

# **Prova Finale - Progetto di Reti Logiche**

**Prof. Fabio Salice**

*Anno Accademico 2022-23*

*Prof. Fabio Salice*



**POLITECNICO**  
**MILANO 1863**

Alessandro Griffanti: codice persona 10680747

Tommaso Ferrario: codice persona 10656892

# Indice

<b>1 INTRODUZIONE.....</b>	<b>3</b>
1.1 Scopo del progetto .....	3
1.2 Interfaccia del componente.....	4
1.3 Dati e Memoria.....	4
<b>2 ARCHITETTURA .....</b>	<b>5</b>
2.1 Segnali usati .....	5
2.2 Macchina a Stati.....	6
2.3 Schema funzionale .....	7
<b>3 RISULTATI SPERIMENTALI .....</b>	<b>8</b>
3.1 Report di sintesi.....	8
3.2 Simulazioni.....	8
<b>4 CONCLUSIONI .....</b>	<b>10</b>

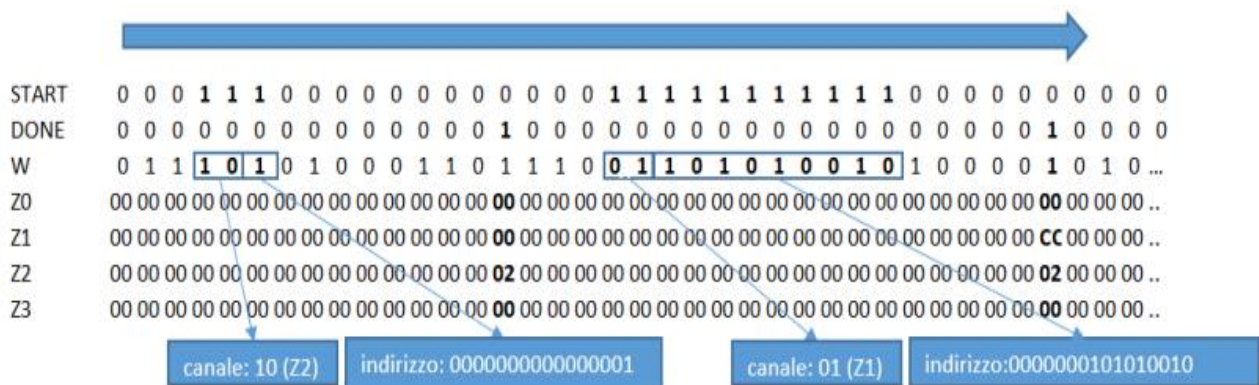
# 1 INTRODUZIONE

## 1.1 Scopo del progetto

La specifica del progetto prevede che il sistema da noi progettato riceva in ingresso indicazioni riguardo una specifica locazione di memoria e un canale di uscita dei quattro disponibili. Il contenuto presente in memoria all'indirizzo ricevuto viene quindi inviato verso il canale di uscita indicato dal segnale di ingresso.

L'ingresso che viene ricevuto è di tipo seriale da un bit e, leggendo il contenuto dell'ingresso sul fronte di salita del clock, si costituisce una sequenza di bit, con lunghezza di almeno due e massimo diciotto.

I primi due bit ricevuti identificano il canale di uscita tra i quattro disponibili, mentre i bit successivi specificano l'indirizzo di memoria in cui è memorizzato il dato richiesto.



Nell'esempio sopra riportato, si può notare come sono disposti i bit nella sequenza che si genera in ingresso e come vengono utilizzati correttamente: infatti, è indicato il canale di uscita del dato (da due bit di lunghezza) e l'indirizzo di memoria in cui è contenuto il dato (non obbligatoriamente da sedici bit, viene infatti effettuato quando necessario un padding dei bit mancanti).

## 1.2 Interfaccia del componente

```
entity project_reti_logiche is
  port (
    i_clk    : in std_logic;
    i_rst    : in std_logic;
    i_start  : in std_logic;
    i_w      : in std_logic;

    o_z0     : out std_logic_vector(7 downto 0);
    o_z1     : out std_logic_vector(7 downto 0);
    o_z2     : out std_logic_vector(7 downto 0);
    o_z3     : out std_logic_vector(7 downto 0);
    o_done   : out std_logic;

    o_mem_addr : out std_logic_vector(15 downto 0);
    i_mem_data : in std_logic_vector(7 downto 0);
    o_mem_we   : out std_logic;
    o_mem_en   : out std_logic
  );
end project_reti_logiche;
```

In particolare:

- **i\_clk** è il segnale di CLOCK in ingresso generato dal testBench.
- **i\_rst** è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START.
- **i\_start** è il segnale di START generato dal TestBench.
- **i\_w** è il segnale, generato dal testBench, che fornisce i dati in ingresso da leggere sul fronte di salita del clock.
- **o\_z0** rappresenta il primo canale di uscita.
- **o\_z1** rappresenta il secondo canale di uscita.
- **o\_z2** rappresenta il terzo canale di uscita.
- **o\_z3** rappresenta il quarto canale di uscita.
- **o\_done** è il segnale di uscita che comunica il termine dell'elaborazione.
- **o\_mem\_addr** è il segnale di uscita che manda l'indirizzo alla memoria.
- **i\_mem\_data** è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura.
- **o\_mem\_we** è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per poter leggere dalla memoria, invece, tale segnale deve essere a 0.
- **o\_mem\_en** è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che scrittura).

## 1.3 Dati e memoria

Come descritto precedentemente, l'ingresso principale del sistema è rappresentato dal segnale **i\_w** dal quale, sequenzialmente e sul fronte di salita del CLOCK, viene letto un numero variabile di bit, tra 2 e 18 (estremi inclusi), dove:

- I primi due bit rappresentano il canale di uscita

- Gli altri N bit rappresentano l'indirizzo di memoria da cui reperire il dato da trasmettere in uscita

Gli indirizzi di memoria da cui reperire i dati sono sempre costituiti da 16 bit.

Il messaggio associato all'indirizzo di memoria è, invece, sempre da 8 bit.

## 2 ARCHITETTURA

### 2.1 Segnali usati

Per poter implementare il componente, i segnali interni che abbiamo utilizzato sono i seguenti:

```
signal flag_shift    : std_logic;
signal reg_mem_load  : std_logic;
signal reg_out       : std_logic;
signal done_ok       : std_logic;
signal reg_mem_data  : std_logic_vector(7 downto 0);
signal reg1_data     : std_logic_vector(15 downto 0) := "0000000000000000";
signal reg3_data     : std_logic_vector(1 downto 0)  := "00";
signal regz0_data    : std_logic_vector(7 downto 0);
signal regz1_data    : std_logic_vector(7 downto 0);
signal regz2_data    : std_logic_vector(7 downto 0);
signal regz3_data    : std_logic_vector(7 downto 0);
signal shift_counter : std_logic_vector(4 downto 0)  := "00001";
signal count         : std_logic_vector(4 downto 0)  := "00000";
```

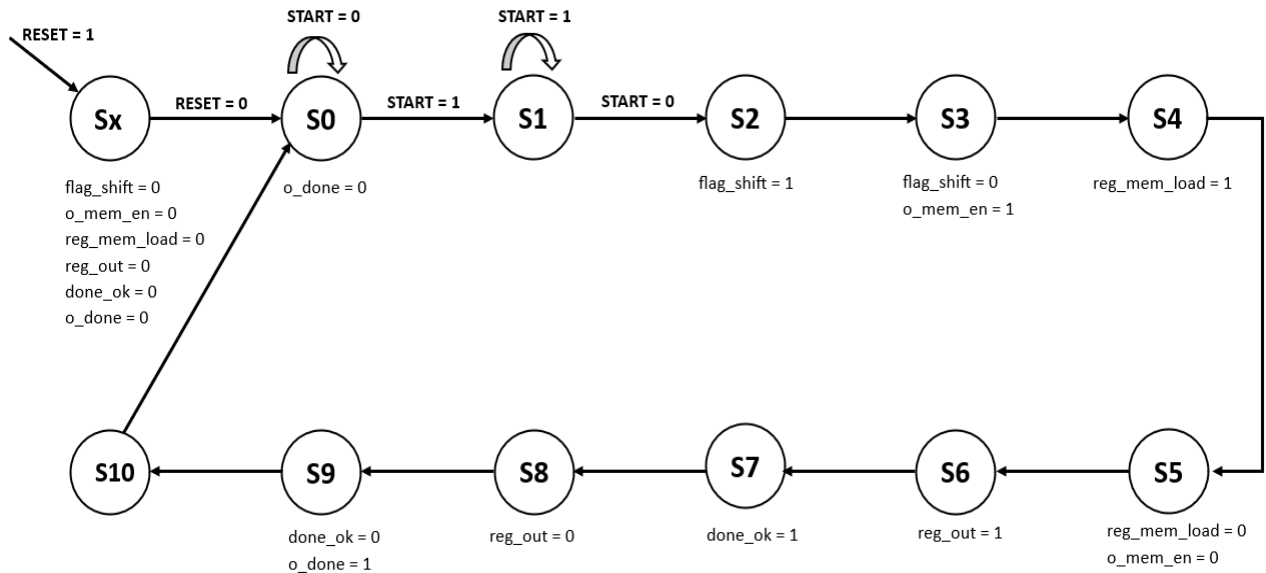
Dove in particolare:

- I segnali *reg\_out* e *reg\_mem\_load* indicano quando devono essere caricati rispettivamente i registri *regz0\_data*, *regz1\_data*, *regz2\_data*, *regz3\_data* (sulla base, chiaramente, del corretto valore letto sull'ingresso seriale *i\_w*) e *reg\_mem\_data* con il valore letto all'indirizzo di memoria.
- I segnali (vettori) *reg1\_data* e *reg3\_data* vengono utilizzati per salvare, rispettivamente, l'indirizzo di memoria da cui leggere il dato e l'uscita verso cui tale dato deve essere trasmesso. Per entrambi questi vettori non è presente un segnale che ne 'abiliti' la modifica, in quanto viene utilizzato lo stesso segnale di *i\_start*.
- Il segnale (vettore) *count* viene utilizzato come supporto per la corretta memorizzazione dei bit di *i\_w*, letti sequenzialmente e sul fronte di salita del clock.
- Il segnale *shift\_counter* contiene la costante '1' ed è stato utilizzato sia in fase di lettura dei bit sull'ingresso *i\_w* per incrementare il valore di *count*, sia in fase di elaborazione dell'indirizzo di memoria per effettuare il corretto padding dell'indirizzo stesso.
- Il segnale *flag\_shift* viene utilizzato per indicare quando deve essere effettuato il calcolo dell'indirizzo di memoria.
- Il segnale *done\_ok* viene utilizzato come segnale per abilitare la scrittura dai valori presenti nei registri *regz0\_data* ecc. ai canali di uscita *o\_z0* ecc.

Tutti i segnali sopra descritti, ad eccezione di shift\_counter che contiene la costante '1', hanno '0' come valore di default.

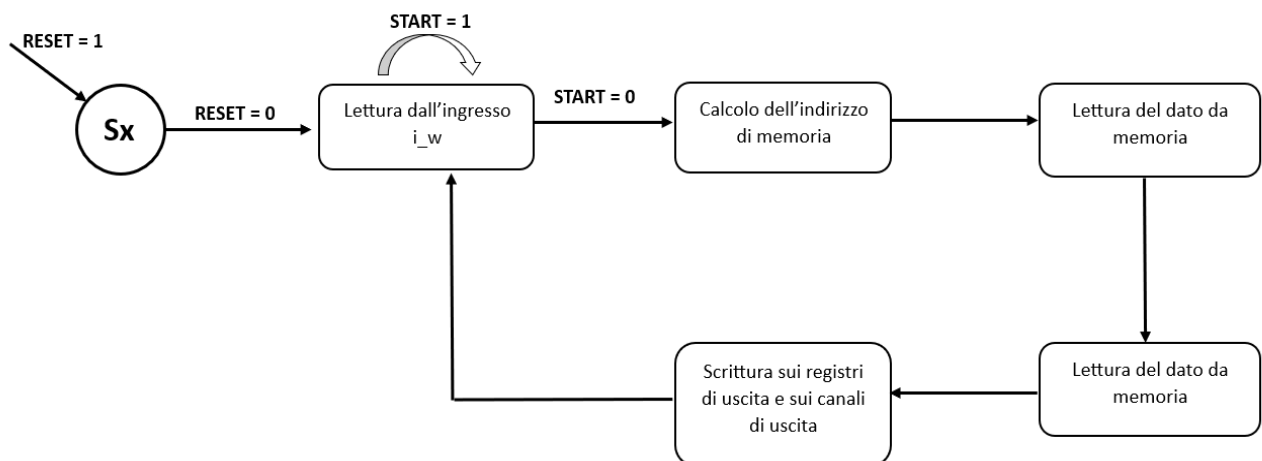
## 2.2 Macchina a stati

La rappresentazione completa della macchina a stati è la seguente:



*Osservazione: In questa immagine sono omesse, per semplicità, le frecce che riportano allo stato Sx da qualsiasi stato nel caso in cui i\_rst sia posto ad 1 in modo asincrono.*

Una versione più semplice e facilmente comprensibile della macchina a stati è la seguente:



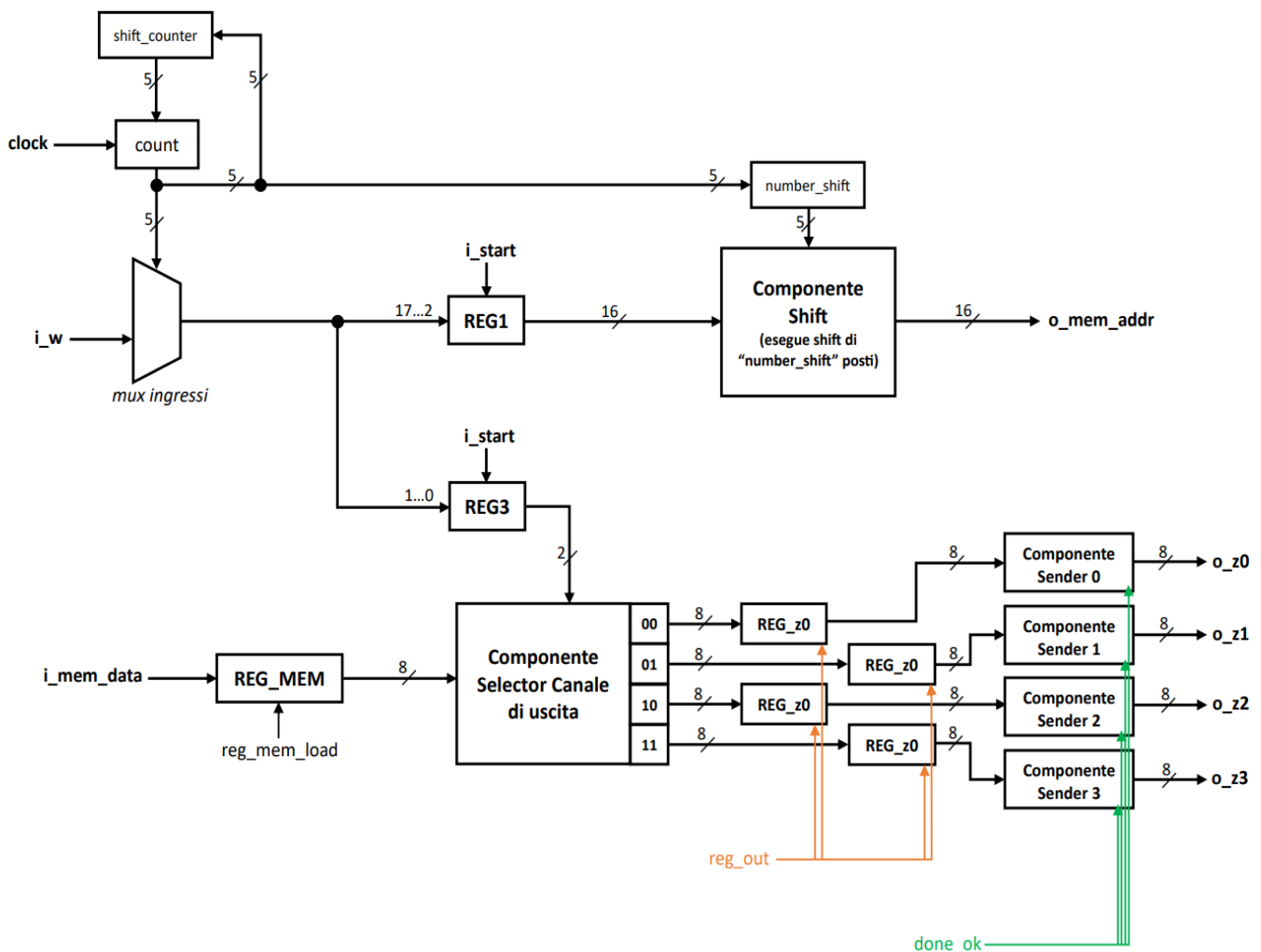
Descrizione:

- **Stato Sx:** rappresenta lo stato di reset, quello a cui la macchina torna non appena i\_rst = 1.

- **Lettura dall'ingresso  $i_w$ :** rappresenta lo stato in cui vengono letti, sequenzialmente, i bit in ingresso, che vengono opportunamente salvati nei registri  $reg1\_data$  e  $reg3\_data$ .
- **Calcolo dell'indirizzo di memoria:** in questo stato viene processato il contenuto di  $reg1\_data$ , colui che contiene l'indirizzo di memoria, affinché risulti essere correttamente esteso a 16 bit, aggiungendo, eventualmente, i bit di padding.
- **Lettura del dato da memoria:** in questo stato viene letto il dato all'indirizzo di memoria fornito e viene salvato nel registro  $reg\_mem\_data$ .
- **Scrittura sui registri di uscita e sui canali di uscita:** in questo stato scriviamo sul registro di uscita corretto il valore letto da memoria e, quando  $o\_done$  passa ad 1, scriviamo sui canali di uscita i valori opportuni.

## 2.3 Schema Funzionale

La rappresentazione dello schema funzionale del nostro sistema è la seguente:



### 3 Risultati Sperimentali

#### 3.1 Report di sintesi

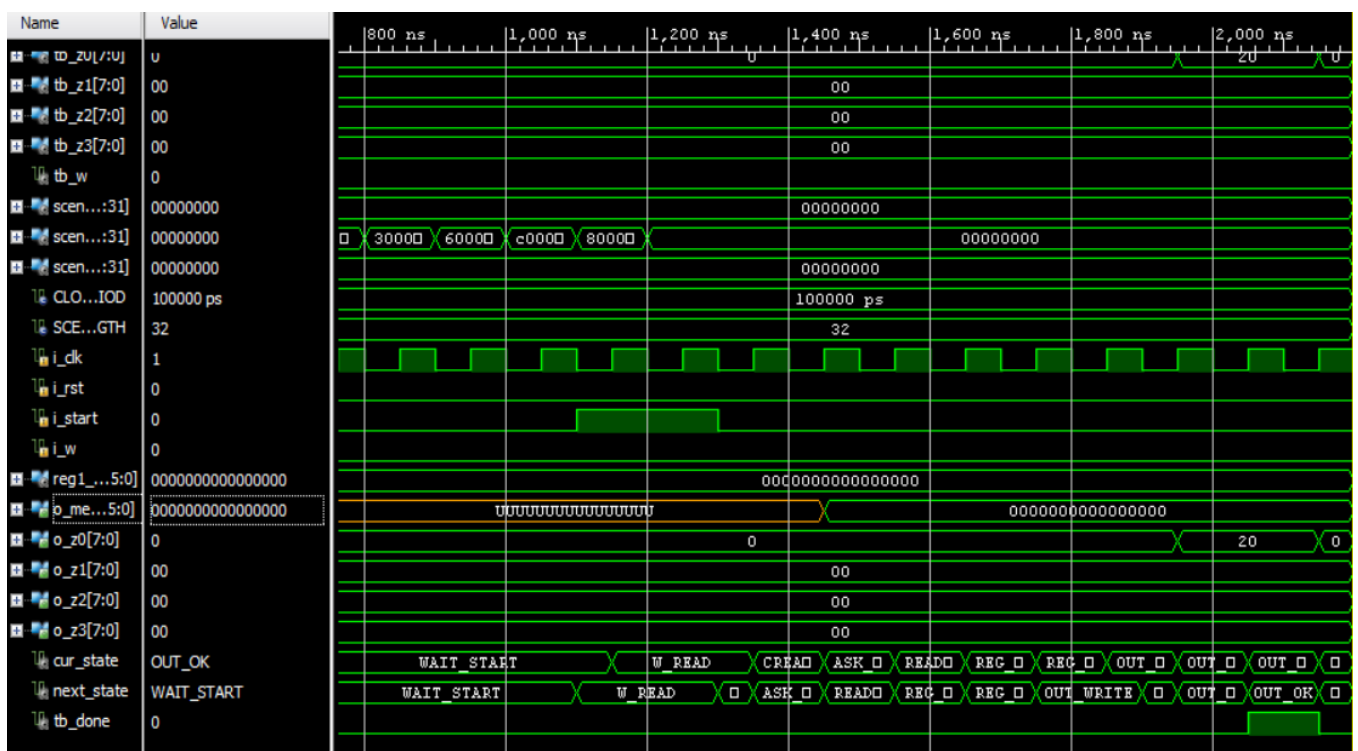
Il circuito risulta correttamente sintetizzabile ed implementabile da Vivado. A seguito del processo di sintesi risultano utilizzate 85 LUT (look up tables) e 115 flip flop che rappresentano, rispettivamente, lo 0.06% e lo 0.04% delle reali capacità della FPGA utilizzata.

#### 3.2 Simulazioni

Il circuito è stato sottoposto a numerose simulazioni per testarne l'efficacia e correttezza. Di seguito si riportano i risultati ottenuti, in behavioral simulation, con particolare attenzione ai casi particolari.

##### 3.2.1 Indirizzo di memoria costituito da sedici bit a 0

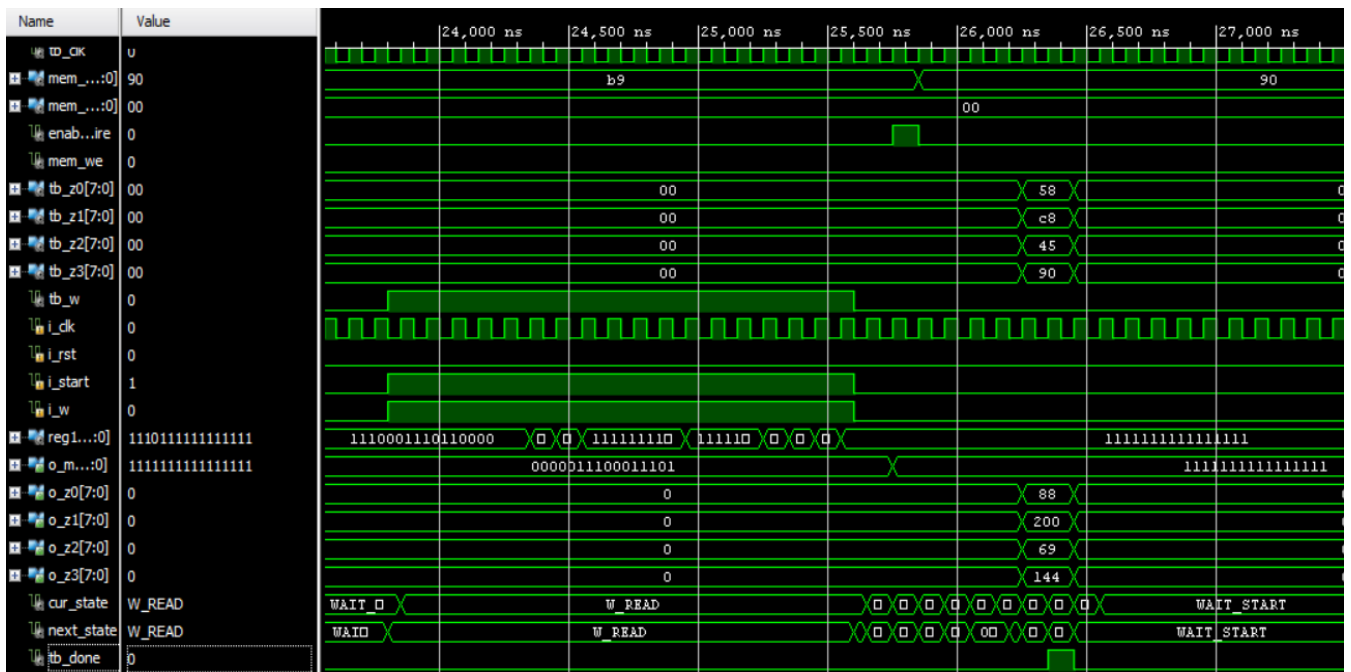
In questo caso, nel periodo in cui il segnale `i_start` rimane alto, `i_w` rimane sempre basso e il risultato è il seguente:



##### 3.2.2 Indirizzo di memoria costituito da sedici bit a 1

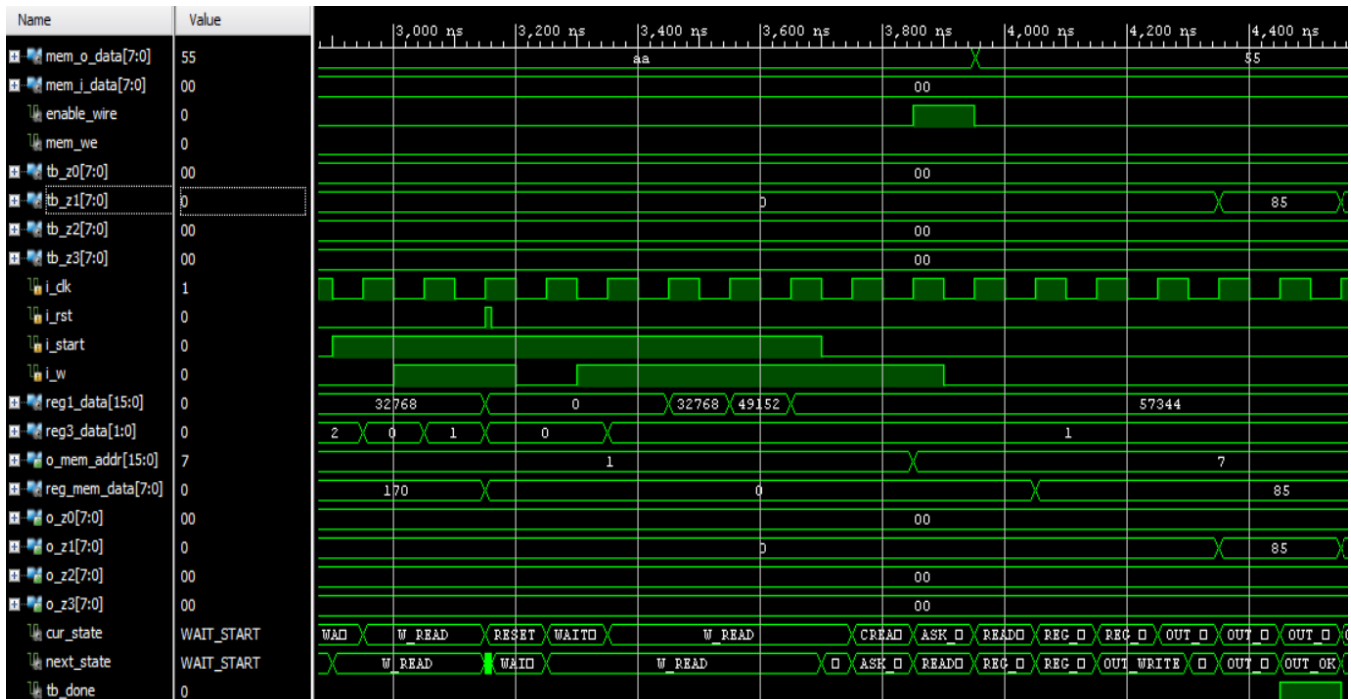
In questo caso invece, nel periodo in cui il segnale `i_start` rimane alto, `i_w` rimane sempre alto e il risultato è il seguente:





### 3.2.3 Reset asincrono

In questo test viene verificato il comportamento del circuito quando il segnale `i_rst` diventa alto in un momento arbitrario, in particolare quando il segnale di start è alto. Il risultato è il seguente:



## 4 Conclusioni

Questo progetto ci ha permesso di crescere e migliorare sotto diversi punti di vista: innanzitutto ci ha consentito di affinare le nostre abilità di lavoro in gruppo e, inoltre, di apprendere un linguaggio di descrizione dell'hardware, Vivado, che prima non conosceavamo e che richiede ragionamenti ed accortezze molto diverse dai linguaggi a cui eravamo abituati.

Per noi è stato particolarmente interessante, e stimolante, vedere come i concetti appresi durante il corso di reti logiche possano essere tradotti nella pratica, arrivando ad avere un componente che possa essere realizzato concretamente.