PROVA FINALE

Progetto di reti logiche

Nome Cognome

Docente: Fabio Salice

Anno accademico 2022/2023

Table of Contents

1 Introduzione 2
1.1 Convenzioni utilizzate 2
2 Architettura 3
2.1 Shifter buffer 5
2.1.1 Introduzione 5
2.1.2 Funzionamento 5
2.2 Shifter mask 5
2.2.1 Introduzione 5
2.2.2 Funzionamento 5
2.3 Address saver 5
2.3.1 Introduzione 5
2.3.2 Funzionamento 5
2.4 Uscita 5
2.5 Note 6
2.6 Macchina a stati 7
3 Risultati sperimentali 8
3.1 Sintesi 8
3.1.1 Timing report 8
3.1.2 Utilization report 9
3.2 Simulazioni 9
3.3 Test benches 9
3.3.1 Test 1 10
3.3.2 Test 2 11
3.3.3 Test 3 12
4 Conclusioni 12

1 Introduzione

Lo scopo del progetto è la realizzazione di un componente hardware in VHDL.
L'obiettivo del progetto è quello di descrivere ad alto livello il funzionamento di un puntatore. Data una cella di memoria si va ad interrogare la stessa per poter ricevere il dato contenuto in quella cella di memoria. In questo caso non è un puntatore modificatore ma semplicemente osservatore.
Esso riceve in ingresso una sequenza di bit che devono essere letti e divisi in canale e indirizzo. Il primo rappresenta il canale di uscita sul quale deve essere reindirizzato il messaggio; il secondo rappresenta l'indirizzo di memoria nel quale è memorizzato (in memoria RAM) il messaggio da fornire in uscita.

1.1 Convenzioni utilizzate

Il codice è commentato il più possibile coerentemente con la specifica, in modo da ritrovare facilmente le informazioni.
All'interno del progetto sono state usate delle convenzioni che verranno definite qui di seguito:

- Signal interni al modulo/componente: tutti i signal che vengono utilizzati internamente, come segnali di appoggio sono contrassegnati da un prefisso "t_". Non si interfacciano con l'esterno ma solo all'interno del modulo e/o componente, per questo vengono trattati come se fossero segnali "temporanei".
- Signal prodotti dalla macchina a stati: sono contrassegnati da un prefisso "m_", che sta per macchina a stati. Tutti questi signal sono in uscita dalla macchina a stati e in ingresso al datapath.
- Ingressi e uscite: sono chiamate come da specifica, non essendo modificabili.

2 Architettura

L'interfaccia del componente da realizzare viene già presentata nelle specifiche di progetto, nel seguente modo:

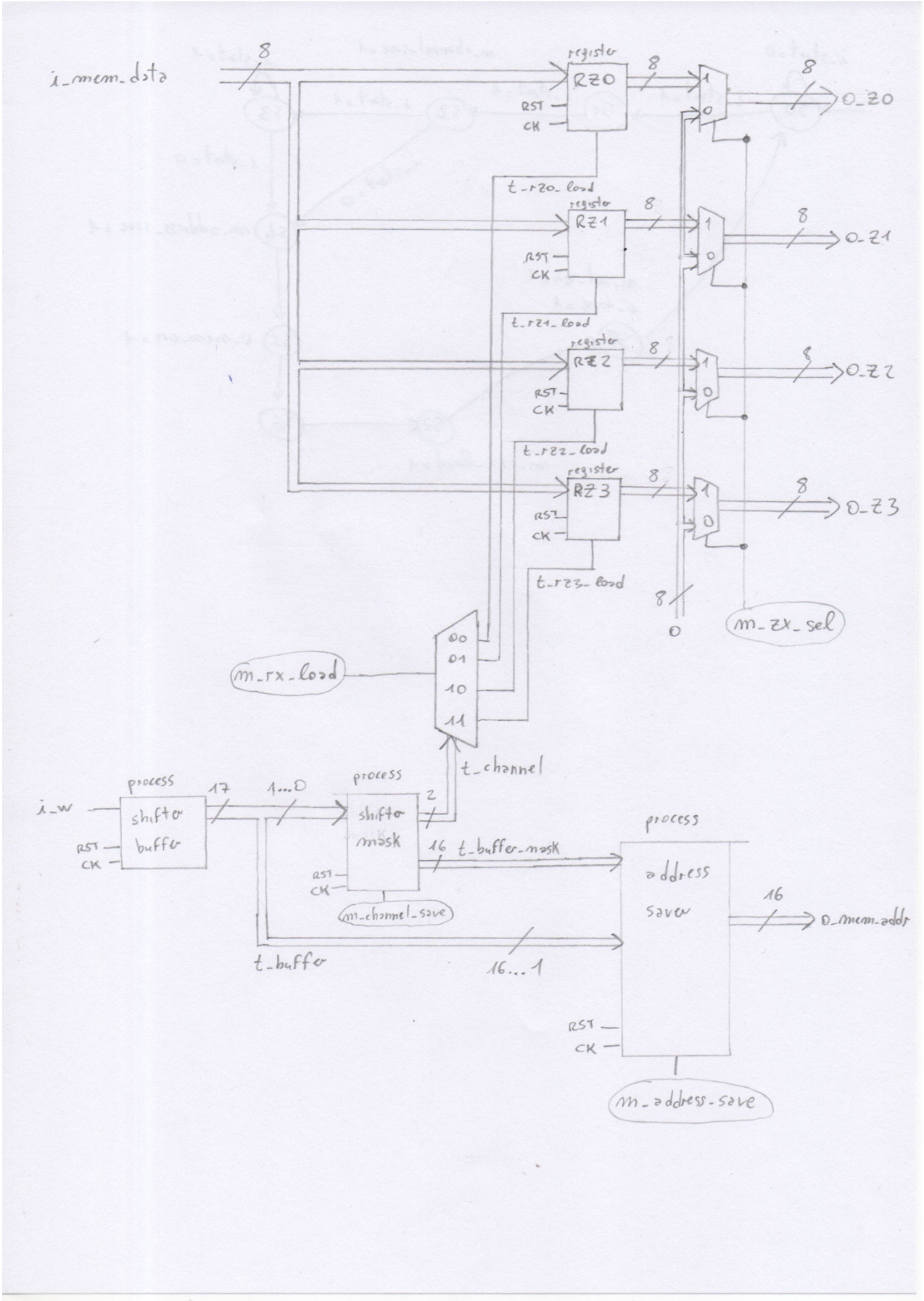
```
entity project_reti_logiche is
    port (
        i_clk   : in std_logic;
        i_rst   : in std_logic;
        i_start : in std_logic;
        i_w     : in std_logic;

        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;
```

Esso inoltre si interfaccia con una RAM su cui sono salvati i messaggi da trasmettere in uscita.

L'idea inziale ad alto livello è stata fin da subito quella di creare un registro a scorrimento, siccome il canale e l'indirizzo non arrivano in blocco ma un bit alla volta. Viene costruito un buffer in cui vengono memorizzati tutti i bit che vengono letti da W, partendo ad inserirli dal bit meno significativo al più significativo, poiché ci deve essere un padding di zeri a sinistra dell'indirizzo. In questo modo si evita di shiftare ulteriormente alla fine della lettura e di implementare un contatore.
Il canale viene semplicemente memorizzato al secondo ciclo di clock (da quando START=1), visto che è sempre fisso e il numero minimo di cicli di clock quando START=1 è 2, come da specifica.
Per l'indirizzo invece viene creata una bitmask che parte a introdurre degli 1, allo stesso modo di prima, a ogni ciclo di clock da quando viene memorizzato il canale. Questo serve a eliminare letture sporche precedenti, come il canale, che devono essere eliminate dal buffer prima di salvare l'indirizzo. Tramite un and finale bit a bit si riesce a eliminare tutta la lettura "sporca" ed estrarre l'indirizzo, avendo così già pronto il padding di zeri, se i bit trasmessi da W sono meno di 16.

Istante di tempo t	**Buffer** **Bitmask**
0 (RESET=1)	0000000000000000 0000000000000000
1 (START=1)	0000000000000001 0000000000000001
2	0000000000000010 0000000000000011
Salvataggio canale 10 e reset della bitmask	
3	0000000000000101 0000000000000001
…	…
n	0000000101100101 0000000001111111
Salvataggio indirizzo	0000000101100101 0000000001111111 ---------- AND 0000000001100101

Partendo da questa idea ad alto livello, ho cercato di suddividere il problema in sotto problemi.
Di seguito viene riportato il datapath, che descrive la logica utilizzata.



2.1 Shifter buffer

2.1.1 Introduzione

Lo scopo di questo modulo è di memorizzare tutti i bit che vengono trasmessi da W. Servirà per memorizzare canale e indirizzo trasmessi.

2.1.2 Funzionamento

Viene creato un registro a scorrimento serial input parallel output, con ingresso W e uscita a 17 bit chiamata “t_buffer”. Il segnale “t_buffer” viene resettato solo con il segnale di reset, fondamentale per poter resettare tutti i bit e procedere con lo scorrimento.

Dopodiché questo registro continua a memorizzare e scorrere verso sinistra (da LSB a MSB) tutti i bit che legge da W, indipendentemente da tutto il contorno. Vengono utilizzati 17 bit perchè la macchina a stati restituisce il segnale "m_address_save" un ciclo dopo dello stop della lettura; di conseguenza se non creo un bit in più nel buffer perdo un bit dell’indirizzo.

2.2 Shifter mask

2.2.1 Introduzione

Lo scopo di questo modulo è salvare il canale di uscita trasmesso da W quando dalla macchina a stati arriva il segnale “m_channel_save”. Dopodiché il componente procede creando una bitmask che trasmette al componente “address saver”. Quest’ultimo la utilizzerà per estrapolare l’indirizzo corretto.

2.2.2 Funzionamento

Viene creato un altro registro a scorrimento serial in e parallel out con ingresso 1 e uscita un vettore di 16 bit, che corrisponde al numero di bit che deve avere l’uscita “o_mem_addr”.

Ad ogni ciclo di clock (rising edge) viene shiftato il contenuto del segnale “t_buffer_mask” a sinistra di una posizione (sempre da LSB a MSB) e viene aggiunto un 1 a destra (LSB).

Quando la macchina a stati produce il segnale “m_channel_save” salvo i due bit fissi letti dall’ingresso in “t_channel” che diventerà il segnale di controllo del multiplexer per poter scegliere il canale giusto. Durante questa operazione viene resettato anche “t_buffer_mask” per iniziare a contare e scorrere i bit di indirizzo di memoria che vengono letti da W.

2.3 Address saver

2.3.1 Introduzione

Questo modulo permette il salvataggio dell’indirizzo trasmesso tramite un and bit a bit con la bitmask precedentemente prodotta.

2.3.2 Funzionamento

Esegue un AND bit a bit sul fronte di salita del segnale prodotto dalla macchina a stati “m_address_save”. L’operazione di AND viene eseguita tra il segnale “t_buffer_mask”, e “t_buffer” che raccoglie tutti i bit passati tramite l’ingresso W. Il risultato di questa operazione viene riportato sull’uscita “o_mem_addr” che da specifica è il segnale di uscita che manda l’indirizzo alla memoria.

2.4 Uscita

Il demultiplexer gestisce il segnale di scrittura sui registri. In questo caso uso un DEMUX per poter salvare il valore restituito da memoria nel giusto registro. Il DEMUX è così composto:

- IN: segnale prodotto dalla macchina a stati chiamato “m_rzx_load”. Diventa positivo dopo aver letto il dato da memoria, indipendentemente dal canale letto.
- SEL: canale estrapolato da W.
- OUT_00: segnale di load del registro corrispondente all’uscita Z0.

- OUT_01: segnale di load del registro corrispondente all'uscita Z1.
- OUT_10: segnale di load del registro corrispondente all'uscita Z2.
- OUT_11: segnale di load del registro corrispondente all'uscita Z3.

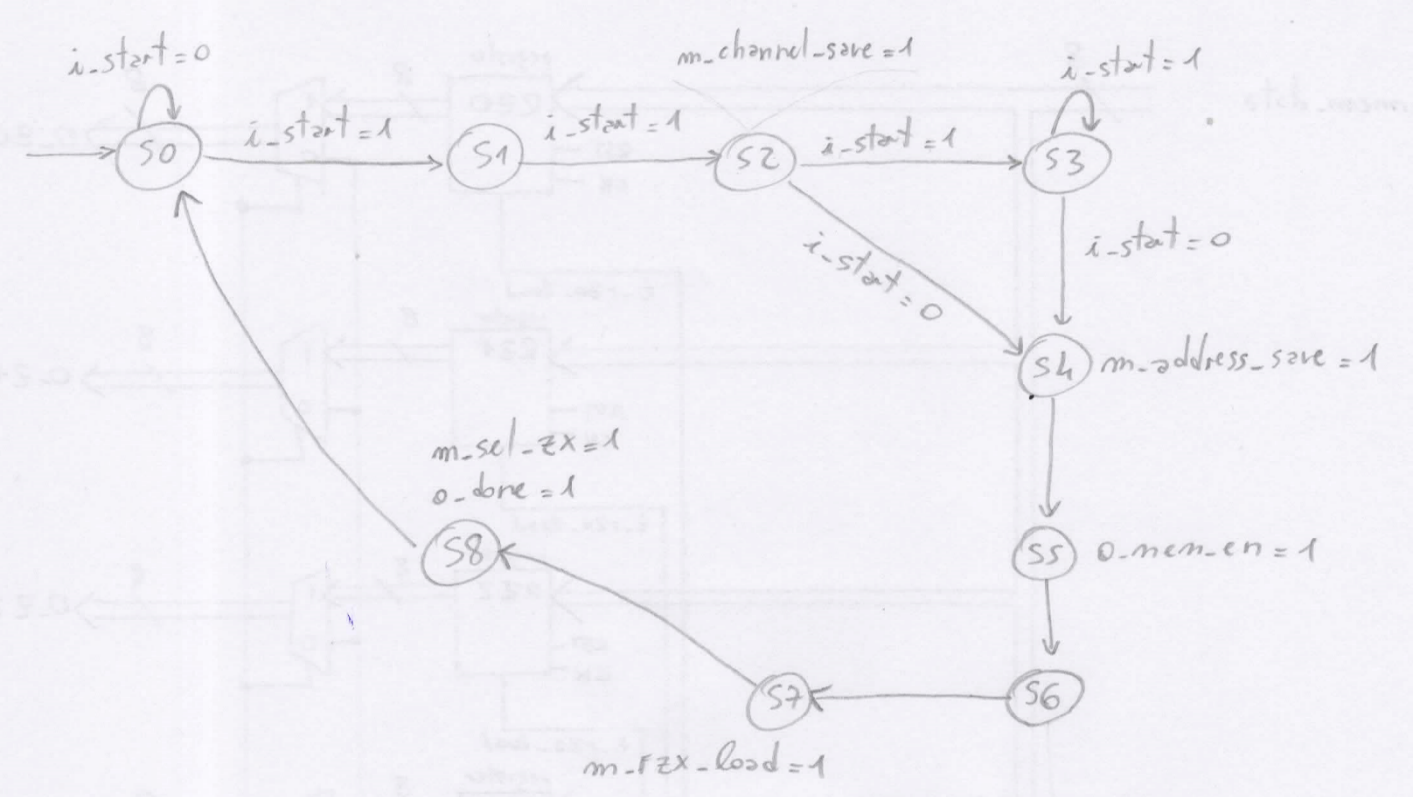
In questo modo posso selezionare il registro su cui scrivere il valore trovato quando mi arriva il segnale dalla FSM di salvare il valore restituitomi dalla memoria.
La scelta di usare dei registri prima dell'uscita "o_zx" è stata obbligata dal fatto che, da specifica, bisogna visualizzare i valori vecchi negli altri registri al momento di DONE=1.
Si visualizza invece il valore aggiornato sul canale corrispondente letto da W.
Prima di arrivare finalmente alle uscite vere e proprie "o_zx", tra i registri e l'uscita vengono stanziati dei MUX, uno per ogni uscita. I singoli MUX sono così composti

- IN0: zero. Questo permette di visualizzare sempre 0 quando DONE=0.
- IN1: valore del registro RZx. Contiene il valore precedentemente memorizzato se non è il canale scelto dalla lettura, oppure il valore aggiornato.
- SEL: è il segnale prodotto dalla FSM chiamato "m_zx_sel" che corrisponde esattamente a "o_done".
- OUT: il vettore di 8 bit corrispondente alle singole uscite "o_zx".

2.5 Note

Sia la macchina a stati che tutti i moduli vengono reinizializzati quando ricevono il segnale di RESET. Tutti i segnali vengono portati a 0 e la macchina a stati viene riportata allo stato S0. Nel grafico della macchina a stati vengono omessi tutti gli archi che portano a S0 per migliorarne la leggibilità.

2.6 Macchina a stati



Si noti che tutti i segnali che vengono portati a 1, rimangono positivi per un solo ciclo di clock. Infatti, quando si passa allo stato successivo, se non espressamente dichiarato, vengono riportati a 0.
La macchina a stati come si può vedere dal disegno ha 9 stati così suddivisi:

- S0: stato iniziale a RST=1. Rimango in S0 fino a che non leggo START=1, per passare allo stato successivo S1.
- S1: non potendo ricevere uno START=0 non riporto quel caso. Quando leggo ancora START=1 passo allo stato successivo S2.
- S2: porto il segnale “m_channel_save” a 1. Ora posso salvare il canale in un registro perché sono trascorsi 2 cicli di clock. In questo caso, a differenza di prima, posso avere che START torni a 0, se l’indirizzo è 0 oppure che START rimanga alto se deve ancora trasmettermi i bit per comporre l’indirizzo. Per questo da S2 posso entrare in due stati diversi a seconda del valore di START.
- S3: dallo stato S2 se START è rimasto a 1, e fino a che quest’ultimo rimane a 1. Vuol dire che sta trasmettendo ancora bit di indirizzo, quindi devo poter continuare a memorizzare nel mio registro a scorrimento i bit letti su W. Si noti che non posso rimanere in questo stato per sempre perché la specifica impone che ci siano massimo 18 cicli di clock con START alto.
- S4: porto il segnale “m_address_save” a 1. Ora posso trasmettere l’indirizzo letto fino a questo momento in uscita a “o_mem_addr”.
- S5: abilito “o_mem_en” per essere sicuro che la memoria legga il valore corretto che ho precedentemente mandato in uscita.
- S6: stato di appoggio per permettere ai componenti precedentemente descritti di leggere il valore restituito dalla memoria.
- S7: Porto a 1 il segnale “m_rzx_load” che in base al canale memorizzato precedentemente, tramite DEMUX andrà a selezionare il registro corretto su cui effettuare la scrittura.
- S8: porto il segnale di DONE a 1 perché ho terminato correttamente tutti i passaggi e mostro in uscita i valori precedentemente memorizzati, tranne per il canale corrispondente alla lettura attuale, che porterà in uscita il dato letto da memoria. Ritornando in S0 posso procedere con un nuovo ciclo al successivo START=1.

3 Risultati sperimentali

3.1 Sintesi

3.1.1 Timing report

```
Timing Report

Slack (MET) :             97.028ns  (required time - arrival time)
  Source:                 FSM_onehot_cur_state_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            FSM_onehot_cur_state_reg[0]/CE
                            (rising edge-triggered cell FDPE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.590ns  (logic 0.875ns (33.784%)  route 1.715ns (66.216%))
  Logic Levels:           2  (LUT5=1 LUT6=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         (clock clock rise edge)      0.000     0.000 r
                                                      0.000     0.000 r  i_clk (IN)
                         net (fo=0)                   0.000     0.000    i_clk
                         IBUF (Prop_ibuf_I_O)         0.944     0.944 r  i_clk_IBUF_inst/O
                         net (fo=1, unplaced)         0.800     1.744    i_clk_IBUF
                         BUFG (Prop_bufg_I_O)         0.096     1.840 r  i_clk_IBUF_BUFG_inst/O
                         net (fo=74, unplaced)        0.584     2.424    i_clk_IBUF_BUFG
                         FDCE                                         r  FSM_onehot_cur_state_reg[1]/C
  -------------------------------------------------------------------    -------------------
                         FDCE (Prop_fdce_C_Q)         0.456     2.880 r  FSM_onehot_cur_state_reg[1]/Q
                         net (fo=2, unplaced)         0.752     3.632    FSM_onehot_cur_state_reg_n_0_[1]
                         LUT5 (Prop_lut5_I2_O)        0.295     3.927 r  FSM_onehot_cur_state[8]_i_2/O
                         net (fo=1, unplaced)         0.449     4.376    FSM_onehot_cur_state[8]_i_2_n_0
                         LUT6 (Prop_lut6_I1_O)        0.124     4.500 r  FSM_onehot_cur_state[8]_i_1/O
                         net (fo=9, unplaced)         0.514     5.014    FSM_onehot_cur_state[8]_i_1_n_0
                         FDPE                                         r  FSM_onehot_cur_state_reg[0]/CE
  -------------------------------------------------------------------    -------------------

                         (clock clock rise edge)    100.000   100.000 r
                                                      0.000   100.000 r  i_clk (IN)
                         net (fo=0)                   0.000   100.000    i_clk
                         IBUF (Prop_ibuf_I_O)         0.811   100.811 r  i_clk_IBUF_inst/O
                         net (fo=1, unplaced)         0.760   101.570    i_clk_IBUF
                         BUFG (Prop_bufg_I_O)         0.091   101.661 r  i_clk_IBUF_BUFG_inst/O
                         net (fo=74, unplaced)        0.439   102.100    i_clk_IBUF_BUFG
                         FDPE                                         r  FSM_onehot_cur_state_reg[0]/C
                         clock pessimism              0.178   102.279
                         clock uncertainty           -0.035   102.243
                         FDPE (Setup_fdpe_C_CE)      -0.202   102.041    FSM_onehot_cur_state_reg[0]
  -------------------------------------------------------------------
                         required time                        102.041
                         arrival time                          -5.014
  -------------------------------------------------------------------
                         slack                                 97.028
```

Tramite il report di timing si riesce ad evidenziare la velocità di computazione del componente rispetto al constraint di clock fornito di 100ns. In questo caso il periodo minimo di clock con cui può funzionare il componente è 100ns – 97ns = 3ns, ovvero 1/3ns = circa 300MHz.

3.1.2 Utilization report

L'utilization report permette di analizzare l'implementazione fisica sulla FPGA del componente hardware progettato. I risultati ottenuti si possono osservare in figura.

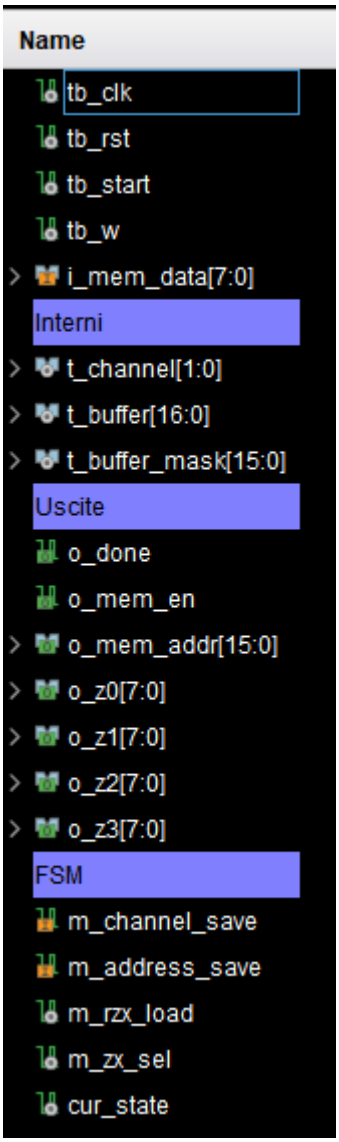
1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	43	0	134600	0.03
LUT as Logic	43	0	134600	0.03
LUT as Memory	0	0	46200	0.00
Slice Registers	92	0	269200	0.03
Register as Flip Flop	92	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

3.2 Simulazioni

Il componente una volta sintetizzato supera in modo corretto e senza errori tutti i test specificati nelle 2 simulazioni (behavioural and post synthesis).

3.3 Test benches



Per questo progetto sono stati sottoposti 7 test benches di esempio. Di seguito vengono riportati solo quelli più rilevanti dato che in alcuni test sono ridondanti e non riportano casi limite.

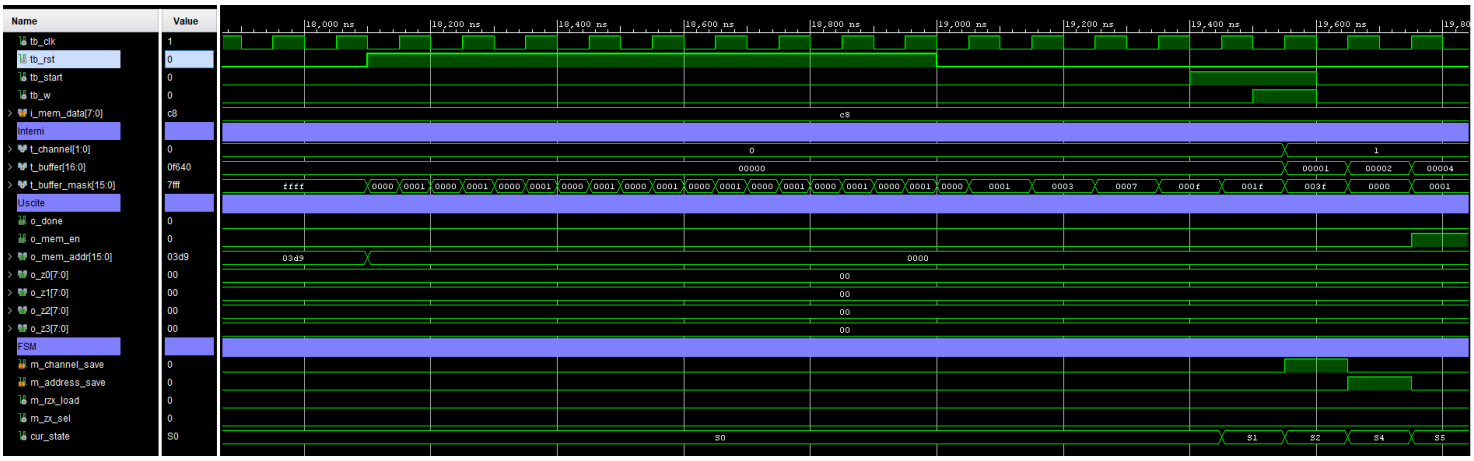
Tutti gli screenshot riportano i segnali che vengono mostrati qui di fianco:

- Segnali generati dal testbench: hanno il prefisso "tb_".
- Segnali interni: prefisso "t_" descritti sopra.
- Uscite: contrassegnati con il prefisso "o_" come da specifica.
- Segnali generati dalla FSM: prefisso "m_" e in aggiunta anche lo stato corrente della macchina a stati

3.3.1 Test 1

Questo test ha lo scopo di verificare il corretto funzionamento del segnale di reset. Questo segnale viene posto a 1 in due momenti diversi: prima del primo start e tra uno start e l'altro. Di seguito riporto alcuni screenshot delle forme d'onda con una breve descrizione sotto.

Come si può notare il segnale di reset rimane alto per diversi cicli di clock. Al reset iniziale tutti i segnali vengono portati a 0, per non avere valori indesiderati durante il funzionamento. L'unico segnale che non risulta correttamente inizializzato è l'ingresso "i_mem_addr" poiché non controllabile. Anche la bitmask creata viene azzerata durante tutta la durata del segnale di reset. Subito dopo il fronte di discesa dello stesso inizia a shiftare.

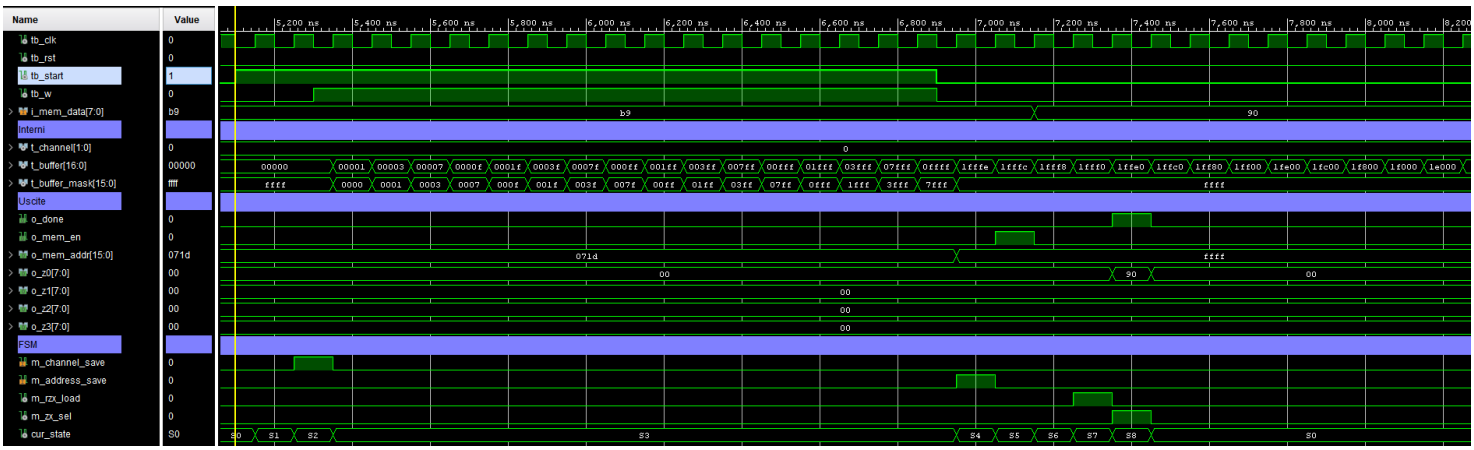


Il secondo segnale di reset viene dato tra una lettura e l'altra. Come si può sempre notare tutti i segnali vengono azzerati correttamente. Lo stato della macchina rimane in S0.

3.3.2 Test 2

Questo test ha lo scopo di controllare le situazioni limite del segnale di start, ovvero quando è alto per solo 2 CC oppure 18.

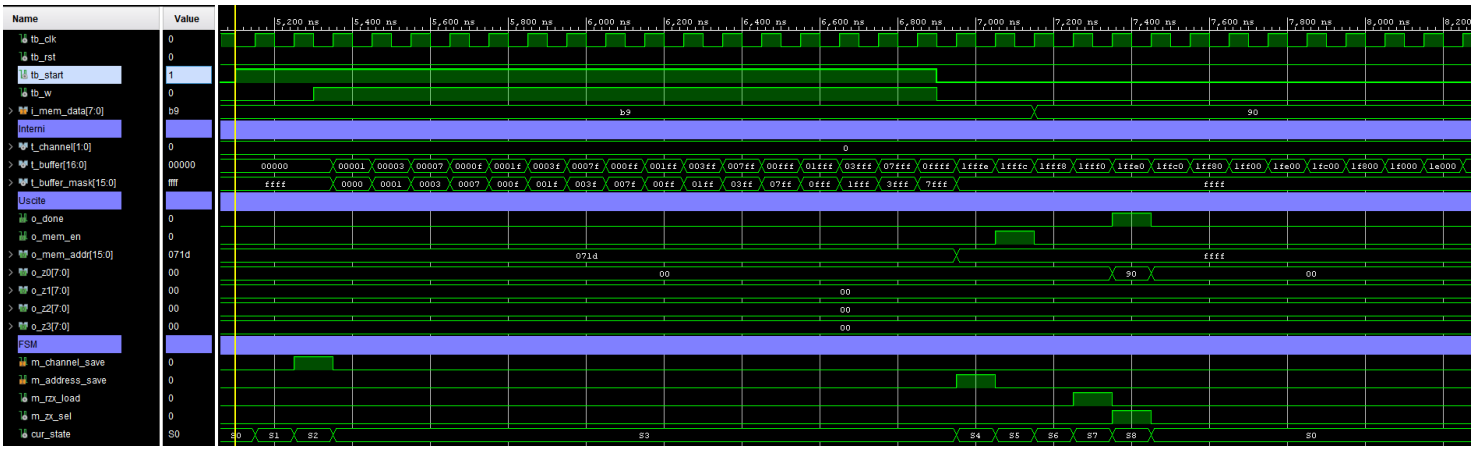
Nel caso sopra riportato START è alto solo per 2 CC. Il segnale "t_channel" memorizza correttamente 01. Il segnale "t_buffer" scorre tutti i bit che legge in input, che in questo caso non dovrebbero essere presi. La bitmask registra correttamente 0000, difatti eseguendo l'and bit a bit tra 00000 e 00002 viene correttamente mostrato in uscita "o_mem_addr" = 00000.
Gli altri segnali prodotti dalla FSM descritti nei paragrafi sopra vengono posti a 1 nel momento in cui si passa allo stato della FSM desiderato.
Il valore in uscita viene correttamente visualizzato nel canale corrispondente (in questo caso 01 ovvero z1) e solamente quando "o_done" = 1. Le altre uscite mostrano i valori vecchi



Anche in questo caso si può notare come tutti i segnali vengono visualizzati correttamente, coerentemente con i dati passati in ingresso.

3.3.3 Test 3

Questo test ha lo scopo di mostrare come le uscite non selezionate dal canale desiderato mostrino i valori vecchi salvati.



Come si può vedere dal primo grafico, viene scritto in Z0 (canale trasmesso 00) il valore 90.
Il secondo grafico rappresenta la batteria successiva alla prima. Viene mostrato correttamente durante DONE=1 che Z0 ha il valore precedente 90 mentre la nuova uscita mostra il valore passato in ingresso da “i_mem_data”.

4 Conclusioni

Il componente è stato realizzato con Vivado versione 2018.2 con FPGA Artix-7 FPGA xc7a200tfbg484-1 come da specifica.
Il componente progettato supera in tutti i casi di test forniti sia la simulazione behavioural che post-synthesis senza la presenza di latch indesiderati. Di conseguenza si ritiene di aver progettato un componente funzionante in grado di risolvere il problema dato.