Prova Finale (Progetto di Reti Logiche)

Vaninetti Lorenzo (CodicePersona: 10709137, Matricola: 956722)

Professore: Palermo Gianluca A.A. 2022/2023

1.Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è quello di implementare un modulo HW (descritto in VHDL) che, ad alto livello di astrazione, si interfacci con una memoria esterna seguendo le indicazioni ad esso fornite.

Il sistema riceve in ingresso rispettivamente l'indicazione del canale su cui indirizzare il contenuto ottenuto dalla memoria e l'indirizzo di memoria a cui accedere e procede a fornire in uscita sul canale indicatogli il contenuto della cella di memoria di cui ha ricevuto l'indirizzo.
Gli ingressi avvengono nella forma di una stringa di bit la cui lunghezza varia da 2 bit (minimo) a 18 bit (massimo): i primi due bit della sequenza identificano il canale di uscita (il primo bit ricevuto è sempre il più significativo) mentre i restanti bit servono per costruire il corretto indirizzo di memoria a cui il modulo dovrà accedere.
L'ingresso è seriale mentre le uscite forniscono i bit della parola di memoria in parallelo.

1.2 Interfaccia del componente

Il componente da descrivere deve avere la seguente interfaccia:

```
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 Test Bench;
- 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 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_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.

1.3 Specifiche generali

Andando più nello specifico, il componente deve avere il seguente comportamento:

- All'istante iniziale, quello relativo al reset del sistema, le uscite hanno i seguenti valori: Z0, Z1, Z2 e Z3 sono 0000 0000, DONE è 0.
- I dati in ingresso, ottenuti come sequenze sull'ingresso primario seriale W lette sul fronte di salita del clock, sono organizzati nel seguente modo:
    - 2 bit di intestazione (i primi della sequenza) seguiti da
    - N bit di indirizzo della memoria.
- Gli N bit permettono di costruire un indirizzo di memoria. All'indirizzo di memoria è memorizzato il messaggio da 8 bit che deve essere indirizzato verso un canale di uscita. I due bit di intestazione identificano il canale d'uscita (Z0, Z1, Z2 o Z3) sul quale deve essere indirizzato il messaggio.
- Se il numero di bit di N è inferiore a 16, l'indirizzo viene esteso con 0 sui bit più significativi.
- Tutti i bit su W devono essere letti sul fronte di salita del clock.
- La sequenza di ingresso è valida quando il segnale START è alto (=1) e termina quando il segnale START è basso (=0).
- Le uscite Z0, Z1, Z2 e Z3 sono inizialmente 0. I valori rimangano inalterati eccetto il canale sul quale viene mandato il messaggio letto in memoria; i valori sono visibili solo quando il valore di DONE è 1.
- Quando il segnale DONE è 0 tutti i canali Z0, Z1, Z2 e Z3 devono essere a zero (32 bit a 0). Contemporaneamente alla scrittura del messaggio sul canale, il segnale DONE passa da 0 passa a 1 e rimane attivo per un solo ciclo di clock (dopo 1 ciclo di clock DONE passa da 1 a 0).
- 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.
- Il modulo deve essere progettato considerando che prima del primo START=1 (e prima di richiedere il corretto funzionamento del modulo) verrà sempre dato il RESET (RESET=1).
- Ogni qual volta viene dato il segnale di RESET (RESET=1), il modulo viene re-inizializzato.

Per informazioni più dettagliate e per consultare eventuali esempi di funzionamento, consultare la Specifica di Progetto fornita sulla pagina WeBeep del corso.

2.Architettura

2.1 Scelte progettuali

Il componente è stato pensato come un unico modulo contente due processi principali:

1. Il primo si occupa di gestire l'aggiornamento e il reset dei registri, rappresentando quindi la parte sequenziale
2. Il secondo è invece una macchina a stati finiti che determina stato prossimo e variazioni dei valori nei registri sulla base dello stato corrente

La parte sequenziale consiste in un processo, sensibile a i_rst e a i_clk, che implementa la gestione della fase di reset della FSM e del passaggio allo stato prossimo.

Quando i_rst assume valore ‘1’ i valori dei registri, tra cui state_reg che gestisce lo sviluppo della FSM, vengono asseriti e si verifica quindi il totale reset della FSM come da specifica.
Quando invece i_clk è sul fronte di salita (il processo è sensibile a i_clk ma le effettive modifiche avvengono solo a fronte di rising_edge(i_clk)) avviene l’aggiornamento dei registri e il passaggio allo stato prossimo grazie al registro state_next.

La FSM, che sarà spiegata nel dettaglio a breve, si occupa invece di realizzare effettivamente quanto richiesto dalla specifica e avrà quindi la capacità di: riconoscere indirizzo e canale forniti in ingresso, accedere correttamente alla cella di memoria indicata ed estrarne la parola corretta, aggiornare i valori delle uscite e il valore di DONE in modo coerente e, infine, ritornare disponibile per una nuova lettura delle istruzioni dall’ingresso.

2.2 FSM

La FSM si compone di 11 stati, di seguito elencati e spiegati nel dettaglio:

2.2.1 RST

Stato iniziale in cui si attende che il segnale i_start venga portato a 1. Ogni volta che il segnale i_rst viene alzato si ritorna in questo stato.

2.2.2 GET_CHAN

In questo stato si effettua la lettura dei due bit che identificano il canale su cui scrivere la parola ottenuta dalla memoria.
Si usano i registri bit_read per tenere conto di quanti bit di i_w siano stati letti e w_reg per conservare nel ciclo di clock successivo il valore letto da i_w. La macchina poi, attraverso un costrutto if-else, riconosce l’indirizzo corretto e lo scrive nel registro curr_channel.

2.2.3 GET_ADDR

Dopo aver letto i primi due bit di i_w la FSM si sposta in questo stato dove, fin quando start_reg (registro che conserva il valore di i_start per il ciclo di clock successivo) ha come valore ‘1’, viene effettuata la lettura dell’indirizzo di memoria a cui accedere. Quest’ultimo viene salvato in un registro chiamato tmp_addr.

2.2.4 FIX_ADDR

L’indirizzo così come è stato salvato in tmp_addr è incompleto: è salvato al contrario, in quanto l’ultimo bit letto prima che i_start venga asserito deve in realtà essere salvato come bit più significativo, e se i_start ha avuto valore ‘1’ per meno di 18 cicli di clock non è nemmeno esteso a 16 bit.
FIX_ADDR si occupa di entrambi gli aspetti: tmp_addr viene ribaltato e copiato in un nuovo registro, chiamato fixed_addr, che è inizialmente una stringa di 16 zeri. Questo garantisce sia l’estensione a 16 bit che la corretta rappresentazione dei bit in ingresso convertiti nell’indirizzo di memoria.

A causa del limite di massimo 20 cicli di clock tra i_start=’0’ e o_done=’1’, si è rivelato necessario rendere molto più snella e veloce la fase di “correzione” di tmp_addr. Per questo motivo l’iniziale implementazione di FIX_ADDR, che prevedeva la modifica di un solo bit per ciclo di clock, è stata sostituita da una che corregge 2 bit per ciclo di clock (più un eventuale ulteriore ciclo di clock per correggere l’ultimo bit rimasto in caso di indirizzi di lunghezza dispari). Questo ha ridotto il numero massimo di cicli di clock richiesti da FIX_ADDR da 16 a 8.

2.2.5 WRITE_ADDR

Questo stato si occupa di copiare su o_mem_addr_next l’indirizzo di memoria ricevuto in ingresso e di modificare i bit o_mem_en_next e o_mem_we_next per preparare la memoria alla lettura dei dati.
Inoltre imposta il reset di tmp_addr e fixed_addr (asserisce tmp_addr_next e fixed_addr_next) in quanto non più utili nel corso del processo.

2.2.6 WAIT_MEMO_1
Assieme a WAIT_MEMO_2 servono solo alla FSM per prendere tempo in attesa che la memoria riceva la richiesta di lettura e scriva il dato sull'ingresso del modulo HW i_mem_data.

2.2.7 WAIT_MEMO_2
Si veda WAIT_MEMO_1.

2.2.8 READ_MEMO
Questo stato si occupa di copiare in curr_word ciò che la memoria ha inserito in i_mem_data. Inoltre asserisce o_mem_en, rendendo di nuovo chiuso il canale di comunicazione tra memoria esterna e modulo HW.

2.2.9 SELECT_CHAN
In base al valore contenuto in curr_channel, questo stato prepara sul canale indicato il dato appena estratto dalla memoria (salvandolo in o_zx_next, dove x è il numero del canale). SELECT_CHAN si occupa anche di tenere aggiornati i registri di tutti e 4 i canali per permettere, quando o_done='1', la corretta visualizzazione dell'ultima parola copiata su ciascun specifico canale.
Infine in questo stato viene anche predisposta la modifica del valore di o_done, in quanto si ha o_done_next='1'.

2.2.10 REPEAT
Questo stato innesca la ripetizione della manovra di lettura di canale e indirizzo dall'ingresso. Imposta il valore di o_done_next='0' (in questo modo la richiesta di o_done='1' per un solo ciclo di clock viene rispettata) e prepara su tutti i registri che regolano le parole in uscita sui canali la stringa "00000000" come richiesto dalla specifica. Asserisce bit_read_next e curr_chan_next per preparare una nuova lettura dell'ingresso.

2.2.11 GET_CHAN_2
Seconda versione di GET_CHAN usata solo nel ciclo di clock successivo a REPEAT e necessaria per rientrare in modo corretto nella sequenza di stati della FSM senza perdere il primo valore di i_w in seguito a i_start='1'. A livello operativo esegue le stesse manovre di GET_CHAN originale.

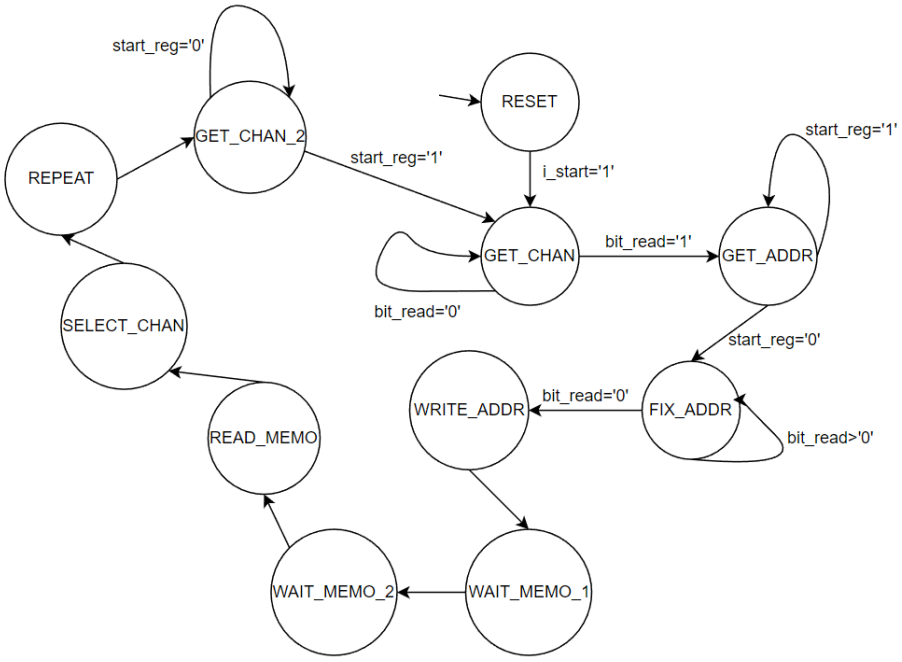


Grafico della macchina a stati

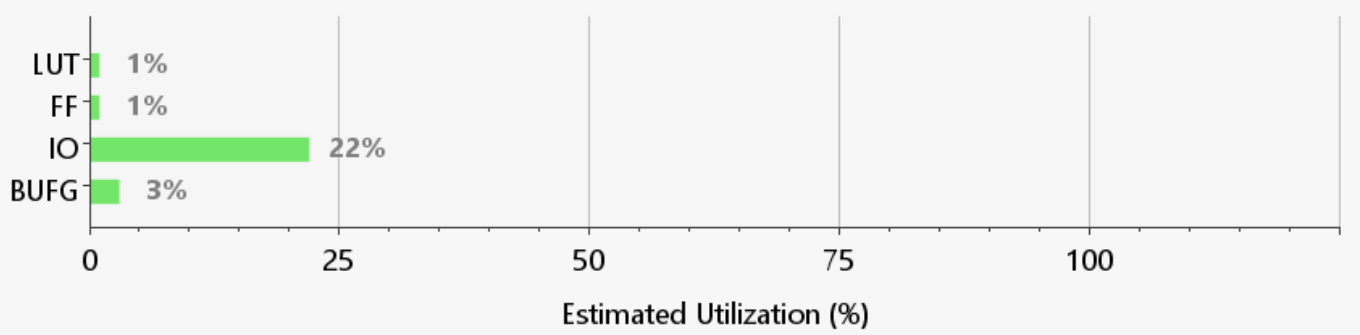
3.Risultati sperimentali

3.1 Report sintesi

Il componente viene sintetizzato senza alcun tipo di errore e tutti i testbench provati terminano la simulazione con successo sia in presintesi che in postsintesi (functional e timing).

Di seguito sono riportate la tabella e il grafico di occupazione delle risorse, dove si può notare un uso marginale di tutti i componenti disponibili. Il modulo viene sintetizzato mediante l'uso di 136 Lookup Tables, 147 Flip Flop, 1 Global Buffer e 63 porte di input/output.

Resource	Estimation	Available	Utilization %
LUT	136	134600	0.10
FF	147	269200	0.05
IO	63	285	22.11
BUFG	1	32	3.13



Di seguito è riportato il report_timing del componente sintetizzato:

```
Timing Report

Slack (MET) :             96.154ns  (required time - arrival time)
  Source:                 bit_read_reg[2]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            FSM_onehot_state_reg_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:        3.464ns  (logic 0.999ns (28.839%)  route 2.465ns (71.161%))
  Logic Levels:           3  (LUT6=3)
  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=147, unplaced)       0.584     2.424    i_clk_IBUF_BUFG
                         FDCE                                         r  bit_read_reg[2]/C
```

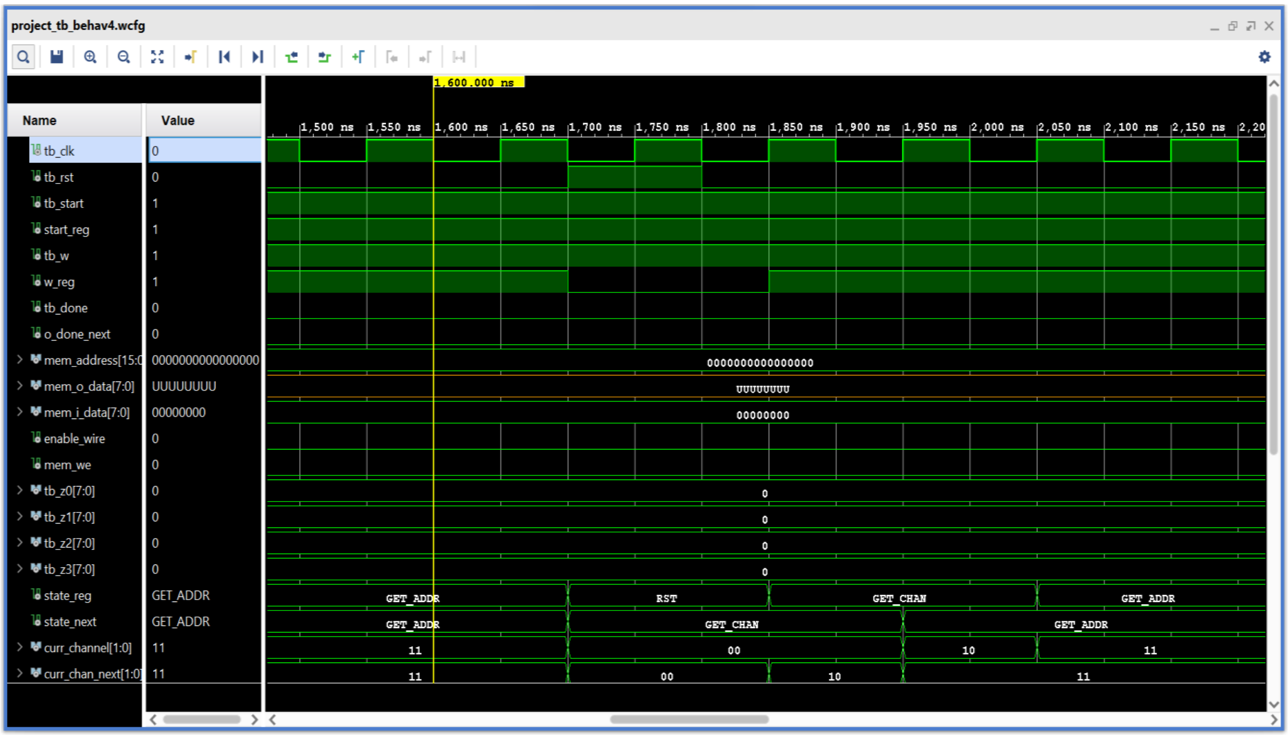
```
-------------------------------------------------------------------    -------------------
                      FDCE (Prop_fdce_C_Q)         0.456     2.880 f  bit_read_reg[2]/Q
                      net (fo=33, unplaced)        1.037     3.917    bit_read[2]
                      LUT6 (Prop_lut6_I0_O)        0.295     4.212 r  bit_read[4]_i_3/O
                      net (fo=2, unplaced)         0.460     4.672    bit_read[4]_i_3_n_0
                      LUT6 (Prop_lut6_I4_O)        0.124     4.796 r  FSM_onehot_state_reg[10]_i_3/O
                      net (fo=1, unplaced)         0.449     5.245    FSM_onehot_state_reg[10]_i_3_n_0
                      LUT6 (Prop_lut6_I0_O)        0.124     5.369 r  FSM_onehot_state_reg[10]_i_1/O
                      net (fo=11, unplaced)        0.519     5.888    FSM_onehot_state_reg[10]_i_1_n_0
                      FDPE                                          r  FSM_onehot_state_reg_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=147, unplaced)       0.439   102.100    i_clk_IBUF_BUFG
                      FDPE                                          r  FSM_onehot_state_reg_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_state_reg_reg[0]
-------------------------------------------------------------------
                      required time                        102.041
                      arrival time                          -5.888
-------------------------------------------------------------------
                      slack                                 96.154
```

3.2 Testbench utilizzati

Dall'analisi delle specifiche del progetto e dalla struttura del modulo e della FSM in particolare, sono emersi alcuni punti critici che meritano un approfondimento maggiore e per questo si è rivelato necessario avvalersi di alcuni testbench (in parte ottenuti da WeBeep e in parte creati ad hoc oppure condivisi da altri studenti) per verificare il corretto funzionamento del componente.

3.2.1 Reset asincrono durante l'elaborazione

Chiaramente il primo punto critico è la verifica dell'effettivo funzionamento del meccanismo di reset della FSM e conseguentemente del componente. Alcuni dei testbench forniti su WeBeep mi hanno permesso di affermare con una certa sicurezza che, ad elaborazione non ancora iniziata, i_rst='1' non provoca problemi all'esecuzione della FSM in quanto il suo comportamento è del tutto coerente con la specifica fornita.
Si è reso comunque necessario implementare un testbench aggiuntivo, modificando leggermente uno dei testbench forniti, per verificare che il modulo HW si comporti nel modo atteso anche a fronte di i_rst='1' nel mezzo della sua esecuzione.

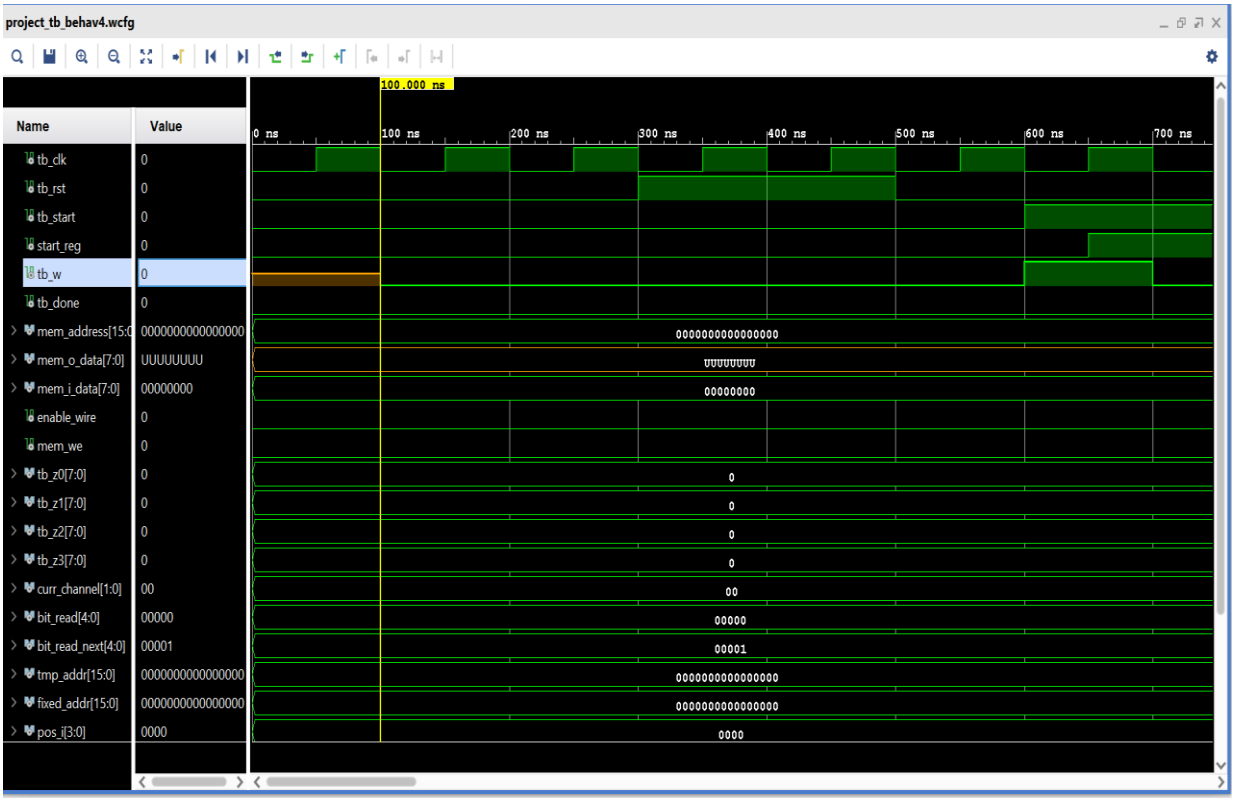


Com'è possibile osservare nell'immagine, a 1700 ns il modulo riceve un segnale di reset asincrono e si comporta nel modo atteso: lo stato corrente diventa RST, o_done e i quattro canali di uscita sono correttamente inizializzati a 0. Inoltre la FSM, una volta che i_rst='0', torna alla sua normale esecuzione riprendendo a leggere l'ingresso seriale e porta a termine l'elaborazione delle istruzioni successive, come possibile osservare dal messaggio seguente:

Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 4300 ns Iteration: 0 Process: /project_tb/testRoutine File: C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_10/imports/Desktop/tb_reset.vhd
$finish called at time : 4300 ns : File "C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_10/imports/Desktop/tb_reset.vhd" Line 162

3.2.2 Segnali con valore definito all'inizio della simulazione

Altro motivo di preoccupazione è quello che ad inizio simulazione i segnali possano presentarsi con valori indefiniti, segno di un errore nella loro inizializzazione. Questo porterebbe poi a problematiche nelle fasi iniziali del funzionamento del componente che andrebbe ovviamente incontro ad errori dovuti all'incapacità di valutare correttamente le condizioni di transizione fra stati che sono legate ai valori dei segnali.
Il testbench tb_example23_agg (scaricato dalla pagina WeBeep del corso di Fornaciari) fornisce in avvio dei segnali con valori indefiniti che il componente deve prontamente inizializzare nel primo ciclo di clock disponibile. A seguire l'immagine che mostra come ciò avviene in simulazione (functional post-synthesis).



La simulazione del test bench termina poi col seguente messaggio che testimonia la correttezza delle operazioni del componente in questa situazione critica:

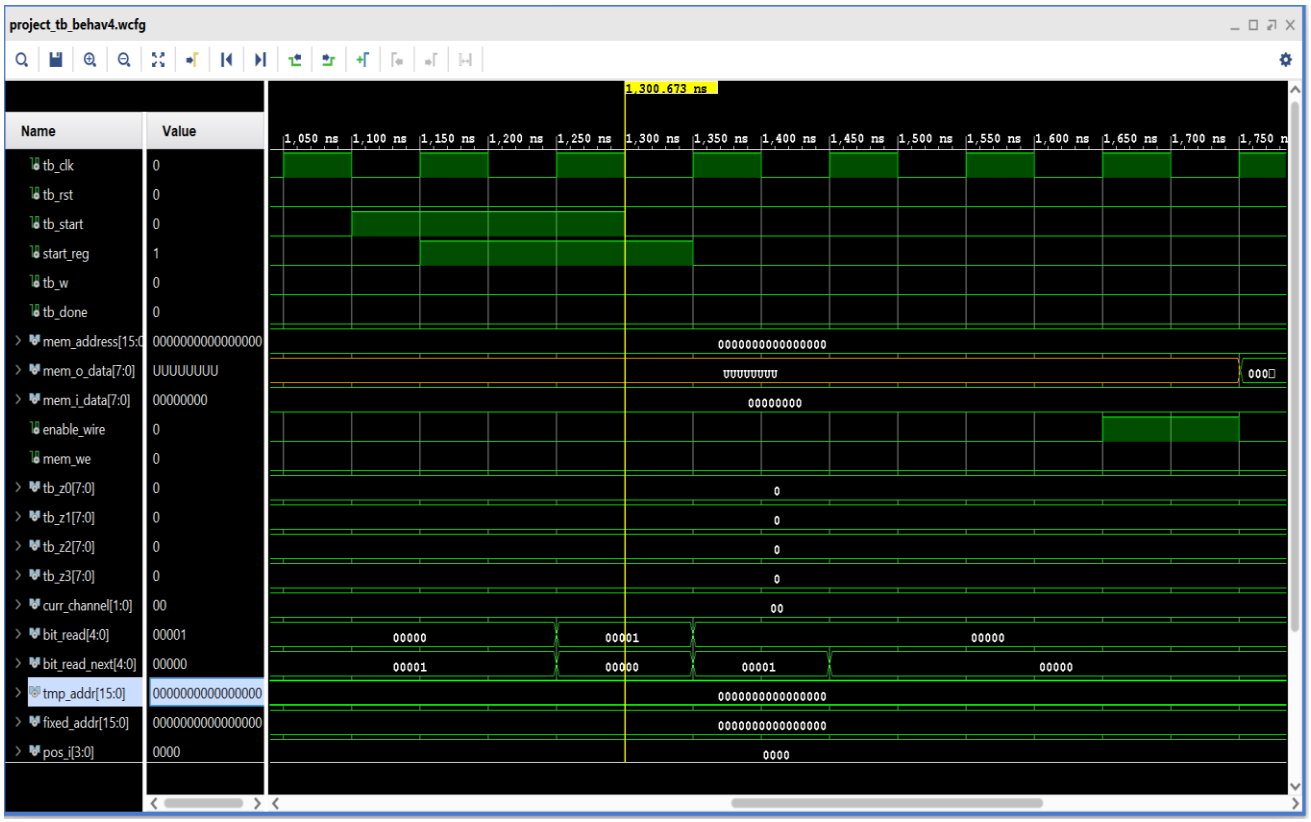
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)

Time: 4400100 ps Iteration: 0 Process: /project_tb/testRoutine File: C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_9/imports/Downloads/tb_example23_agg.vhd

$finish called at time : 4400100 ps : File "C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_9/imports/Downloads/tb_example23_agg.vhd" Line 156

3.2.3 Indirizzo in memoria è “0000000000000000”

In questo caso il componente viene testato nella situazione in cui, dopo aver letto il canale su cui scrivere la parola ottenuta dalla memoria, il segnale i_start viene immediatamente asserito.
In alcuni dei testbench forniti su WeBeep ci si imbatte in questa situazione: ho quindi scelto, a titolo puramente esemplificativo perché anche in tutti gli altri casi il funzionamento si rivela corretto, il testbench denominato tb_7. Il risultato della simulazione (functional post-synthesis) è qui sotto riportato.

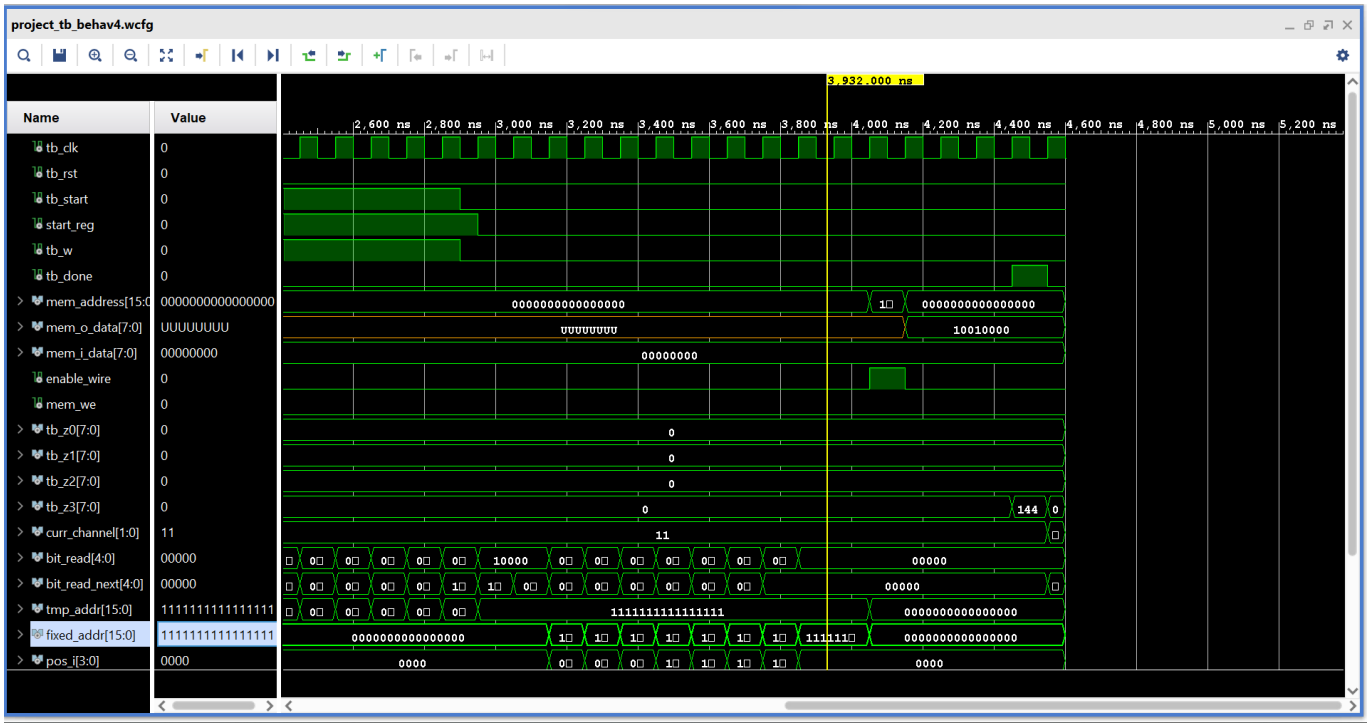


Come si può osservare dall’immagine, tb_start=’1’ solo per due cicli di clock tra 1150 ns e 1250 ns, a 1350ns tb_start=’0’. Questo implica che il componente deve identificare il canale su cui scrivere e poi ottenere dalla memoria la parola all’indirizzo “0000000000000000”. Nello svolgimento delle sue operazioni la FSM si comporta nelle modalità attese selezionando il canale 00 e l’indirizzo (fixed_addr) corretto.
Il testbench termina col seguente messaggio, evidenziando la correttezza del funzionamento della FSM:

Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 2200100 ps Iteration: 0 Process: /project_tb/testRoutine File: C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_8/imports/Downloads/tb_7.vhd
$finish called at time : 2200100 ps : File "C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_8/imports/Downloads/tb_7.vhd" Line 162

3.2.4 Indirizzo di memoria è di 16 bit/o_done='0' → canali in uscita asseriti

Questo è il caso in cui i_start='1' per 18 cicli di clock consecutivi: in questo caso la computazione del componente è quella più lunga possibile (quantomeno in termini di cicli di clock). Eseguo quindi un test bench che mi permetta di verificare se tra i_start='0' e o_done='1' intercorrano al massimo 19 cicli di clock.
Il test bench scelto è il tb_6 fornito su WeBeep che contiene in una sua parte l'indicazione di un indirizzo a 16 bit. Di seguito viene fornita l'immagine della simulazione (functional post-synthesis).



L'immagine mostra in modo chiaro che in fixed_addr c'è un indirizzo di 16 bit ("1111111111111111") ricevuti da i_w, si tratta quindi del caso che richiede la computazione più lunga in termini di cicli di clock.
E' quindi possibile verificare che la specifica viene rispettata anche in questo caso, in quanto tra i_start='0' e o_done='1' intercorrono 15 (15,5 nello specifico) cicli di clock, 4 (o 3,5) in meno rispetto al limite massimo.

Questo testbench mi permette inoltre di certificare che, nel momento in cui o_done='0' tutti i canali in uscita presentano la stringa "00000000" come è possibile vedere a 4500 ns.

Inoltre il seguente messaggio certifica la correttezza del procedimento in questo ennesimo caso di criticità.

Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 4600100 ps Iteration: 0 Process: /project_tb/testRoutine File:
C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_7/imports/Downloads
/tb_6.vhd
$finish called at time : 4600100 ps : File
"C:/Users/PC/PROGETTO_RETI_LOGICHE_V2/PROGETTO_RETI_LOGICHE_V2.srcs/sim_7/imports/Download
s/tb_6.vhd" Line 162

3.2.5 Test generici (riscrittura su canali già modificati, stress test, etc.)
Avendo eseguito correttamente la simulazione sia in pre che in post sintesi con i 9 testbench forniti e con il testbench aggiuntivo da me ideato per il reset asincrono, posso con un buon grado di certezza ritenere che il componente sia implementato in modo sufficientemente solido nei confronti di numerosi bug minori e anche in situazioni di stress dovute al susseguirsi dell'intera sequenza della FSM.

4.Conclusioni

Risultato della sintesi
Il componente sintetizzato supera correttamente tutti i test specificati nelle tre simulazioni: Behavioral, Post-Synthesis Functional e Post-Synthesis Timing. Nel generare la sintesi della rete non vengono presentati problemi od errori di nessun tipo e non è necessario l'utilizzo di latch grazie alle corrette inizializzazioni ed aggiornamenti dei segnali utilizzati nel componente.

La decisione di lavorare con un singolo modulo su due processi distinti si è rivelata molto pratica e mi ha permesso di concentrarmi volta per volta su delle porzioni dell'intero progetto più facilmente gestibili e di dimensioni ridotte.

L'elevato numero di testbench forniti ha reso più semplice il lavoro di testing e debugging in quanto molte, se non quasi tutte, le problematiche emerse si sono rivelate facilmente individuabili all'interno dei testbench stessi. Questo mi ha permesso di risparmiare tempo nella creazione di testbench personalizzati e di concentrarmi maggiormente sull'aumento dell'efficienza del componente, portandomi a considerazioni, come quella precedentemente riportata circa il funzionamento dello stato FIX_ADDR, che hanno notevolmente ridotto la complessità e la lunghezza della computazione.