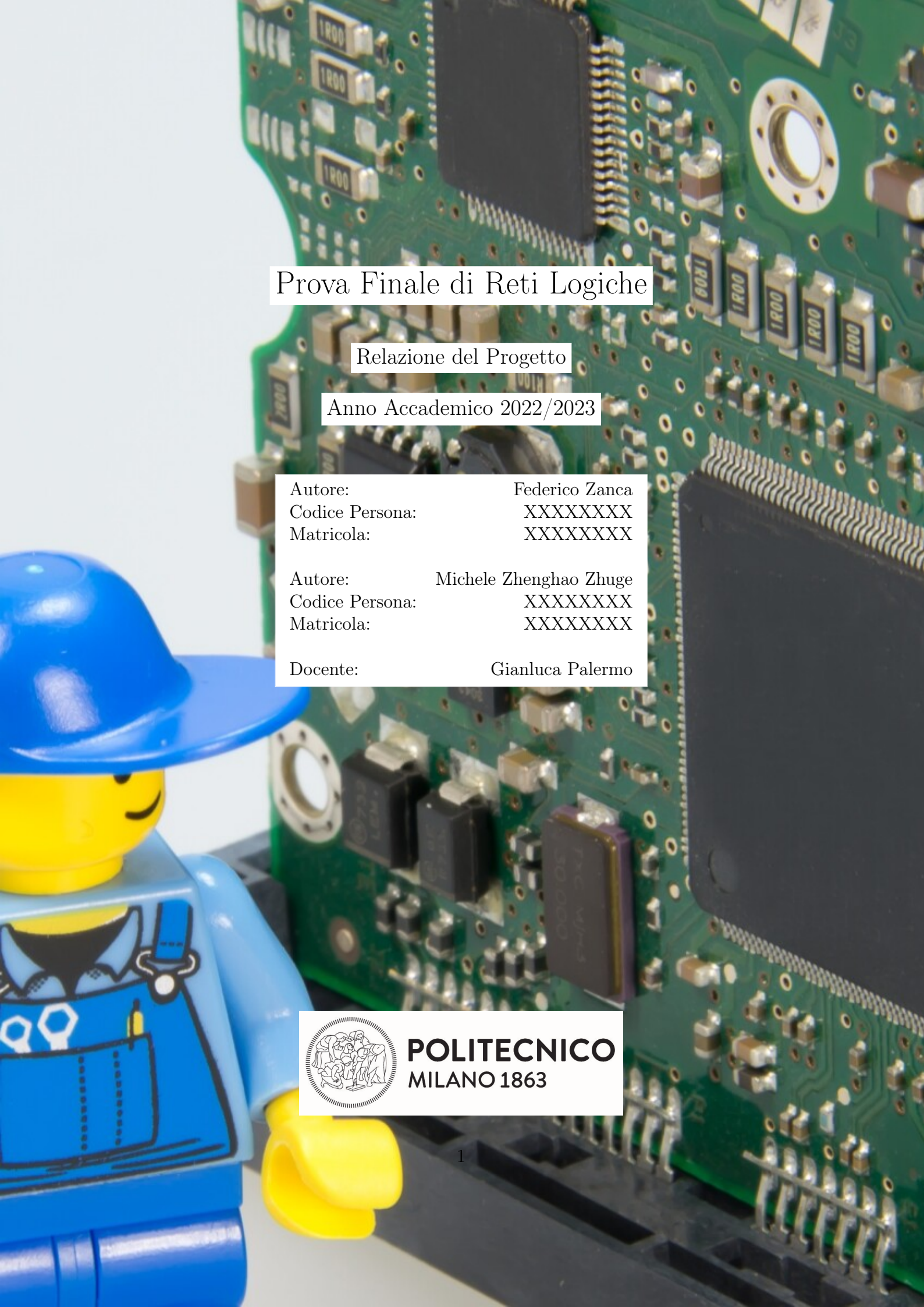


Prova Finale di Reti Logiche

Relazione del Progetto

Anno Accademico 2022/2023

Autore:	Federico Zanca
Codice Persona:	XXXXXXXX
Matricola:	XXXXXXXX
Autore:	Michele Zhenghao Zhuge
Codice Persona:	XXXXXXXX
Matricola:	XXXXXXXX
Docente:	Gianluca Palermo

POLITECNICO MILANO 1863

Indice

1 Introduzione

1.1 Scopo del progetto

L'obiettivo del progetto consiste nel descrivere in linguaggio VHDL e sintetizzare un componente hardware che si interfacci con una memoria estraendo il contenuto presente all'indirizzo ricevuto in input.

Il contenuto estratto deve poi essere indirizzato verso un canale di uscita (anche questo specificato nell'input del componente) fra i quattro disponibili.

Le informazioni relative al canale di uscita e all'indirizzo di memoria a cui accedere sono fornite al componente mediante un ingresso seriale da un bit. I quattro canali di uscita forniscono tutti i bit del valore prelevato dalla memoria parallelamente.

Software di sintesi: XILINX VIVADO WEBPACK

FPGA target per la sintesi: FPGA xc7a200tfbg484-1.

1.2 Interfacce e Funzionamento

Il componente ha due ingressi primari da 1 bit, un segnale di clock, uno di reset e 5 uscite primarie, i cui dettagli sono spiegati di seguito.

1.2.1 Ingressi

Ingresso W I dati in ingresso vengono ottenuti in sequenza sull'ingresso primario seriale W, i cui bit vengono letti sul fronte di salita del clock.

I dati sono organizzati nel seguente modo:

- i primi 2 bit identificano il canale di uscita su cui indirizzare la parola di memoria estratta. Il primo bit è il più significativo e nello specifico:
  - 00 identifica l'uscita Z0;
  - 01 identifica l'uscita Z1;
  - 10 identifica l'uscita Z2;
  - 11 identifica l'uscita Z3;

  I due bit del canale sono seguiti da
- N bit di indirizzo della memoria a cui accedere; gli N bit di indirizzo possono variare da un minimo di 0 a un massimo di 16 bit. La memoria lavora esclusivamente con indirizzi di 16 bit quindi indirizzi in ingresso

di lunghezza inferiore devono essere estesi con 0 sui bit più significativi. Per esempio:

$(N = 6)$	100110	$->$	0000000000100110
$(N = 16)$	1001001110100011	$->$	1001001110100011
$(N = 0)$	0000000000000000	$->$	0000000000000000

Ingresso START La sequenza di ingresso W viene considerata valida dal componente solo quando il segnale di ingresso primario seriale ad 1 bit START è alto (START=1), mentre termina quando START è basso (START=0). START è garantito rimanere alto per un minimo di 2 cicli di clock e per un massimo di 18 (si ha qui il caso limite in cui oltre ai 2 bit di indirizzo di canale si ha un indirizzo di 16 bit nativamente).

RESET Quando arriva il segnale di RESET il componente viene re-inizializzato. Il modulo deve essere progettato considerando che prima del primo START=1 verrà sempre dato RESET=1, ma non è necessario per le elaborazioni successive con START=1.

1.2.2 Uscite

Segnale DONE Il segnale di output DONE rimane a 0 finché il componente non ha ricevuto correttamente il valore desiderato dalla memoria; a questo punto DONE viene settato a DONE=1 per un solo ciclo di clock. Il segnale l segnale START è garantito rimanere a 0 fino a che il segnale DONE non è tornato a 0.

Canali di uscita Z# Le uscite Z0, Z1, Z2, e Z3 sono inizialmente 0000 0000 (uscite a 8 bit). Questi valori rimangono inalterati ad eccezione del canale sul quale viene mandato il valore letto dalla memoria; i valori sono visibili solo quando DONE=1, quando DONE=0 le uscite devono essere rigorosamente settate a 0000 0000.
Quando DONE=1 il canale associato al messaggio cambierà il suo valore mentre gli altri canali mostreranno l'ultimo valore trasmesso derivato dai messaggi precedenti ad esso associati.

1.3 Requisiti prestazionali del componente

Il tempo massimo per produrre il risultato (ovvero il tempo trascorso tra START=0 e DONE=1) deve essere inferiore a 20 cicli di clock.

1.4 Interfaccia del componente

L'interfaccia del componente da implementare è la seguente:

```
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;
```

i_clock	Segnale di CLOCK dato in ingresso dal Test Bench
i_rst	Segnale di RESET che re-inizializza il componente e lo prepara ad una nuova esecuzione; necessario RESET=1 prima della prima elaborazione
i_start	Segnale di START generato dal Test Bench per iniziare l'elaborazione
i_start	Segnale di START generato dal Test Bench
i_w	È il segnale W precedentemente descritto e generato dal Test Bench
o_z0, o_z1, o_z2, o_z3	Sono i quattro canali di uscita
o_done	è il segnale di uscita che comunica la fine dell'elaborazione;
o_we	Segnale di ENABLE che abilita la scrittura in memoria
o_mem_addr	È il segnale (vettore) di uscita che manda l'indirizzo alla memoria

i_mem_data	È il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura
o_mem_en	È il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura)
o_mem_we	È il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0

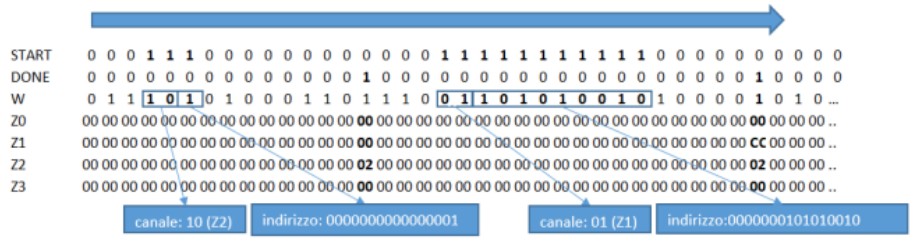
1.5 Descrizione della memoria

Il modulo comunica con la memoria, il cui protocollo è di seguito riportato.

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

entity rams_sp_wf is
    port(
        clk : in std_logic;
        we : in std_logic;
        en : in std_logic;
        addr : in std_logic_vector(15 downto 0);
        di : in std_logic_vector(7 downto 0);
        do : out std_logic_vector(7 downto 0)
    );
end rams_sp_wf;

architecture syn of rams_sp_wf is
    type ram_type is array (65535 downto 0)
                    of std_logic_vector(7 downto 0);
    signal RAM : ram_type;
    begin
        process(clk)
        begin
            if clk'event and clk = '1' then
                if en = '1' then
                    if we = '1' then
                        RAM(conv_integer(addr)) <= di;
                        do              <= di after 2 ns;
                    else
                        do <= RAM(conv_integer(addr)) after 2 ns;
                    end if;
                end if;
            end if;
        end process;
    end syn;
```



		reset	00	00	00	00	[LSB … MBS | LSB MSB]
CANALE	INDIRIZZO	DATO	Z0	Z1	Z2	Z3	W utile
02	0001	**02**	00	00	02	00	**101**
01	0152	**CC**	00	CC	02	00	**01001010110**
02	045B	A1	00	CC	A1	00	**1101101000101**
00	0751	7E	7E	CC	A1	00	**1000101011100**
00	07B2	69	69	CC	A1	00	**0100110111100**
02	0A58	7B	69	CC	7B	00	**00011010010101**
00	0B1C	AC	AC	CC	7B	00	**00111000110100**
03	0DFC	EA	AC	CC	7B	EA	**00111111101111**
00	1025	03	03	CC	7B	EA	**101001000000100**
01	1335	D8	03	D8	7B	EA	**101011001100110**
02	1605	4F	03	D8	4F	EA	**101000000110101**

Figura 1

1.6 Esempio di funzionamento

In Figura 1 è riportato un esempio di cosa dovrebbe riportare in uscita il componente a fronte di una certa sequenza in ingresso.

2 Architettura

Per risolvere il problema si è scelto di ricorrere a una FSM realizzata ad hoc, i cui stati avessero ciascuno uno scopo preciso.

2.1 Funzionamento della macchina e scopo dei singoli stati

Le operazioni svolte dai singoli stati sono:

a. Attesa del segnale di START alto (START = 1);

b. Lettura dei due bit del canale di uscita;

c. Lettura degli N bit dell'indirizzo di memoria a cui accedere;

d. Richiesta del dato alla memoria una volta disponibile l'indirizzo completo;

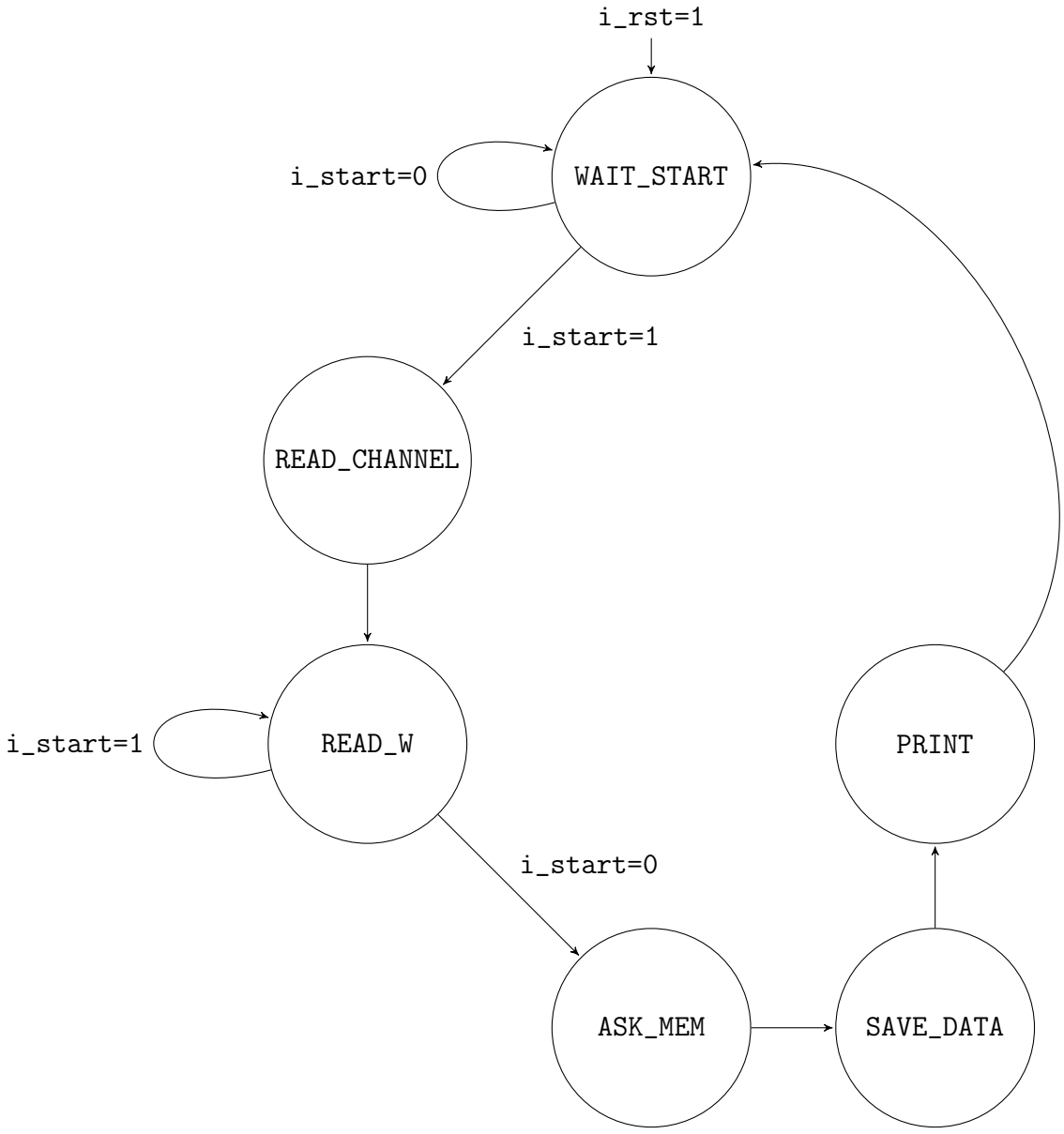
e. Ricezione del dato dalla memoria e salvataggio di quest'ultimo in un apposito registro;

f. Stampa delle 4 uscite ed emissione del segnale DONE=1;

g. Ritorno allo stato iniziale

2.2 Diagramma degli stati

Per semplicità e chiarezza sono omessi dal diagramma gli archi di RESET (`i_rst = 1`) uscenti da ogni stato che riportano allo stato iniziale.



2.3 Descrizione degli stati della FSM

WAIT_START	Stato iniziale in cui la FSM rimane finché il segnale di START non diventa 1. Quando `i_start` passa da 0 a 1 la macchina va allo stato successivo.
READ_CHANNEL	Stato in cui vengono letti i primi due bit di `i_w`, relativi al canale di uscita; si rimane in questo stato per i soli 2 cicli di clock necessari a completare questa operazione
READ_W	Stato in cui vengono letti i bit di `i_w` relativi all'indirizzo di memoria a cui accedere; quando `i_start` diventa di nuovo 0 si smette di leggere l'input e si passa quindi allo stato successivo
ASK_MEM	Stato in cui viene interrogata la memoria fornendole l'indirizzo a cui accedere ricavato negli stati precedenti
SAVE_DATA	Stato in cui si salva il dato che arriva dalla memoria
PRINT	Stato in cui si emette il segnale DONE=1 (`o_done`) e si stampano sulle 4 uscite i dati estratti dalla memoria fino a quel momento nei relativi canali assegnati

2.4 Segnali interni, variabili e tipi utilizzati

Per descrivere la macchina in VHDL si è deciso di definire un tipo di dato (S) che enumerasse tutti i possibili stati della macchina (precedentemente elencati). Di seguito sono presentati invece i segnali interni e le variabili più rilevanti utilizzate.

curr_state	Segnale che indica lo stato S in cui si trova la macchina
channel	`std_logic_vector` da 2 bit che contiene l'indirizzo del canale di uscita ricevuto in ingresso
addr	`std_logic_vector` di 16 bit in cui viene salvato l'indirizzo di memoria (ricevuto in ingresso) a cui accedere, esteso con 0 sui bit più significativi
data_reg	`std_logic_vector` da 32 bit, rappresenta il registro in cui vengono salvati i dati estratti dalla memoria. Dividendo i 32 bit in 4 chunk da 8 bit si hanno infatti le uscite Z0, Z1, Z2 e Z3 con i rispettivi dati ottenuti nelle precedenti elaborazioni, che vengono stampati contemporaneamente al segnale DONE=1
GROUND	`constant st_logic_vector` da 8 bit, tutti settati a 0. Utilizzato per rendere più semplici gli assegnamenti e la stampa quando le uscite devono essere tutte 00000000

2.5 Funzionamento nel dettaglio

La macchina inizia dallo stato di WAIT_START appena arriva il primo segnale di RESET=1 (`i_rst = 1`) dal quale esce solo a seguito dell'arrivo del segnale `i_start` alto. L'inizio di una sequenza di START=1 rappresenta l'inizio di una sequenza valida in arrivo dall'ingresso `i_w`.

Vengono quindi letti i primi due bit, relativi al canale di uscita tra i 4 disponibili, e vengono salvati nello `std_logic_vector channel`. Queste operazioni avvengono nello stato READ_CHANNEL e nella transizione verso lo stato READ_W, quindi negli stessi due cicli di clock in cui si leggono i dati in ingresso.

La macchina rimane quindi nello stato READ_W finché non si riceve in ingresso START=0, che indica la fine della sequenza di input valida. In questo arco di tempo, pari a N cicli di clock, vengono letti e salvati nello `std_logic_vector address` gli N bit di indirizzo. La lettura e lo shift avvengono contemporaneamente:

a. viene inizializzato lo `std_logic_vector address` a 0000000000000000;

b. ricevuto in ingresso il primo bit utile lo si concatena alla fine del segnale e si scarta il primo bit con una semplice operazione su `std_logic_vector` in VHDL;

c. alla fine della sequenza di ingresso si avrà un segnale composto da 16-N bit zeri seguiti dagli N bit dell'indirizzo, ovvero l'indirizzo a cui accedere esteso con degli 0 sui bit più significativi;

Disponendo dell'indirizzo completo, la macchina (avendo già del segnale `o_mem_en` settato a 1) mappa `o_mem_addr` sul segnale `addr` e fa così richiesta alla memoria per il dato presente all'indirizzo specificato.

Dopo un altro ciclo di clock (la macchina attende un ciclo di clock per permettere alla memoria di fornire il dato in tempo) il valore letto all'indirizzo ricevuto in ingresso viene salvato nel registro descritto dallo `std_logic_vector data_reg`, nel quarto di array riservato al canale corrispondente indicato dal segnale `channel`. Per chiarezza, i 32 bit di `data_reg` assumono il seguente significato:

01000110	11100100	00111000	11101101
$Z0$	$Z1$	$Z2$	$Z3$

I valori dei singoli bit sono stati scelti a caso a titolo di esempio.

A questo punto la macchina ha finito l'elaborazione, quindi nello stato PRINT assegna alle proprie 4 uscite le 4 corrispondenti sezioni da 8 bit dello `std_logic_vector data_reg` e setta `o_done = 1`.
Dopo un ciclo di clock torna allo stato WAIT_START in attesa di una nuova sequenza di ingresso valida, rimettendo `o_done=0`.

I segnali di uscita `o_done, o_z0, o_z1, o_z2` e `o_z3` rimangono a 0 in tutti gli stati tranne per lo stato di PRINT appena descritto.

2.6 Accortezze e scelte prese durante la realizzazione del progetto

Il componente è stato descritto mediante due soli processi:

- un processo della FSM che gestisce i segnali nei vari stati e le transizioni da uno stato all'altro;
- un processo dedicato al salvataggio dei dati provenienti dalla memoria nel registro e alla gestione del reset di quest'ultimo

È stato scelto questo approccio per rendere la comprensione dell'implementazione semplice e per mantenere un alto livello di astrazione rispetto alle componenti singole che costituiscono il progetto sintetizzato. Il primo processo viene attivato dalla variazione del segnale di clock o di reset, il secondo anche dalla variazione di `curr_state` e quindi dallo stato in cui si trova la FSM.

Il controllo del segnale di reset è asincrono nel primo processo e sincrono nel secondo (in quanto la macchina viene riportata immediatamente allo stato WAIT_START, quindi se il registro rappresentato dallo `std_logic_vector data_reg` viene resettato sul fronte di salita del clock non ci sono problemi visto che in questo stato le uscite del componente sono fissate a 0).
L'evoluzione della macchina è invece sincronizzata con il fronte di salita del segnale di clock.

3 Sintesi del componente

3.1 Report Utilization

Completata la sintesi Vivado fornisce numerosi report; dal Report Utilization possiamo vedere che il Device descritto dal codice VHDL utilizza 56 LookUp Tables e 102 registri (FlipFlop).

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	56	0	0	134600	0.04
LUT as Logic	56	0	0	134600	0.04
LUT as Memory	0	0	0	46200	0.00
Slice Registers	102	0	0	269200	0.04
Register as Flip Flop	102	0	0	269200	0.04
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figura 2: Report Utilization - Slice Logic

3.2 Component Info Report

Analizzando sempre il Synthesis Report, sotto la voce **Detailed RTL Component Info** è possibile vedere i dettagli dei componenti utilizzati per la sintesi del Device progettato.

```
Detailed RTL Component Info :
+---Registers :
                   32 Bit    Registers := 1
                   16 Bit    Registers := 2
                    8 Bit    Registers := 4
                    2 Bit    Registers := 1
                    1 Bit    Registers := 1
+---Muxes :
       4 Input   32 Bit         Muxes := 2
       2 Input   32 Bit         Muxes := 2
       6 Input   16 Bit         Muxes := 1
       6 Input    8 Bit         Muxes := 4
       6 Input    3 Bit         Muxes := 1
       2 Input    2 Bit         Muxes := 1
       6 Input    2 Bit         Muxes := 2
       6 Input    1 Bit         Muxes := 4
```

4 Simulazioni Pre e Post Sintesi - Custom Test Benches

Il componente dopo aver passato i test benches base è stato sottoposto a una serie test custom che lo testassero in situazioni particolari e casi limite.

Di seguito sono elencati alcuni dei più significativi e "critici".

4.1 Test Bench senza indirizzo di memoria nella sequenza in ingresso

Caso di test che valuta il comportamento del device a seguito della ricezione di una sequenza di ingresso composta dai soli due bit del canale di uscita; le specifiche garantiscono che in ingresso la sequenza di START=1 sia lunga da un minimo di 2 a un massimo di 18 bit, quindi è possibile che in ingresso venga data solo la codifica del canale di uscita senza l'indirizzo di memoria.
In questo caso l'indirizzo di memoria a cui accedere è composto da soli zeri.
ESITO DEL TEST PRE-SINTESI: ***PASSATO***
ESITO DEL TEST POST-SINTESI: ***PASSATO***

4.2 Test Bench con indirizzo di memoria lungo 16 bit

Caso di test in cui l'indirizzo di memoria fornito in ingresso è lungo 16 bit, il massimo consentito dalle specifiche. La fase critica in questo caso potrebbe essere l'estensione con zeri ai bit più significativi, che non deve avvenire perché l'indirizzo viene fornito nel formato di lunghezza richiesto dalla memoria.
ESITO DEL TEST PRE-SINTESI: ***PASSATO***
ESITO DEL TEST POST-SINTESI: ***PASSATO***

Figura 3: Simulazione - reset durante una sequenza di `start=1`

4.3 Test Bench con segnale di RESET=1 in contemporanea a una sequenza di START=1 in ingresso

Con questo test bench si è voluta testare il comportamento del device nel caso in cui venisse dato un comando di reset in mezzo alla ricezione di una sequenza di bit da considerare validi (sequenza di START=1).

Il reset asincrono permette al componente di ritornare nello stato iniziale di WAIT_START cancellando dai registri i dati precedentemente memorizzati senza perdere neanche un bit della nuova sequenza di ingresso (il cui inizio viene considerato in concomitanza con il segnale di RESET).

ESITO DEL TEST PRE-SINTESI: ***PASSATO***
ESITO DEL TEST POST-SINTESI: ***PASSATO***

4.4 Test Bench con reset durante lo stato di ASK_MEM

In questo testbench il segnale di reset arriva mentre viene settato `o_mem_en = 1` iniziando quindi l'interrogazione della memoria.

ESITO DEL TEST PRE-SINTESI: ***PASSATO***
ESITO DEL TEST POST-SINTESI: ***PASSATO***

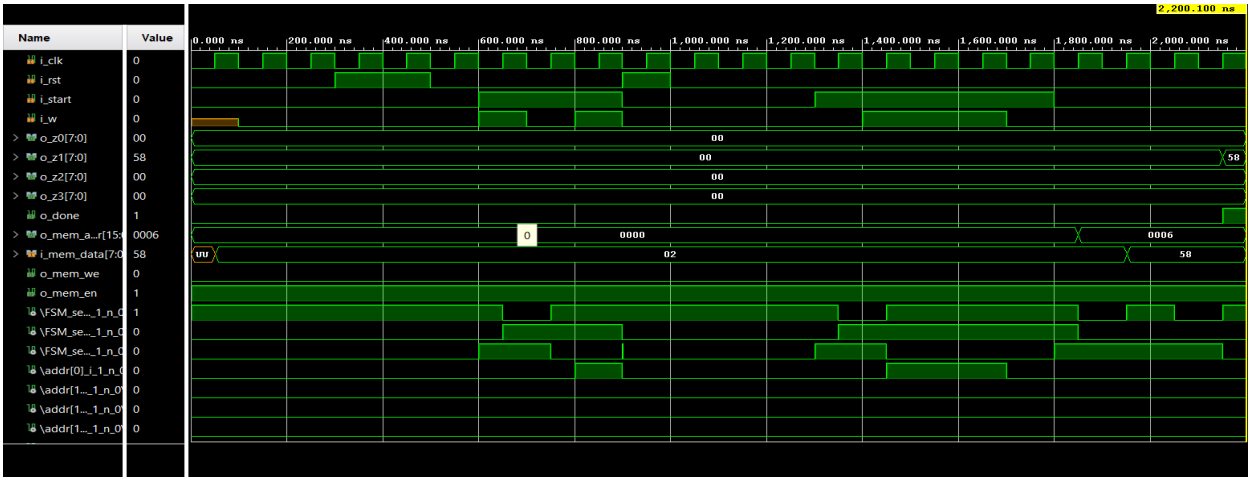


Figura 4: Simulazione - reset durante l'interrogazione della memoria

4.5 Test Bench con reset durante lo stato di SAVE_DATA

In questo testbench il segnale di reset arriva mentre il dato ricevuto dalla memoria viene salvato nel registro apposito.

ESITO DEL TEST PRE-SINTESI: ***PASSATO***
ESITO DEL TEST POST-SINTESI: ***PASSATO***

4.6 Test Bench in cui vengono sovrascritte tutte le uscite

In questo testbench le 4 uscite del componente (Z=, Z1, Z2, Z3 e Z4) sono state tutte assegnate almeno una volta e, nelle varie elaborazioni, sono anche state sovrascritte.
Lo scopo di questo testbench era quello di testare in modo esaustivo la capacità del componente di assegnare i valori giusti alle uscite giuste e di sovrascrivere tali valori quando richiesto.

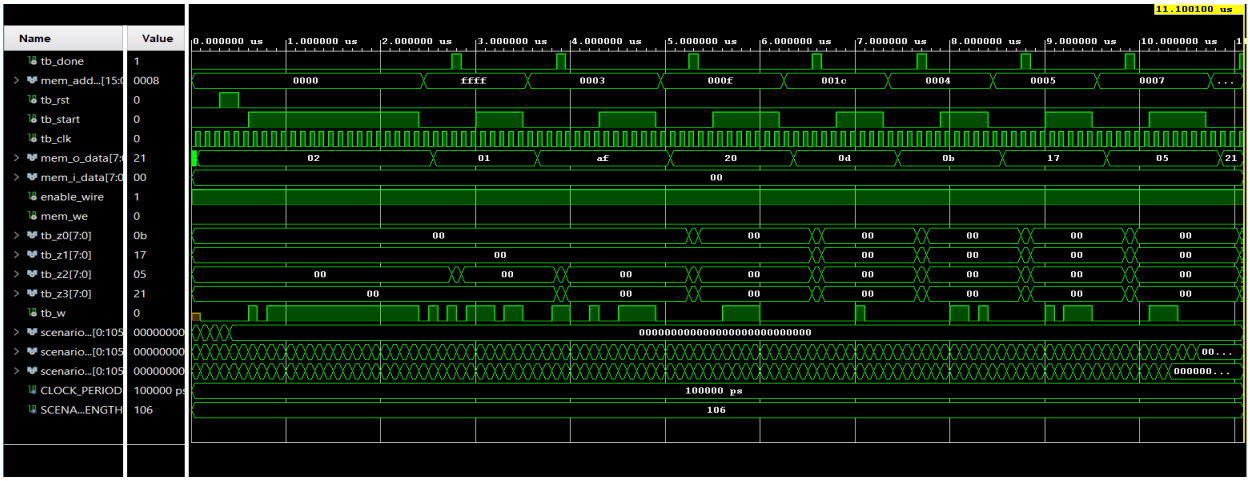


Figura 5: Simulazione - sovrascrittura di tutte le uscite

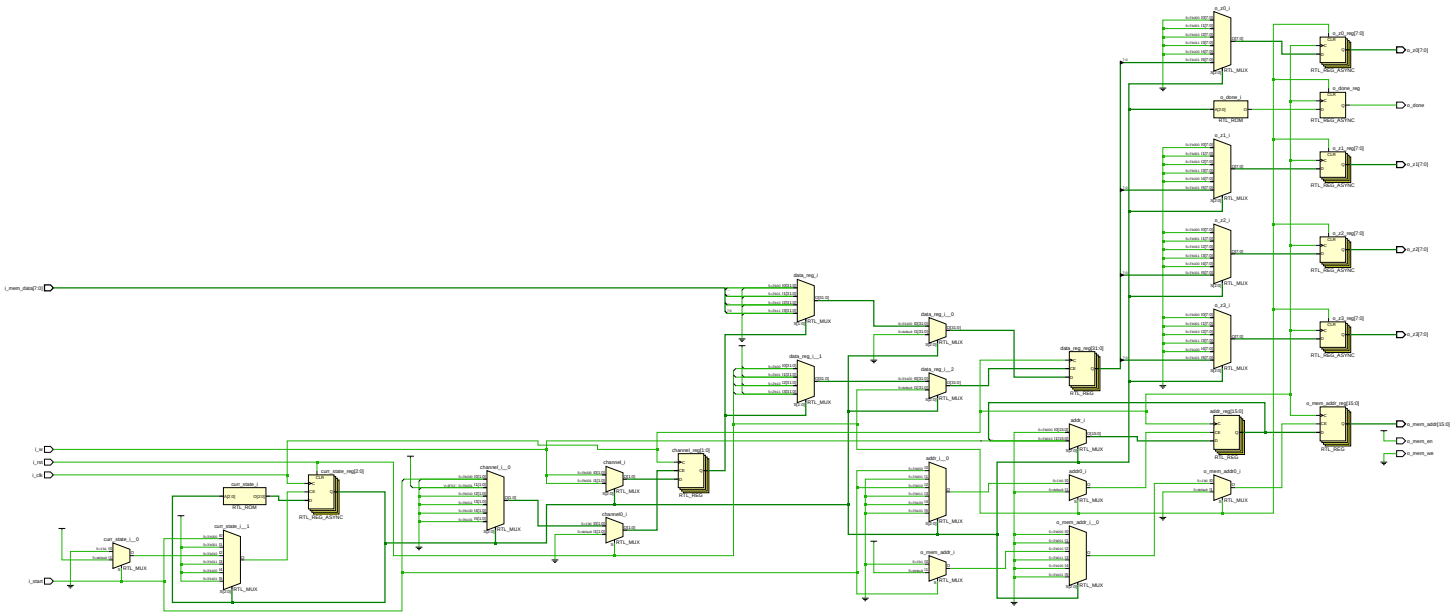
ESITO DEL TEST PRE-SINTESI: ***PASSATO***
ESITO DEL TEST POST-SINTESI: ***PASSATO***

5 Conclusioni

Il componente passa tutti i testbench a cui è stato sottoposto nel tempo richiesto dalle specifiche di progetto (tra la fine della sequenza d'ingresso e l'emissione del segnale DONE=1 intercorrono solo 3 cicli di clock contro i 20 massimi richiesti), sia in pre-sintesi che in post-sintesi.

La soluzione è stata raggiunta partendo dall'ideazione su carta della macchina a stati e di qualche componente di supporto (quali registri e multiplexer), che nell'implementazione nel codice in VHDL si è ridotta al solo processo della macchina a stati pià uno per il registro contenente i dati provenienti dalla memoria da mappare sulle quattro uscite.
Gli autori ritengono dunque che la soluzione proposta rispetti pienamente le specifiche assegnate.

La seguente immagine è lo schematico Design RTL elaborato da Vivado (si consiglia di zoomare su di essa).


i_mem_data[7:0]
i_en
i_rst
i_clk
i_start
curr_state_i_0
curr_state_i_1
curr_state_reg[2:0]
channel_i_0
channel_i
channel_reg[3:0]
data_reg_i
data_reg_i_1
data_reg_i_0
data_reg_i_2
data_reg_reg[31:0]
addr_i_0
addr_i
addr_reg[15:0]
o_mem_addr_i_0
o_mem_addr_i
o_mem_addr0_i
o_mem_addr_reg[15:0]
o_d0_i
o_d0_reg[7:0]
o_d1_i
o_d1_reg[7:0]
o_d2_i
o_d2_reg[7:0]
o_d3_i
o_d3_reg[7:0]
o_done_i
o_done_reg
o_d0[7:0]
o_d1[7:0]
o_d2[7:0]
o_d3[7:0]
o_done
o_mem_addr[15:0]
o_mem_en
o_mem_we
RTL_MUX
RTL_REG
RTL_REG_ASYNC
RTL_ROM