

Progetto di Reti Logiche

Prof. Gianluca Palermo - Anno 2019/2020

Rigutti Luca [codice persona: 10558383]

Tortorelli Giuseppe [codice persona: 10582962]

Contents

1	Introduzione	2
1.1	Scopo del progetto	2
1.2	Specifiche generali	2
1.3	Interfaccia del componente	3
1.4	Dati e descrizione memoria	4
2	Design	5
2.1	Stati della macchina	6
2.1.1	IDLE: $i_rst = 0$	6
2.1.2	READ: $i_start = 1$ e $status = 0$	6
2.1.3	ENCODE: $i_start = 1$, $status = 1$ e $encode_status = 0$	6
2.1.4	WRITE: $i_start = 1$, $status = 1$ e $encode_status = 1$	6
2.1.5	DONE: $i_start = 1$ e $status = 2$	6
2.1.6	END: $i_start = 0$ e $status = 3$	6
3	Risultati dei test	7
4	Conclusione	10
4.1	Risultati della sintesi	10

1 Introduzione

1.1 Scopo del progetto

Il progetto di reti logiche dell'anno accademico 2019-2020 si basa sul metodo di codifica a bassa dissipazione di potenza detto "Working Zone". Il metodo "Working Zone" lavora sul *Bus* Indirizzi e si usa per codificare il valore di un indirizzo nel caso questo appartenga a certi intervalli noti: le *working-zone*. Ci possono essere multiple *working-zone*, ognuna delle quali parte da un indirizzo base e si estende per una dimensione fissa.

1.2 Specifiche generali

Vengono fornite otto *working-zone* e l'indirizzo da codificare. Ogni *working-zone* parte dall'indirizzo base e si estende per una dimensione complessiva di quattro indirizzi (incluso quello base).

Si possono presentare due casi:

1. Indirizzo non presente in nessuna *working-zone*

In questo caso l'indirizzo codificato da restituire in *output* è così formato:

WZ_BIT & ADDR

- **WZ_BIT**: è il *bit* che indica se l'indirizzo appartiene o meno a qualche *working-zone* ed in questo caso vale 0.
- **ADDR**: è l'indirizzo originale fornito in *input*.

2. Indirizzo presente in una *working-zone*

In questo caso l'indirizzo codificato da restituire in *output* è così formato:

WZ_BIT & WZ_NUM & WZ_OFFSET

- **WZ_BIT**: è il *bit* che indica se l'indirizzo appartiene o meno a qualche *working-zone* ed in questo caso vale 1.
- **WZ_NUM**: è il numero della *working-zone* a cui l'indirizzo appartiene.
- **WZ_OFFSET**: è l'*offset* tra l'indirizzo base della *working-zone* e l'indirizzo da codificare.

L'indirizzo da codificare è espresso su 7 *bit*, in modo tale da rappresentare tutti i valori che vanno da 0 a 127. Gli indirizzi base delle otto *working-zone* e l'indirizzo codificato sono espressi su 8 *bit*.

WZ_NUM è espresso su 3 *bit* per rappresentare gli otto indirizzi, quindi WZ_OFFSET su 4 *bit*.

In particolare WZ_OFFSET è codificato *one-hot* così come segue:

- $WZ_OFFSET = 0$ è codificato come 0001;
- $WZ_OFFSET = 1$ è codificato come 0010;
- $WZ_OFFSET = 2$ è codificato come 0100;
- $WZ_OFFSET = 3$ è codificato come 1000;

1.3 Interfaccia del componente

```
entity project_reti_logiche is
  port (
    i_clk      : in std_logic;
    i_start    : in std_logic;
    i_rst      : in std_logic;
    i_data     : in std_logic_vector(7 downto 0);
    o_address  : out std_logic_vector(15 downto 0);
    o_done     : out std_logic;
    o_en       : out std_logic;
    o_we       : out std_logic;
    o_data     : out std_logic_vector(7 downto 0)
  );
end project_reti_logiche;
```

- i_clk è il segnale di CLOCK;
- i_start è il segnale di START;
- i_rst è il segnale di RESET;
- i_data è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_address è il segnale di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione
- o_en è il segnale di ENABLE per abilitare le operazioni sulla memoria
- o_we è il segnale di WRITE ENABLE per abilitare la scrittura (o_en deve essere alto)
- o_data è il segnale di uscita che invia alla memoria l'indirizzo codificato

1.4 Dati e descrizione memoria

I dati, ciascuno di dimensione 8 *bit* (ADDR è esteso con uno 0 in posizione più significativa), sono memorizzati in una memoria RAM con indirizzamento al *byte*:

- Le celle di indirizzi dallo 0 al 7 contengono gli indirizzi base delle otto *working-zone*;
- La cella di indirizzo 8 contiene l'indirizzo da codificare;
- La cella di indirizzo 9 contiene l'indirizzo codificato che viene fornito in *output*;
- Le restanti celle sono inutilizzate;

WZ 0	Indirizzo 0
WZ 1	Indirizzo 1
WZ 2	Indirizzo 2
WZ 3	Indirizzo 3
WZ 4	Indirizzo 4
WZ 5	Indirizzo 5
WZ 6	Indirizzo 6
WZ 7	Indirizzo 7
ADDR	Indirizzo 8
OUTPUT	Indirizzo 9
...	
unused	Indirizzo n

Figure 1: schema della memoria

2 Design

L'esecuzione inizia con un segnale di `i_rst` posto a 1. Dopo l'abbassamento di `i_rst`, si attende che `i_start` diventi 1. Quest'ultimo rimarrà alto fintanto che il segnale `o_done` è basso. Quindi, dopo un ciclo di *clock* per portare a 1 il segnale `o_en`, si inizia con il prendere i dati dalla memoria.

Successivamente si abilita il segnale di scrittura (`o_we`) e si cerca la *working-zone* corrispondente all'indirizzo da codificare. A seconda che la *working-zone* venga trovata o meno, si scrive sul segnale `o_data` l'indirizzo codificato nella maniera opportuna. Conclusa questa fase, si porta il segnale `o_done` a 1 per indicare di aver finito con l'esecuzione e in modo tale da poter far scendere prima `i_start` e di conseguenza ancora `o_done`. Quindi la macchina si pone in attesa di un nuovo segnale di inizio o di *reset* con la differenza che nel primo caso si procede a leggere la memoria solo nella posizione corrispondente all'indirizzo da codificare.

L'implementazione è stata sviluppata tramite un'unica architettura di tipo *Behavioral*. Di seguito sono illustrati i vari segnali interni utilizzati:

```
signal wz0 : std_logic_vector(7 downto 0);
signal wz1 : std_logic_vector(7 downto 0);
signal wz2 : std_logic_vector(7 downto 0);
signal wz3 : std_logic_vector(7 downto 0);
signal wz4 : std_logic_vector(7 downto 0);
signal wz5 : std_logic_vector(7 downto 0);
signal wz6 : std_logic_vector(7 downto 0);
signal wz7 : std_logic_vector(7 downto 0);
signal addr : std_logic_vector(7 downto 0);
signal en_status : std_logic;
signal we_status : std_logic;
signal wz_found : std_logic;
signal encode_status : std_logic;
signal tmp_o_data : std_logic_vector( 7 downto 0);
signal mem_counter : integer;
signal status : integer;
```

- `wz0` : è utilizzato per memorizzare l'indirizzo base della prima *working-zone*;
- `wz1` : è utilizzato per memorizzare l'indirizzo base della seconda *working-zone*;
- `wz2` : è utilizzato per memorizzare l'indirizzo base della terza *working-zone*;
- `wz3` : è utilizzato per memorizzare l'indirizzo base della quarta *working-zone*;
- `wz4` : è utilizzato per memorizzare l'indirizzo base della quinta *working-zone*;
- `wz5` : è utilizzato per memorizzare l'indirizzo base della sensta *working-zone*;
- `wz6` : è utilizzato per memorizzare l'indirizzo base della settima *working-zone*;
- `wz7` : è utilizzato per memorizzare l'indirizzo base della ottava *working-zone*;
- `addr` : è utilizzato per memorizzare l'indirizzo da codificare;
- `en_status` : è utilizzato per controllare il valore di `o_en`;
- `wn_status` : è utilizzato per controllare il valore di `o_we`;
- `wz_found` : è utilizzato per controllare se l'indirizzo è stato trovato in una delle *working-zone*;
- `encode_status` : è utilizzato per controllare la fare si codifica;
- `tmp_o_data` : è utilizzato per memorizzare un valore temporaneo dell'indirizzo codificato;
- `mem_counter` : è utilizzato per realizzare il contatore che legge i valori dalla memoria;
- `status` : è utilizzato per distinguere le varie fasi di esecuzione della macchina;

2.1 Stati della macchina

Le principali fasi di esecuzione sono scandite dal segnale `status`. Di seguito la descrizione precisa degli stati più interessanti della macchina.

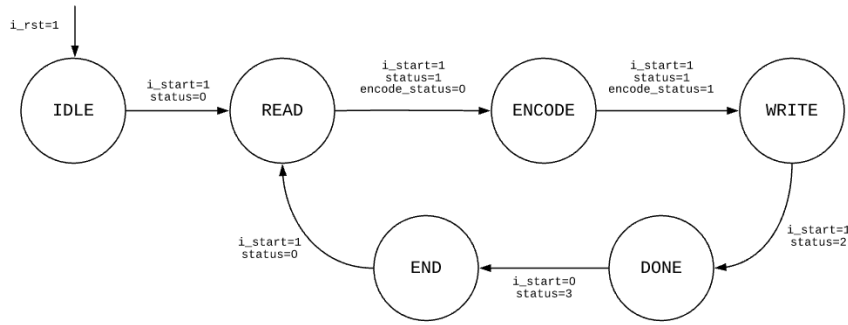


Figure 2: diagramma degli stati

2.1.1 IDLE: `i_rst = 0`

Lo stato si *reset* nel quale vengono inizializzati i segnali.

2.1.2 READ: `i_start = 1` e `status = 0`

Dopo un ciclo di *clock* utile per attivare la lettura tramite i segnali `en_status` e `o_en`, inizia il contatore che legge i dati dalla memoria: ogni due cicli di *clock* viene posto in `o_address` l'indirizzo della memoria che contiene il valore che si vuole leggere al ciclo successivo.

La scelta di usare due cicli per leggere i dati dalla memoria è stata presa al fine di evitare sfasamenti sulla lettura dei dati a causa di eventuali ritardi sul segnale `i_data`. Va precisato che se gli eventuali ritardi superano il periodo di *clock*, la soluzione adottata non risulta più efficace, ma dal momento che non è fornito nessun modo per verificare che il dato richiesto è stato effettivamente ricevuto, si è assunto che tali ritardi siano frutto di un funzionamento non contemplato dalla macchina.

2.1.3 ENCODE: `i_start = 1`, `status = 1` e `encode_status = 0`

Dopo un ciclo di *clock* utile per attivare la scrittura tramite i segnali `we_status` e `o_we`, inizia la fase di codifica. Viene confrontato l'indirizzo da codificare con ogni set di *working-zone* parallelamente e nel caso venga trovata una corrispondenza si scrive l'indirizzo codificato in `temp_o_data`. Il segnale `wz_found` serve per discriminare se la *working-zone* è stata trovata o meno.

2.1.4 WRITE: `i_start = 1`, `status = 1` e `encode_status = 1`

Questo è lo stato in cui viene scritto il risultato nella memoria. Grazie al segnale `wz_found` è possibile scrivere l'indirizzo codificato nella maniera opportuna.

2.1.5 DONE: `i_start = 1` e `status = 2`

Finita l'elaborazione, si settano i vari segnali ai valori opportuni e si alza il segnale di `o_done` per notificare che l'esecuzione è stata completata.

2.1.6 END: `i_start = 0` e `status = 3`

`i_start` è tornato a 0 quindi si riabbassa anche `o_done`.

I segnali `mem_counter` e `o_address` vengono settati in maniera tale da entrare nel ciclo di conteggio (stato READ) nel momento della lettura dell'indirizzo da codificare. Questo perchè gli indirizzi delle *working-zone* non cambiano tra un segnale di *start* e un'altro ma solamente quando viene resettata la macchina.

3 Risultati dei test

Per verificare il corretto funzionamento del componente sintetizzato, sono stati scritti alcuni *test bench*, al fine di testare il componente nei casi limite della macchina. Un fattore comune a tutti i *test bench* applicati è quello di aver sincronizzato il segnale di *i_data* prima sul fronte di salita del *clock* e successivamente su quello di discesa. L'obiettivo è stato quello di verificare che in entrambe le situazioni la lettura dei dati non risulta sfasata.

Di seguito sono descritti i test effettuati ognuno con due *screenshot* delle *waveform*: uno per *clock* su fronte di salita e uno per *clock* su fronte di discesa.

1. Due segnali di *start* consecutivi

Questo test è stato utilizzato per verificare che dopo il secondo segnale di *start* il componente legga solo la cella della memoria contenente l'indirizzo e che quindi i restanti segnali non vengano compromessi.

L'indirizzo corrispondente al primo segnale di *start* non è presente in nessuna *working-zone*, invece quello corrispondente al secondo segnale è presente nella terza *working-zone*.

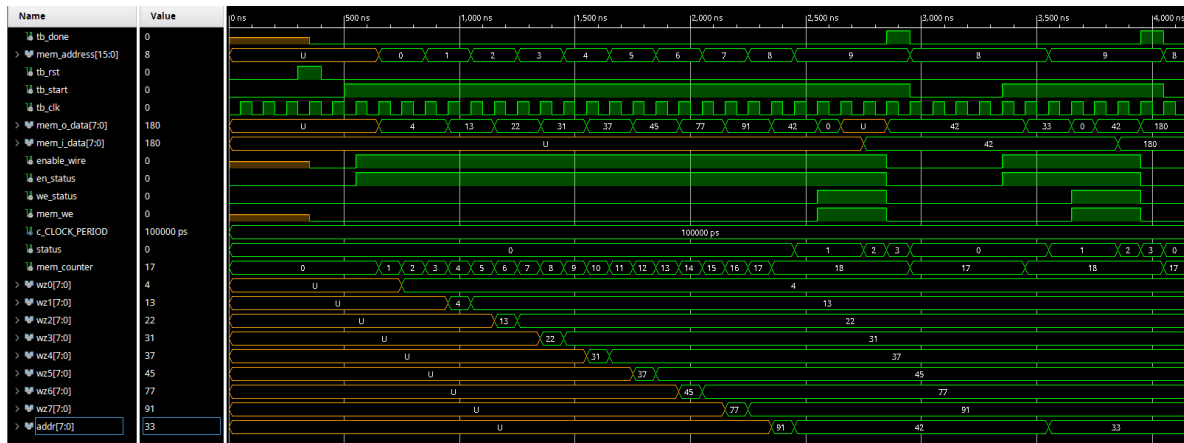


Figure 3: *waveform* con segnale sincronizzato su fronte di salita

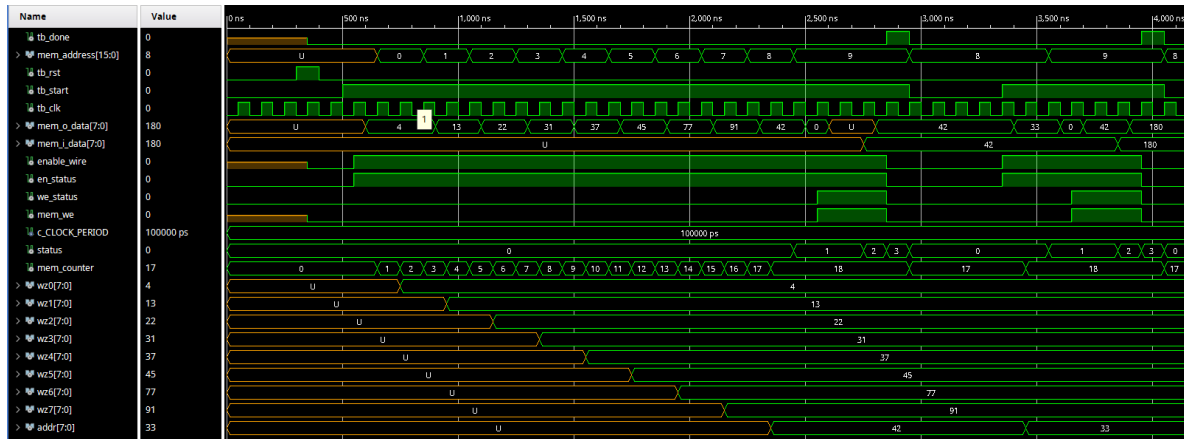


Figure 4: *waveform* con segnale sincronizzato su fronte di discesa

2. Un segnale di *start* seguito da uno di *reset* seguito da uno di *start*

Questo test è stato utilizzato per verificare che il componente resettasse in maniera corretta i segnali e che rilegga tutti i nuovi valori dalla memoria.

Durante la prima esecuzione l'indirizzo non è in nessuna *working-zone*, mentre dopo il segnale di reset l'indirizzo è presente nella quarta *working-zone*.

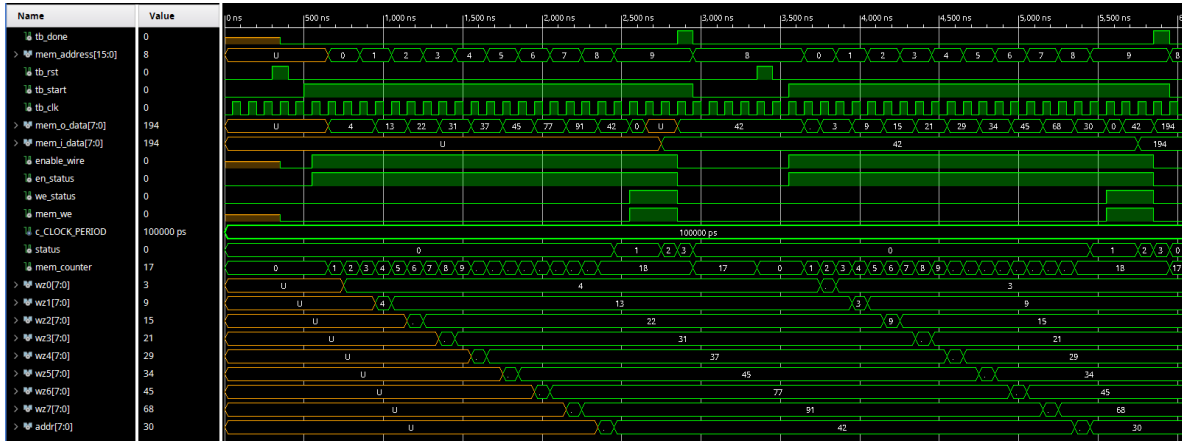


Figure 5: *waveform* con segnale sincronizzato su fronte di salita

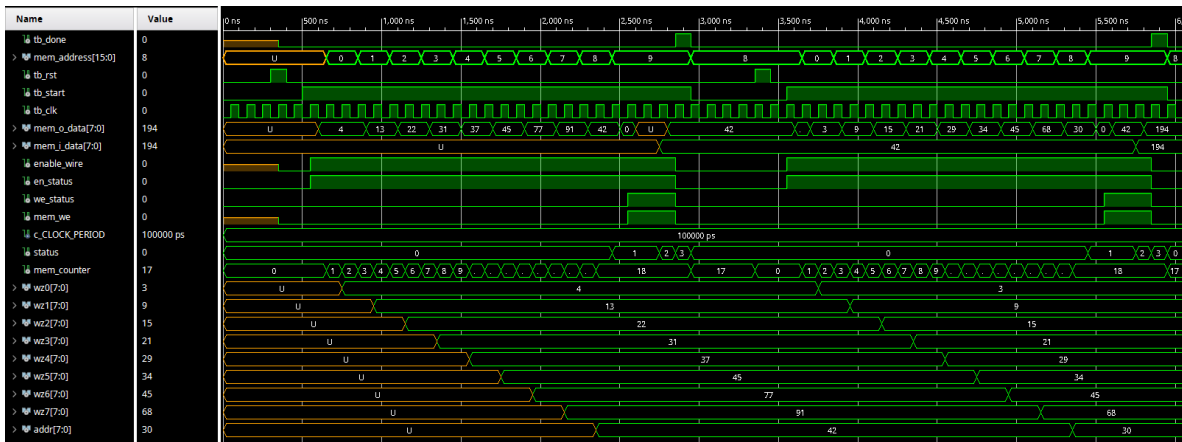


Figure 6: *waveform* con segnale sincronizzato su fronte di discesa

3. Un segnale di *reset* durante segnale di *start* alto

Questo test è stato utilizzato per verificare che un segnale di *reset* durante l'esecuzione della macchina, non comprometta l'esito positivo della computazione. Si è ipotizzato che appena viene lanciato il segnale di *reset* il segnale *i_start* scenda subito a 0 e che ritorni ad 1 dopo qualche ciclo di *clock* successivo all'abbassamento di *i_rst*.

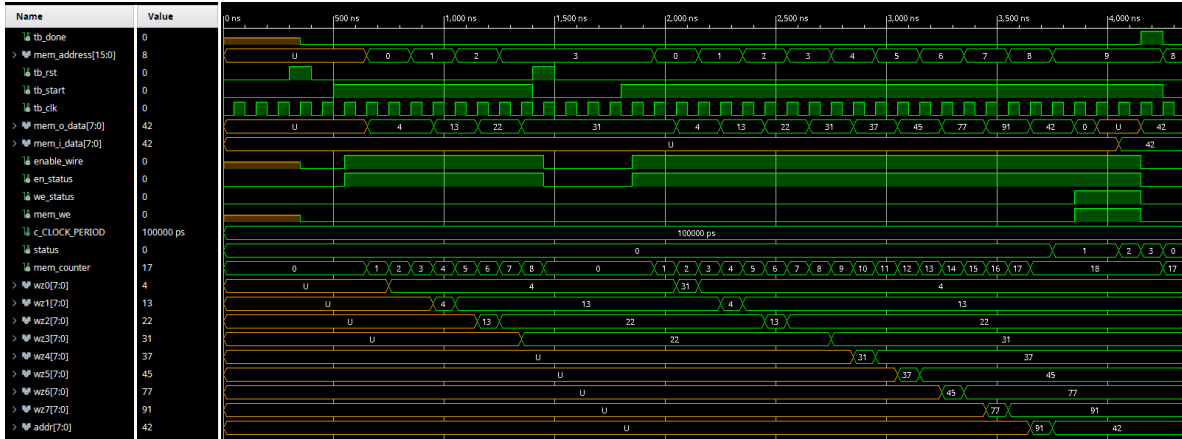


Figure 7: *waveform* con segnale sincronizzato su fronte di salita

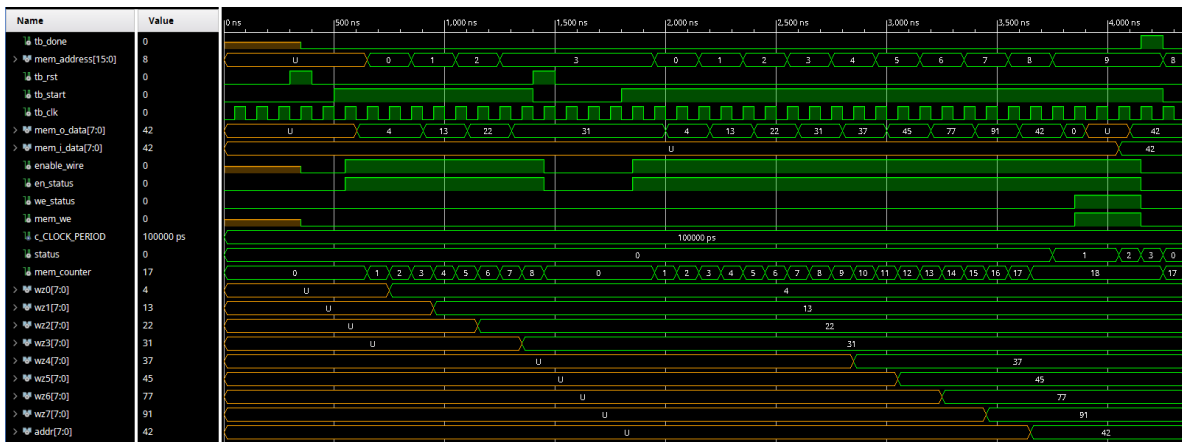


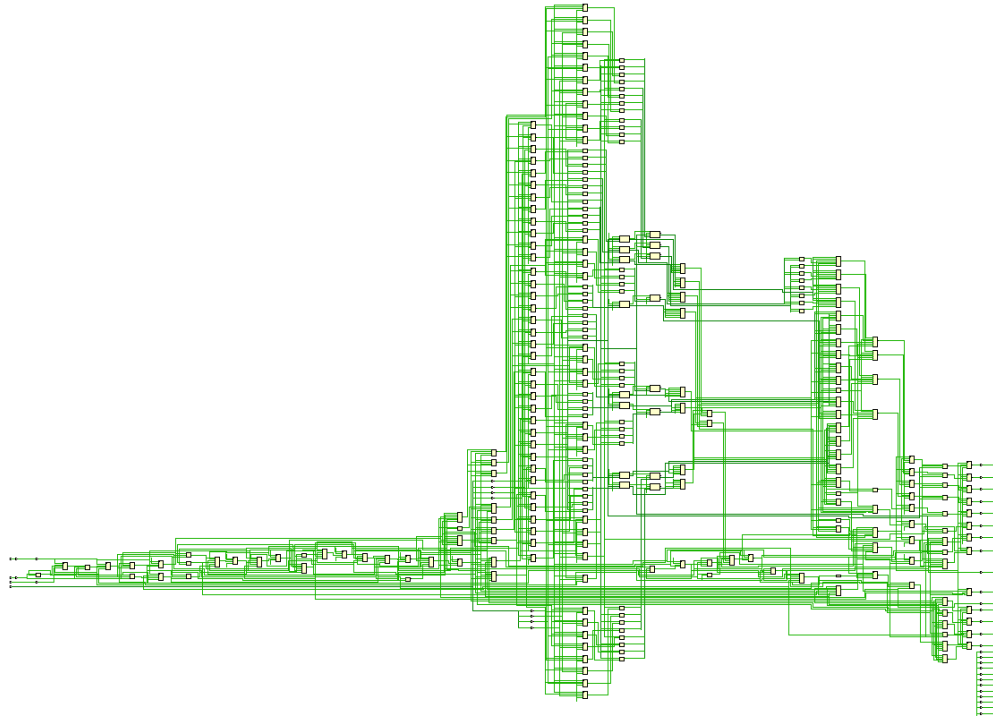
Figure 8: *waveform* con segnale sincronizzato su fronte di discesa

4 Conclusione

4.1 Risultati della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle tre simulazioni: *Behavioral*, *Post-Synthesis Functional* e *Post-Synthesis Timing*. Inoltre tutti i test restituiscono esito positivo sia in *Pre-Synthesis* che in *Post-Synthesis*, con un periodo di *clock* fino a 1[ns].

Di seguito lo schema del circuito sintetizzato.



Resource	Estimation	Available	Utilization %
LUT	143	134600	0.11
FF	105	269200	0.04
IO	38	285	13.33
BUFG	1	32	3.13

Figure 9: schema del circuito e tabella di utilizzo