, ma contiene al suo interno i segnali che vengono abilitati in ogni fase del processo.



Timing



Timing dei segnali



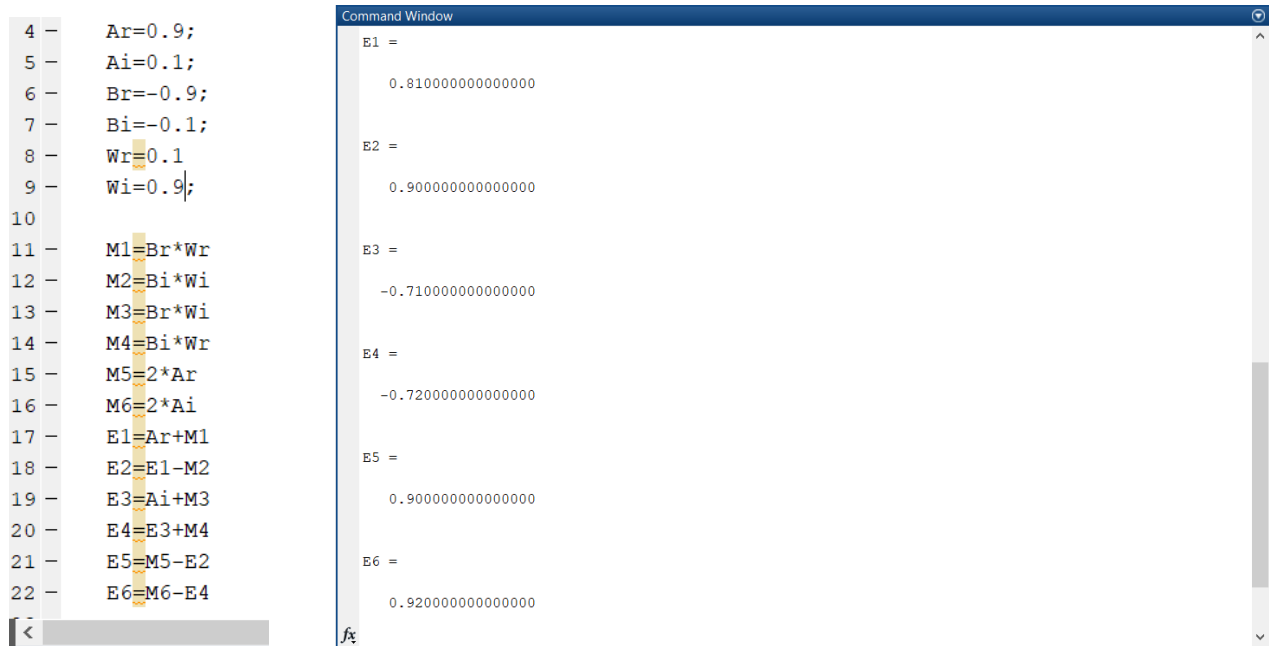
Timing dei registri

Simulazioni

NOTA: la stesura del progetto è stata fatta usando il linguaggio VHDL, il compilatore QUARTUS II.

Caso di una esecuzione isolata

Il vettore di test è stato prima simulato usando MATLAB.



The image shows a MATLAB script on the left and its execution results in the Command Window on the right. The script defines input variables Ar, Ai, Br, Bi, Wr, Wi and calculates intermediate variables M1 through M6, and final outputs E1 through E6. The Command Window displays the numerical results for E1 through E6.

```
4 - Ar=0.9;  
5 - Ai=0.1;  
6 - Br=-0.9;  
7 - Bi=-0.1;  
8 - Wr=0.1  
9 - Wi=0.9;  
10  
11 - M1=Br*Wr  
12 - M2=Bi*Wi  
13 - M3=Br*Wi  
14 - M4=Bi*Wr  
15 - M5=2*Ar  
16 - M6=2*Ai  
17 - E1=Ar+M1  
18 - E2=E1-M2  
19 - E3=Ai+M3  
20 - E4=E3+M4  
21 - E5=M5-E2  
22 - E6=M6-E4
```

Command Window

```
E1 =  
    0.8100000000000000  
  
E2 =  
    0.9000000000000000  
  
E3 =  
   -0.7100000000000000  
  
E4 =  
   -0.7200000000000000  
  
E5 =  
    0.9000000000000000  
  
E6 =  
    0.9200000000000000
```

Risultato della simulazione su MATLAB

A questo punto sono stati inseriti gli stessi dati in ingresso alla butterfly, simulando il caso di esecuzione isolata.

La macchina viene inizializzata mettendo a 0 il RESETn. A questo punto vengono forniti in ingresso i dati e questi vengono elaborati.

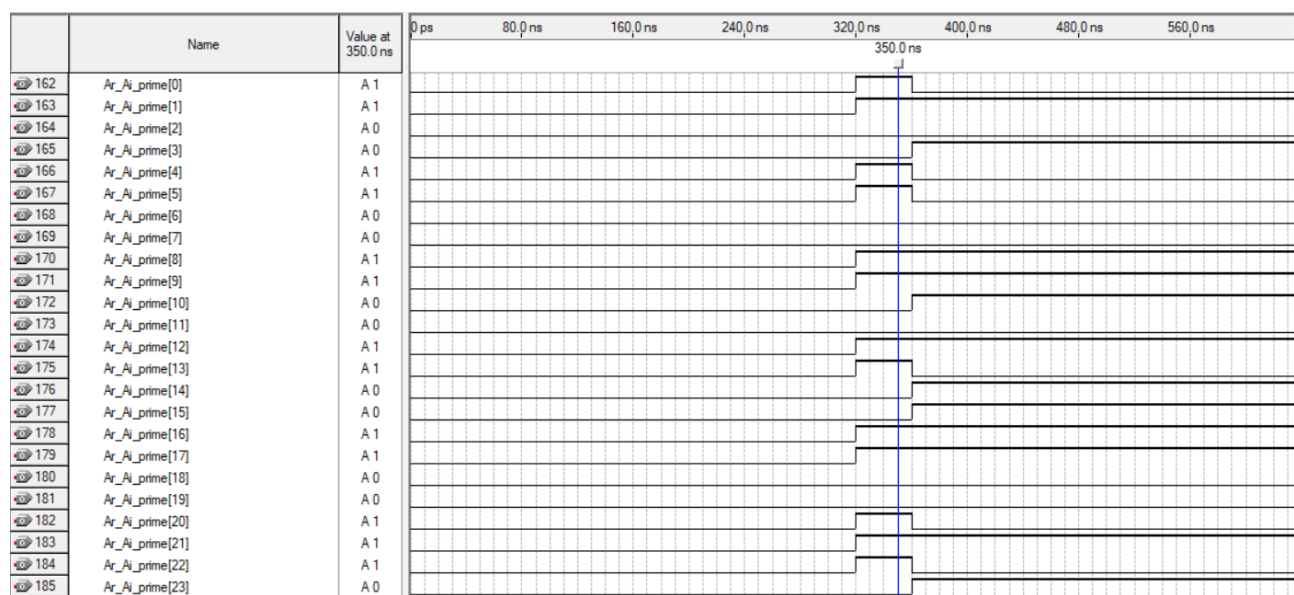
Gli ingressi provati sono questi: Ar, Ai, Br, Bi, Wr, Wi.

Name	Value at 370.0 ns	Name	Value at 370.0 ns
Ar[0]	A 0	Ar[0]	A 1
Ar[1]	A 0	Ar[1]	A 1
Ar[2]	A 1	Ar[2]	A 0
Ar[3]	A 1	Ar[3]	A 0
Ar[4]	A 0	Ar[4]	A 1
Ar[5]	A 0	Ar[5]	A 1
Ar[6]	A 1	Ar[6]	A 0
Ar[7]	A 1	Ar[7]	A 0
Ar[8]	A 0	Ar[8]	A 1
Ar[9]	A 0	Ar[9]	A 1
Ar[10]	A 1	Ar[10]	A 0
Ar[11]	A 1	Ar[11]	A 0
Ar[12]	A 0	Ar[12]	A 1
Ar[13]	A 0	Ar[13]	A 1
Ar[14]	A 1	Ar[14]	A 0
Ar[15]	A 1	Ar[15]	A 0
Ar[16]	A 0	Ar[16]	A 1
Ar[17]	A 0	Ar[17]	A 1
Ar[18]	A 1	Ar[18]	A 0
Ar[19]	A 1	Ar[19]	A 0
Ar[20]	A 0	Ar[20]	A 1
Ar[21]	A 0	Ar[21]	A 1
Ar[22]	A 0	Ar[22]	A 1
Ar[23]	A 0	Ar[23]	A 0

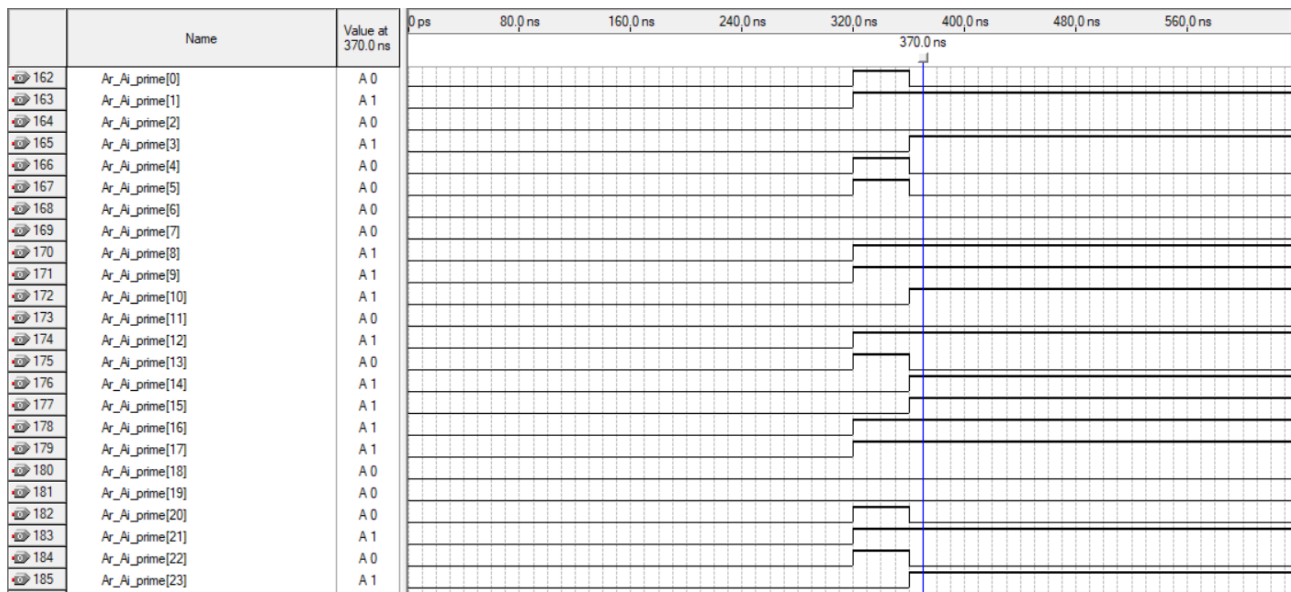
Name	Value at 370.0 ns	Name	Value at 370.0 ns
Br[0]	A 0	Br[0]	A 1
Br[1]	A 0	Br[1]	A 0
Br[2]	A 1	Br[2]	A 1
Br[3]	A 0	Br[3]	A 1
Br[4]	A 1	Br[4]	A 0
Br[5]	A 1	Br[5]	A 0
Br[6]	A 0	Br[6]	A 1
Br[7]	A 0	Br[7]	A 1
Br[8]	A 1	Br[8]	A 0
Br[9]	A 1	Br[9]	A 0
Br[10]	A 0	Br[10]	A 1
Br[11]	A 0	Br[11]	A 1
Br[12]	A 1	Br[12]	A 0
Br[13]	A 1	Br[13]	A 0
Br[14]	A 0	Br[14]	A 1
Br[15]	A 0	Br[15]	A 1
Br[16]	A 1	Br[16]	A 0
Br[17]	A 1	Br[17]	A 0
Br[18]	A 0	Br[18]	A 1
Br[19]	A 0	Br[19]	A 1
Br[20]	A 1	Br[20]	A 0
Br[21]	A 1	Br[21]	A 0
Br[22]	A 1	Br[22]	A 0
Br[23]	A 1	Br[23]	A 1

Name	Value at 370.0 ns	Name	Value at 370.0 ns
Wi[0]	A 1	Wr[0]	A 0
Wi[1]	A 1	Wr[1]	A 0
Wi[2]	A 0	Wr[2]	A 1
Wi[3]	A 0	Wr[3]	A 1
Wi[4]	A 1	Wr[4]	A 0
Wi[5]	A 1	Wr[5]	A 0
Wi[6]	A 0	Wr[6]	A 1
Wi[7]	A 0	Wr[7]	A 1
Wi[8]	A 1	Wr[8]	A 0
Wi[9]	A 1	Wr[9]	A 0
Wi[10]	A 0	Wr[10]	A 1
Wi[11]	A 0	Wr[11]	A 1
Wi[12]	A 1	Wr[12]	A 0
Wi[13]	A 1	Wr[13]	A 0
Wi[14]	A 0	Wr[14]	A 1
Wi[15]	A 0	Wr[15]	A 1
Wi[16]	A 1	Wr[16]	A 0
Wi[17]	A 1	Wr[17]	A 0
Wi[18]	A 0	Wr[18]	A 1
Wi[19]	A 0	Wr[19]	A 1
Wi[20]	A 1	Wr[20]	A 0
Wi[21]	A 1	Wr[21]	A 0
Wi[22]	A 1	Wr[22]	A 0
Wi[23]	A 0	Wr[23]	A 0

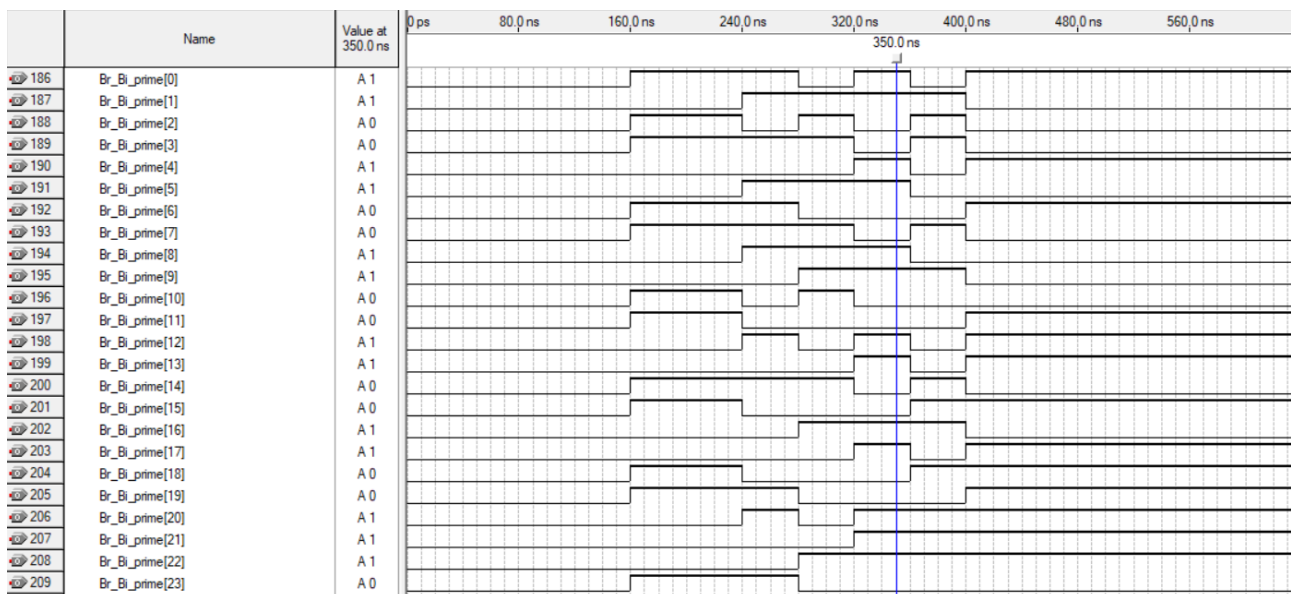
Il risultato dei dati conferma i dati simulati in MATLAB.



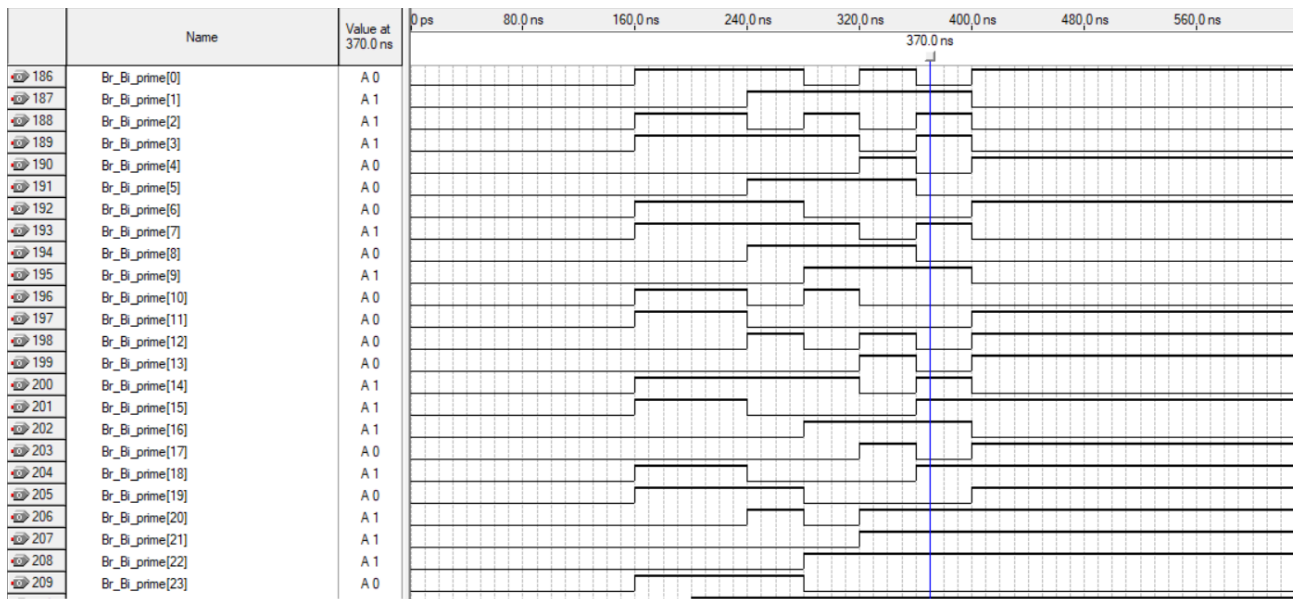
Risultato della simulazione sulla variabile in uscita A'r (o Ar_prime)



Risultato della simulazione sulla variabile in uscita A'i (o Ai_prime)



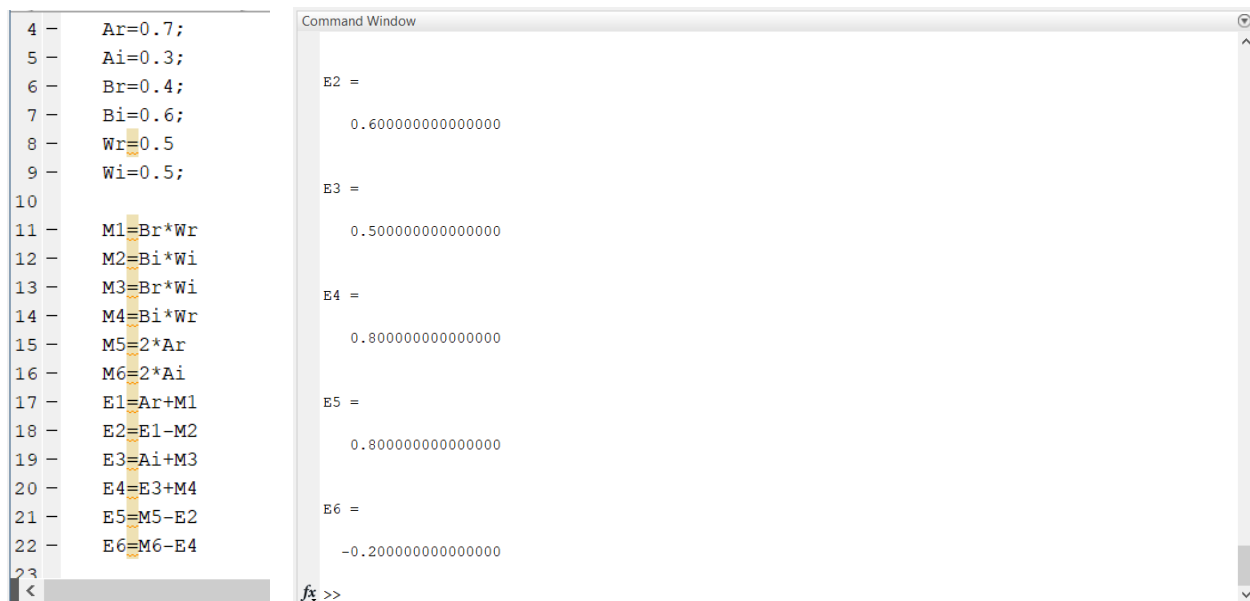
Risultato della simulazione sulla variabile in uscita B'r (o Br_prime)



Risultato della simulazione sulla variabile in uscita B'i (o Bi_prime)

Caso di una esecuzione continua

Anche qui il vettore di test è stato prima simulato usando MATLAB.

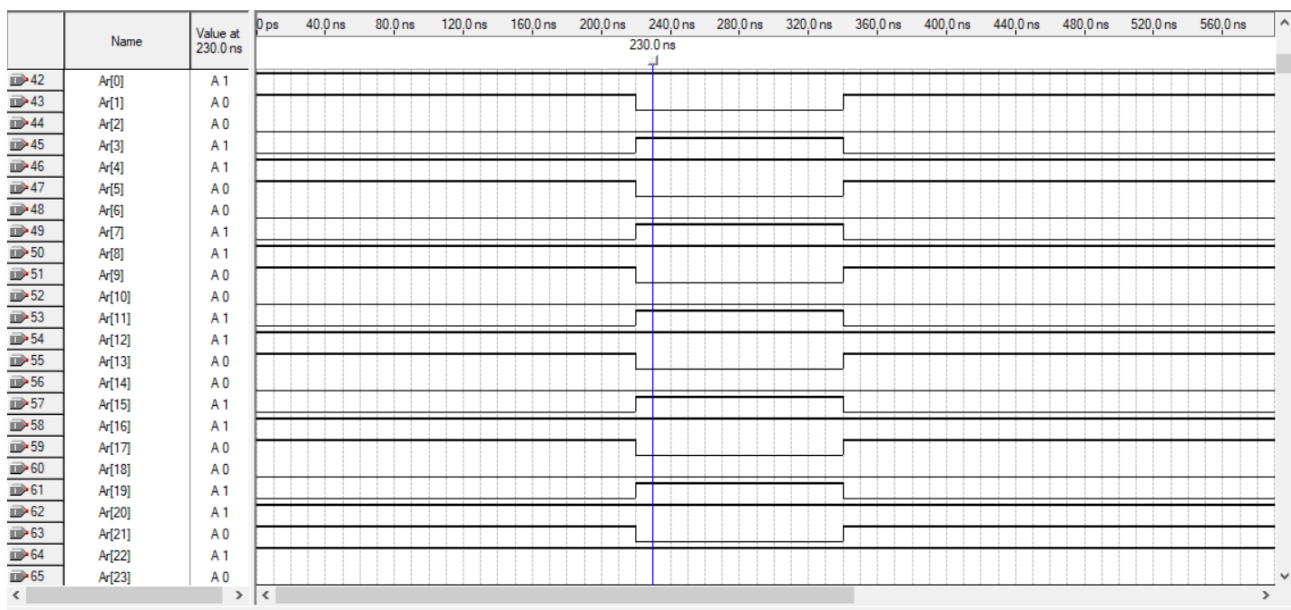


Risultato della simulazione su MATLAB

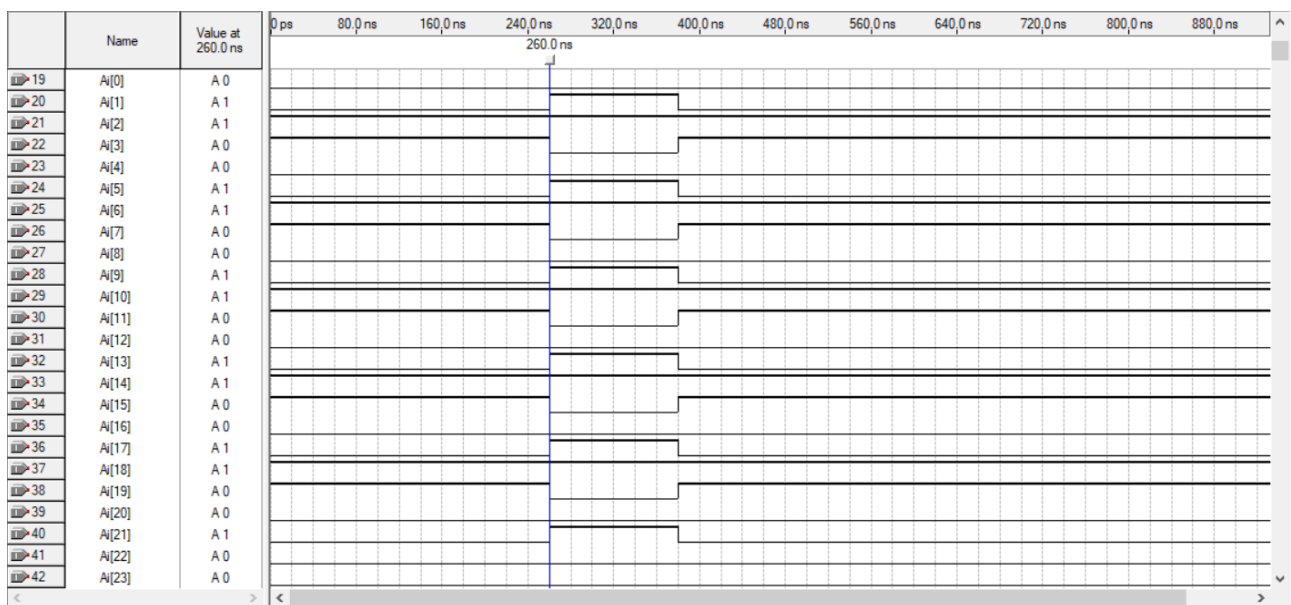
Si è simulato il caso di esecuzione continua. Alla butterfly, avendo inviato gli stessi dati del caso isolato, dopo 5 colpi di clock viene inviato un nuovo segnale di START (nel caso reale inviato dalla butterfly precedente). La butterfly campionerà i nuovi segnali in ingresso e fornirà il nuovo risultato ogni 5 colpi di clock.

La macchina viene inizializzata mettendo a 0 il RESETn.

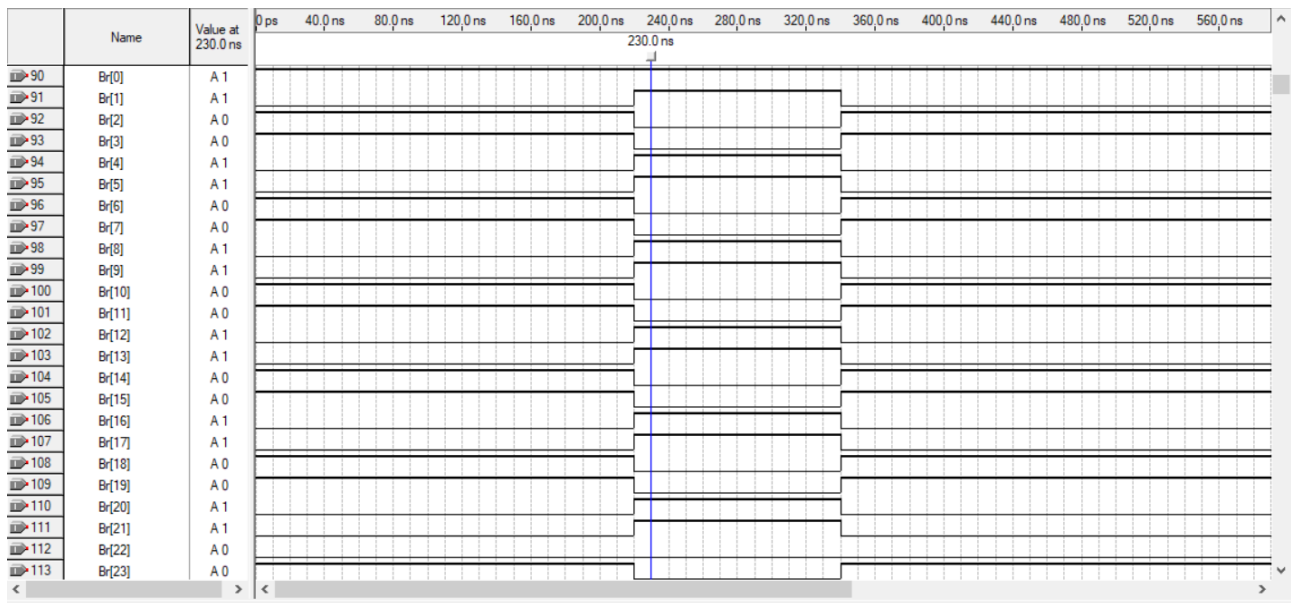
Gli ingressi provati sulla butterfly, al secondo START, sono questi: Ar, Ai, Br, Bi, Wr, Wi.



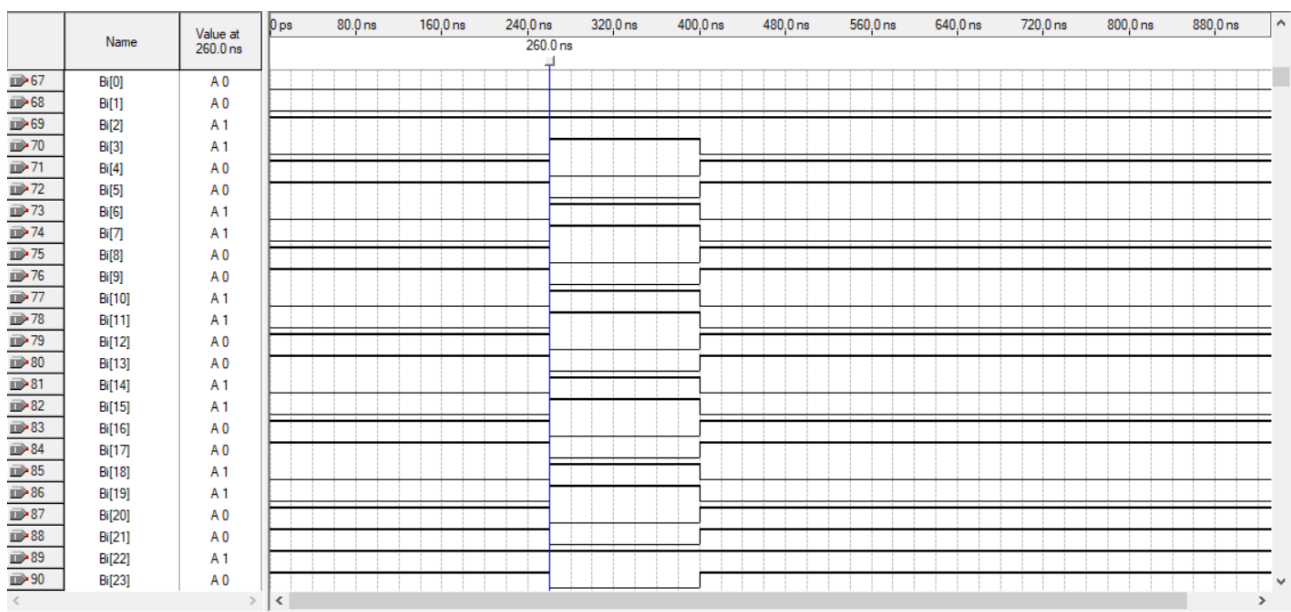
Segnale Ar in ingresso alla butterfly



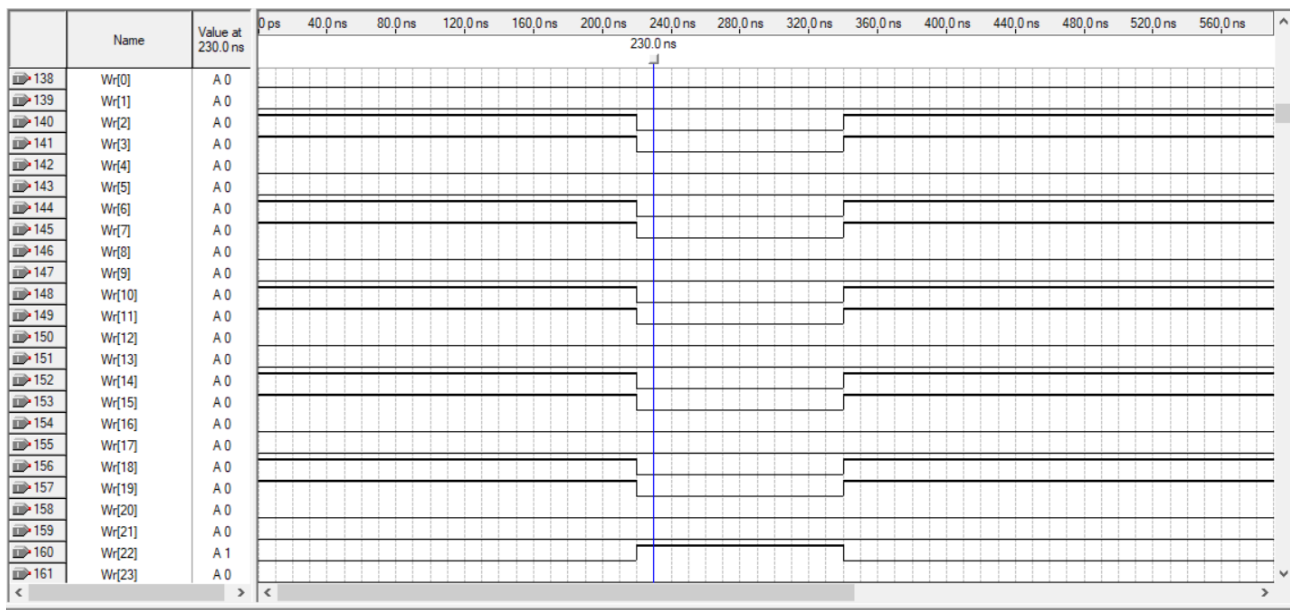
Segnale Ai in ingresso alla butterfly



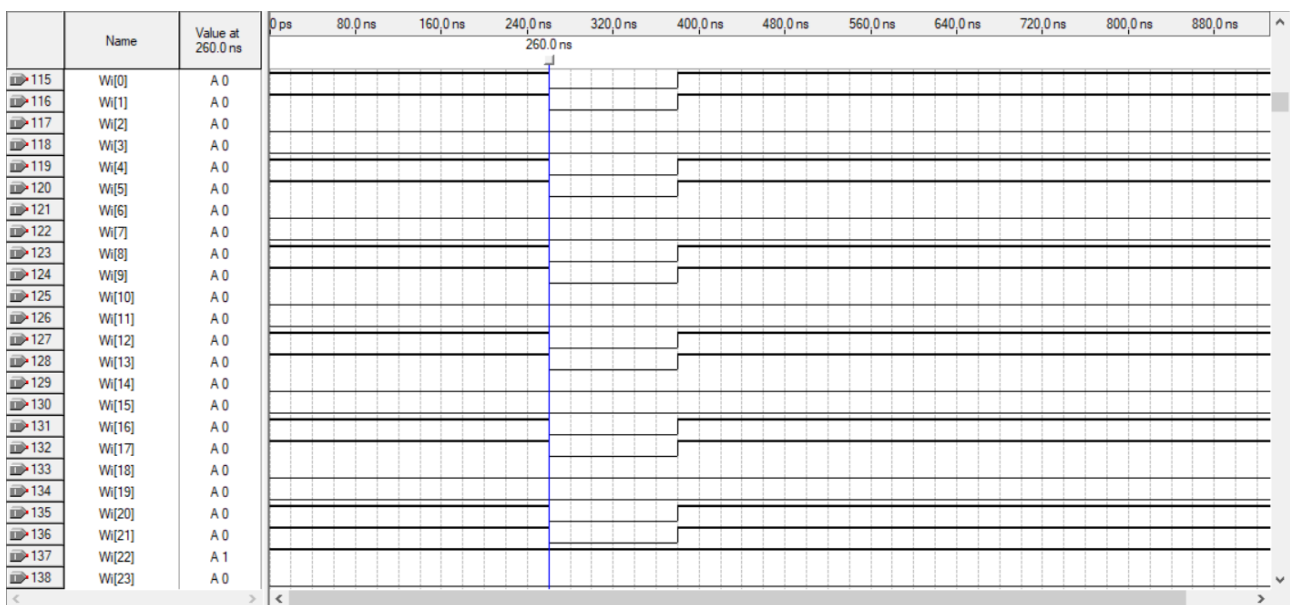
Segnale Br in ingresso alla butterfly



Segnale Bi in ingresso alla butterfly

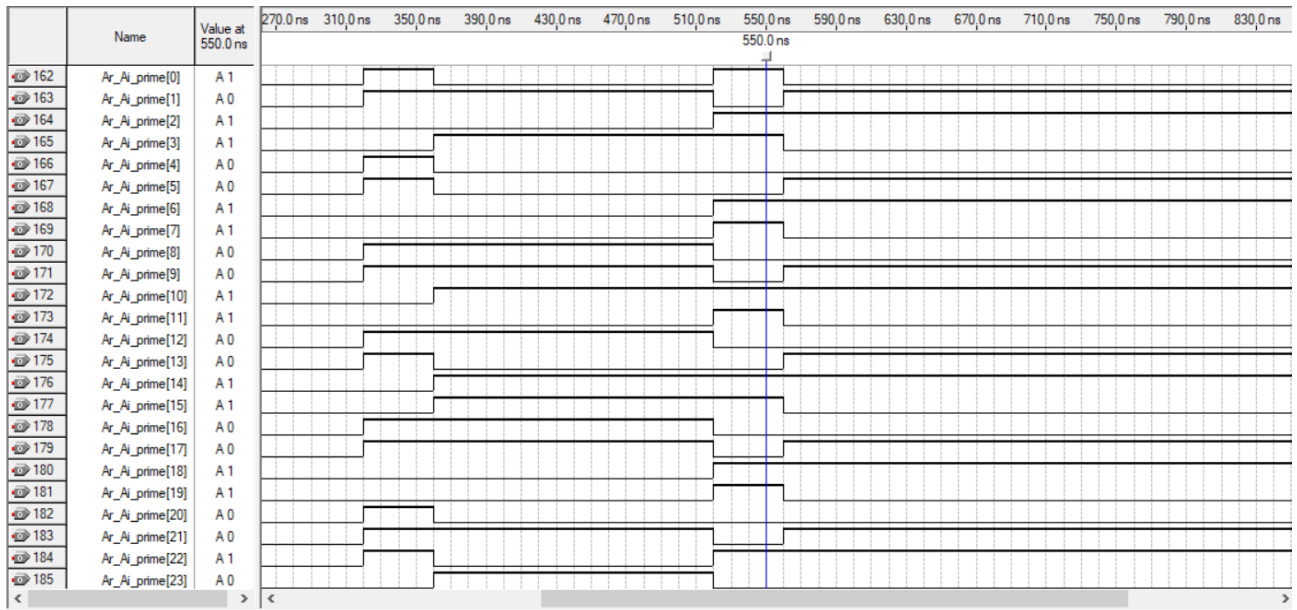


Segnale Wr in ingresso alla butterfly

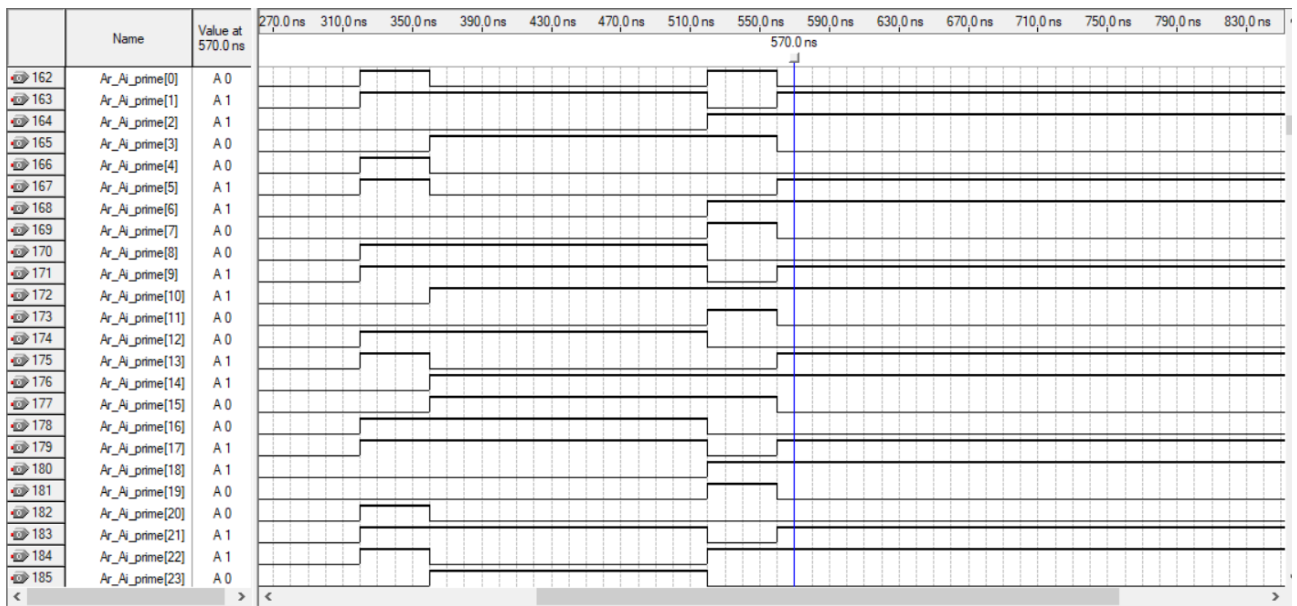


Segnale Wi in ingresso alla butterfly

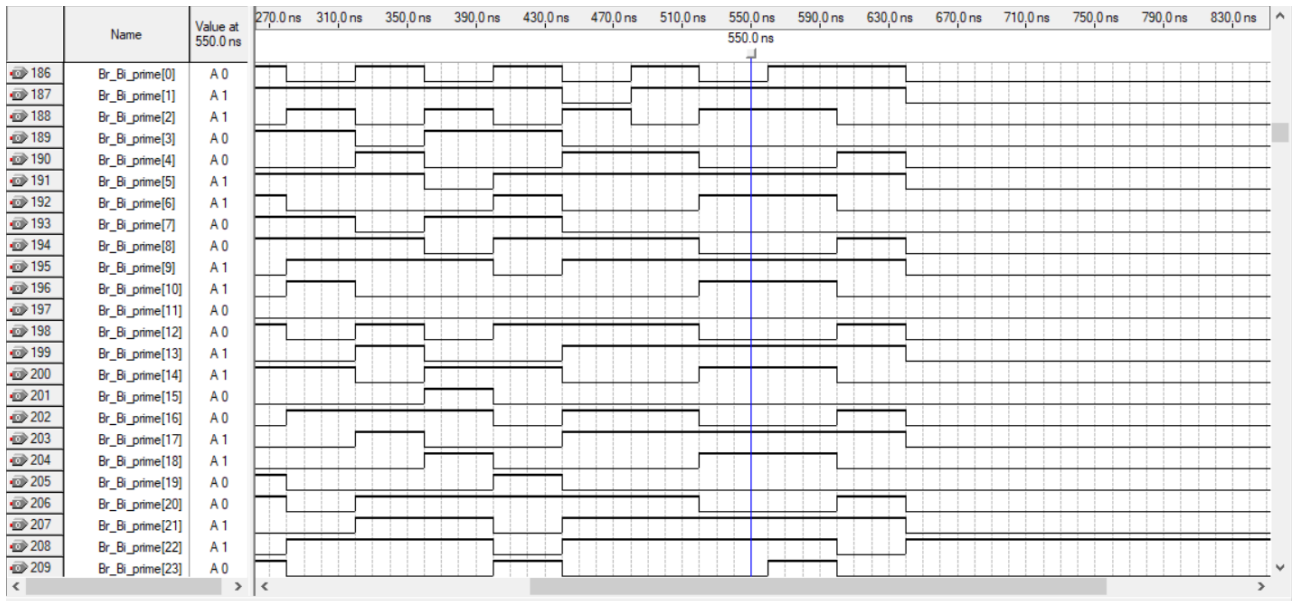
I risultati dell'esecuzione sulla butterfly.



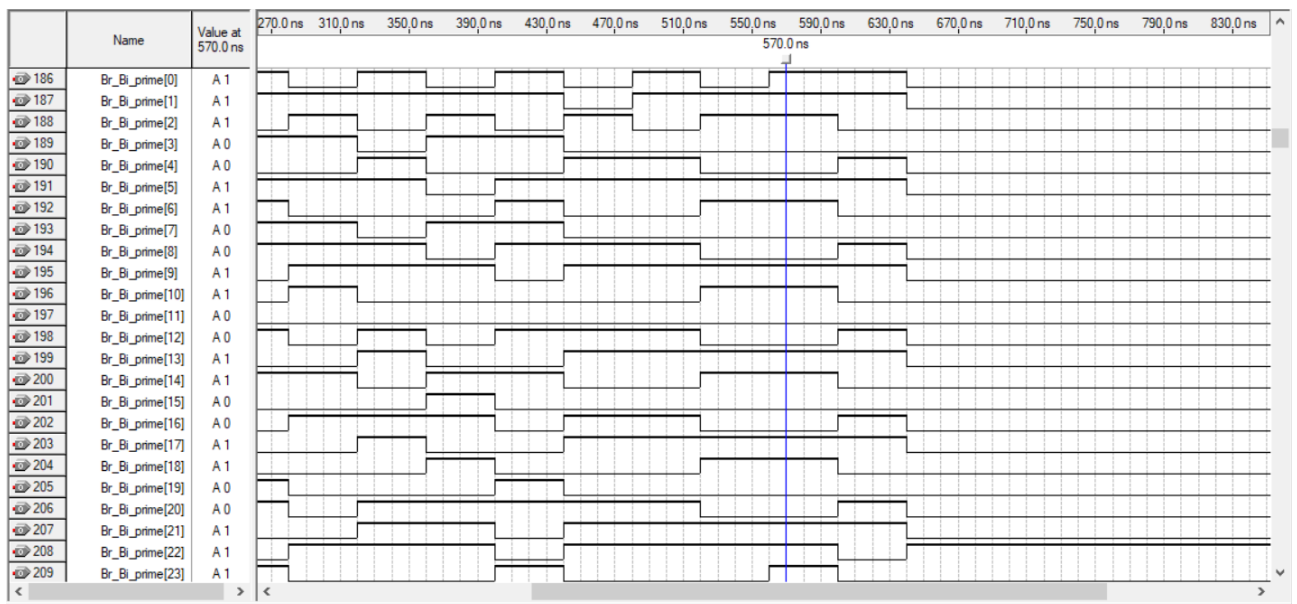
Segnale in uscita A'r (o Ar_prime) dalla butterfly



Segnale in uscita A'i (o Ai_prime) dalla butterfly



Segnale in uscita B'r (o Br_prime) dalla butterfly



Segnale in uscita B'i (o Bi_prime) dalla butterfly

Codice VHDL

File VHDL principale: butterfly.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity Butterfly is
port
    ( clk      : in std_logic;
      Start    : in std_logic;
      Resetn   : in std_logic;
      Done     : out std_logic;
      Ar       : in signed(23 downto 0);
      Ai       : in signed (23 downto 0);
      Br       : in signed (23 downto 0);
      Bi       : in signed (23 downto 0);
      Wr       : in signed(23 downto 0);
      Wi       : in signed (23 downto 0);
      Ar_Ai_prime : out signed(23 downto 0);
      Br_Bi_prime : out signed (23 downto 0)
    );

end Butterfly;

architecture behavior of Butterfly is

    signal microcontrol_address: std_logic_vector(17 downto 0);

    COMPONENT butterfly_control_unit IS

    PORT (

        START, clock : IN STD_LOGIC;                                --STATUS
        RESETN : IN STD_LOGIC;
        microcontrol_address : out std_logic_vector (17 downto 0)
        --segnali di controllo mandati al datapath
    );

    end COMPONENT;

    COMPONENT Butterfly_Data_Path is
        port (
            clk                : in std_logic;
            Ar                  : in signed (23 downto 0);
            Ai                  : in signed (23 downto 0);
```



```

        Br                : in signed (23 downto 0);
        Bi                : in signed (23 downto 0);
        Wr                : in signed (23 downto 0);
        Wi                : in signed (23 downto 0);
        load_ar           : in std_logic;
        Load_ai           : in std_logic;
        Load_br           : in std_logic;
        Load_bi           : in std_logic;
        Load_wr           : in std_logic;
        Load_wi           : in std_logic;
        Load_S5           : in std_logic;
        Load_S6           : in std_logic;
        sel_alpha         : in std_logic;
        sel_beta          : in std_logic;
        Load_sum1_3       : in std_logic;
        sel_W             : in std_logic;
        sel_b             : in std_logic;
        sel_a_Sft         : in std_logic;
        sel_a_sum         : in std_logic;
        sel_sub_alpha     : in std_logic;
        Load_Ar_ai_prime  : in std_logic;
        Ar_Ai_prime       : out signed (23 downto 0);
        Br_Bi_prime       : out signed (23 downto 0)
    );

```

end COMPONENT;

-DATAPATH top-level

BEGIN

Data_path: Butterfly_Data_Path

Port map

(clk,Ar,Ai,Br,Bi,Wr,Wi,microcontrol_address(17),microcontrol_address(15),microcontrol_address(14),

microcontrol_address(13),microcontrol_address(12),microcontrol_address(11),microcontrol_address(10),

microcontrol_address(9),microcontrol_address(8),microcontrol_address(7),microcontrol_address(6),

microcontrol_address(5),microcontrol_address(4),microcontrol_address(3),microcontrol_address(2),

microcontrol_address(1),microcontrol_address(0) ,Ar_Ai_prime,Br_Bi_prime);

Control_Unit: Butterfly_control_unit

Port map (start,clk,Resetn,microcontrol_address);

Done<=microcontrol_address(16);

end behavior;

File VHDL del datapath: butterfly_data_path.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Butterfly_Data_Path is
    port
    (
        clk                : in std_logic;
        Ar                  : in signed (23 downto 0);
        Ai                  : in signed (23 downto 0);
        Br                  : in signed (23 downto 0);
        Bi                  : in signed (23 downto 0);
        Wr                  : in signed (23 downto 0);
        Wi                  : in signed (23 downto 0);
        load_ar              : in std_logic;
        Load_ai              : in std_logic;
        Load_br              : in std_logic;
        Load_bi              : in std_logic;
        Load_wr              : in std_logic;
        Load_wi              : in std_logic;
        Load_S5              : in std_logic;
        Load_S6              : in std_logic;
        sel_alpha            : in std_logic;
        sel_beta             : in std_logic;
        Load_sum1_3          : in std_logic;
        sel_W                : in std_logic;
        sel_b                : in std_logic;
        sel_a_Sft            : in std_logic;
        sel_a_sum            : in std_logic;
        sel_sub_alpha        : in std_logic;
        Load_Ar_ai_prime     : in std_logic;
        Ar_Ai_prime          : out signed (23 downto 0);
        Br_Bi_prime          : out signed (23 downto 0);
    );

end Butterfly_Data_Path;

architecture rtl of Butterfly_data_Path is
    signal mpy_Br, mpy_Bi, mpy_Wr, mpy_Wi, mpy_Br_Bi, mpy_Wr_Wi, mux_Sft_Ar, mux_Sft_Ai,
    out_mux_A_Sft, out_mux_A_sum, out_trunc_Alpha: signed(23 downto 0);
    signal out_Sft: signed(24 downto 0);
    signal in_sum_ar_ai, in_sum_alpha_b, in_sum_alpha_a, in_sum_beta_a, in_sum_beta_b, out_sum1_3,
    in_reg_s5, in_reg_s6, in_sum_s5, in_sum_s6, out_sum_alpha, in_sum_b, in_sum_a, out_sum_beta: signed(49
    downto 0);
    signal no_trunc_alpha, no_trunc_beta, out_mpy : signed(46 downto 0);

    component Reg_N_bit is
```

```

generic
(
    DATA_WIDTH : natural := 8
);

port
(
    Clk          : in std_logic;
    resetn       : in std_logic;
    pr_in        : in signed(DATA_WIDTH-1 DOWNT0 0);
    load_parallel : in std_logic;
    pr_out       : out signed(DATA_WIDTH-1 downto 0)
);

end component;

```

component Adder is

```

generic
(
    DATA_WIDTH : natural := 50
);

port
(
    clk   : in std_logic;
    resetn : in std_logic;
    a      : in signed ((DATA_WIDTH-1) downto 0);
    b      : in signed ((DATA_WIDTH-1) downto 0);
    add_sub : in std_logic;
    result  : out signed ((DATA_WIDTH-1) downto 0)
);

```

end component;

component Multiplier is

```

generic
(
    DATA_WIDTH : natural := 24
);

port
(
    clk   : in std_logic;
    resetn : in std_logic;
    a      : in signed ((DATA_WIDTH-1) downto 0);
    b      : in signed ((DATA_WIDTH-1) downto 0);
    result : out signed (((2*DATA_WIDTH-1)-1) downto 0)
);

```

end component;

component Trunc is

```
generic
(
    DATA_WIDTH : natural := 50
);
port
(
    clk    : in std_logic;
    resetn : in std_logic;
    a      : in signed ((DATA_WIDTH-1) downto 0);
    result : out signed ((23) downto 0)
);
```

end component;

component Mux2to1 is

```
generic
(
    DATA_WIDTH : natural := 50
);
port
(
    a      : in signed ((DATA_WIDTH-1) downto 0);
    b      : in signed ((DATA_WIDTH-1) downto 0);
    s      : in std_logic;
    result : out signed ((DATA_WIDTH-1) downto 0)
);
```

end component;

-DATAPATH

begin

-RegFile

```
RegAr: Reg_N_bit generic map(24)
    port map (clk,'1',Ar,Load_Ar,Mux_sft_Ar);
RegAi: Reg_N_bit generic map(24)
    port map (clk,'1',Ai,Load_Ai,Mux_sft_Ai);
RegBr: Reg_N_bit generic map(24)
    port map (clk,'1',Br,Load_Br,Mpy_Br);
RegBi: Reg_N_bit generic map(24)
    port map (clk,'1',Bi,Load_Bi,Mpy_Bi);
RegWr: Reg_N_bit generic map(24)
    port map (clk,'1',Wr,Load_Wr,Mpy_Wr);
RegWi: Reg_N_bit generic map(24)
    port map (clk,'1',Wi,Load_Wi,Mpy_Wi);
```

--Mpy

Mpy: Multiplier generic map(24)

```
port map( clk,'1',Mpy_Br_Bi,Mpy_Wr_Wi,Out_Mpy);
```

--Mux

MuxB: Mux2to1 Generic Map(24)

```
Port map ( Mpy_Br,Mpy_Bi,sel_B,Mpy_Br_Bi);
```

MuxW: Mux2to1 Generic Map(24)

```
Port map ( Mpy_Wr,Mpy_Wi,sel_W,Mpy_Wr_Wi);
```

Mux_A_Sft: Mux2to1 Generic Map(24)

```
Port map ( Mux_Sft_Ar, Mux_Sft_Ai,sel_A_Sft,out_Mux_A_Sft);
```

Mux_A_Sum: Mux2to1 Generic Map(24)

```
Port map ( Mux_Sft_Ar, Mux_Sft_Ai,sel_A_Sum,out_Mux_A_Sum);
```

Mux_Alpha: Mux2to1 Generic Map(50)

```
Port map ( in_Sum_Ar_Ai, Out_Sum1_3,sel_Alpha,In_Sum_Alpha_A);
```

Mux_Beta: Mux2to1 Generic Map(50)

```
Port map ( in_Sum_S5, in_Sum_S6,sel_Beta,In_Sum_Beta_A);
```

```
in_Sum_Alpha_B<= out_mpy(46)& out_mpy(46) & out_mpy(46) & out_mpy;
```

```
out_Sft<=out_mux_A_Sft & '0';
```

```
in_Sum_Ar_Ai<=out_Mux_A_Sum(23) & out_Mux_A_Sum(23) & out_Mux_A_Sum(23) &
```

```
out_Mux_A_Sum & "000000000000000000000000";
```

```
in_Reg_S5<= out_Sft(24) & out_Sft(24) & out_Sft & "000000000000000000000000";
```

```
in_Reg_S6<= out_Sft(24) & out_Sft(24) & out_Sft & "000000000000000000000000";
```

--Reg

Reg_S5: Reg_N_bit generic map(50)

```
port map( clk,'1',in_reg_S5,Load_S5,in_Sum_S5);
```

Reg_S6: Reg_N_bit generic map(50)

```
port map( clk,'1',in_reg_S6,Load_S6,in_Sum_S6);
```

Reg_Sum1_3: Reg_N_bit generic map(50)

```
port map( clk,'1',out_Sum_Alpha,Load_Sum1_3,out_Sum1_3);
```

Reg_trunc_alpha:Reg_N_bit generic map(24)

```
port map( clk,'1',out_trunc_alpha,Load_Ar_Ai_prime,Ar_Ai_prime);
```

```
In_Sum_beta_B<=Out_Sum_Alpha;
```

--Sum

Sum_Alpha: Adder generic map(50)

```
port map (clk,'1',in_sum_Alpha_A,in_Sum_alpha_B,Sel_Sub_alpha,Out_Sum_Alpha);
```

Sum_Beta: Adder generic map(50)

```
port map (clk,'1',In_Sum_beta_A,in_Sum_Beta_B,'1',Out_Sum_Beta);
```

```
No_Trunc_alpha<=out_Sum_Alpha(46 downto 0); --scale factor
```

```
No_Trunc_beta<=out_Sum_Beta(46 downto 0); --scale factor
```

--Truncation

```

Trunc_Alpha: Trunc Generic Map (47)
    Port Map ( clk,'1',no_trunc_alpha,out_trunc_alpha);
Trunc_Beta: Trunc Generic Map (47)
    Port Map ( clk,'1',no_trunc_beta,Br_bi_prime);

end rtl;

```

File VHDL della control unit: butterfly_control_unit.vhd

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY butterfly_control_unit IS

PORT (
    START, clock : IN STD_LOGIC;                                --STATUS
    RESETN : IN STD_LOGIC;
    microcontrol_address : out std_logic_vector(17 downto 0)    --segnali di controllo mandati al datapath
);

end butterfly_control_unit;

ARCHITECTURE structure OF butterfly_control_unit is

component mux2to1_4bit IS

PORT (
    w0, w1 : IN STD_LOGIC_vector (3 downto 0) ;
    s : IN STD_LOGIC;
    f : OUT STD_LOGIC_vector( 3 downto 0)
);
END component;

component reg4 IS
PORT (
    D : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
    Resetn, Clock : IN STD_LOGIC ;
    Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
);

End component;

component Reg_N_bit is
generic
(
    DATA_WIDTH : natural := 50

```

```

);

port
(
    clk          : in std_logic;
    resetn       : in std_logic;
    pr_in        : in signed(DATA_WIDTH-1 DOWNT0 0);
    load_parallel : in std_logic;
    pr_out       : out signed(DATA_WIDTH-1 downto 0)
);

```

end component;

COMPONENT ROM_16X23 IS

```

PORT (
    ADDRESS : IN INTEGER RANGE 0 TO 15;
    DATA_OUT : OUT STD_LOGIC_VECTOR (22 DOWNT0 0)
);

```

END COMPONENT;

component adder_control IS

```

PORT (
    X : IN SIGNED(3 DOWNT0 0);
    S : OUT SIGNED (3 DOWNT0 0)
);

```

END component;

component status_pla is

```

port (
    start, CC : IN STD_LOGIC;
    ADDRESS_REG : IN STD_LOGIC_VECTOR ( 3 DOWNT0 0);
    SEL_STATUS_PLA : OUT STD_LOGIC
);

```

END component;

-SIGNALS

```

signal out_mux_address , address_rom_out : std_logic_vector ( 3 downto 0);
signal out_incrementatore_micro_address, jump_status : std_logic_vector(3 downto 0);
signal sel_mux : std_logic;
signal status_control_vector : std_logic_vector ( 22 downto 0);
signal address_rom : std_LOGIC_VECTOR ( 15 DOWNT0 0);
signal address_rom_integer : integer range 0 to 15 ;
signal out_micro_instruction,status_control_vector_signed: signed(22 downto 0);
signal address_rom_out_signed, out_incrementatore_micro_address_signed : signed ( 3 downto 0);

```

-DATAPATH della control unit

begin

```
sommatore : adder_control port map ( address_rom_out_signed , out_incrementatore_micro_address_signed
);
address_rom_out_signed <= signed(address_rom_out);
out_incrementatore_micro_address <= std_logic_vector(out_incrementatore_micro_address_signed);
micro_address_register : reg4 port map ( out_mux_address , resetn , clock, address_rom_out);
sel_micro_address_register : mux2to1_4bit port map ( out_incrementatore_micro_address, jump_status,
sel_mux, out_mux_address);
memory_ROM_16X23 : ROM_16X23 PORT MAP ( address_rom_integer , status_control_vector);
jump_status <= status_control_vector (21 downto 18);
status_pla_combinatoria : status_pla port map ( start , status_control_vector (22) , address_rom_out, sel_mux);

address_rom_integer <= to_integer(unsigned(address_rom_out));
microcontrol_address<=std_logic_vector(out_micro_instruction(17 downto 0));
status_control_vector_signed<=signed(status_control_vector);
micro_instruction_register: Reg_N_bit generic map(23)
    port map (clock,'1',status_control_vector_signed,'1',out_micro_instruction);

end structure;
```