Progetto di Reti Logiche A. A. 2023/24

Marco Zanzottera

20 marzo 2024

Indice

1 Introduzione

In questo progetto di Reti Logiche verrà sviluppato un modulo hardware, in grado di soddisfare la specifica fornita mantenendo un tempo di clock inferiore a 20 ns.

1.1 Specifica

Memoria Viene fornita una memoria con $2^{16} = 65536$ indirizzi di memoria, indicizzabile da 0 a 65535 tramite il segnale `o_mem_addr` di 16 bit. Ogni cella di memoria contiene una parola da 8 bit (1 byte).

È possibile accedere alla memoria in lettura e in scrittura: i segnali di controllo coinvolti sono `o_mem_en` e `o_mem_we`.

Si hanno poi due segnali `i_mem_data` e `o_mem_data` da 8 bit, rispettivamente utilizzati per leggere o scrivere il dato.

Il dato viene letto dalla posizione `o_mem_addr` e portato su `i_mem_data` quando, sul fronte di salita del clock, `o_mem_en`=1 e `o_mem_we`=0.

Il dato presente in `o_mem_data` viene scritto in posizione `o_mem_addr` quando, sul fronte di salita del clock, `o_mem_en`=1 e `o_mem_we`=1.

Quando `o_mem_en`=0 non si stanno effettuando letture o scritture sulla memoria.

Ingressi Oltre ai segnali in ingresso relativi al collegamento con la memoria (contraddistinti dal prefisso `i_`), vengono forniti dei segnali in ingresso al sistema da sviluppare:

- `i_clk`, segnale di clock con un periodo di 20 ns;
- `i_rst`, segnale di reset che quando posto a 1 inizializza il sistema prima di ricevere il segnale di inizio, in modo asincrono: non dipende dal segnale di clock;
- `i_start`, segnale di inizio che viene posto a 1 quando il sistema deve iniziare a operare sulla memoria;
- `i_k`, segnale a 10 bit che rappresenta in binario naturale la lunghezza della sequenza di byte sulla quale il sistema deve operare. È possibile operare su un numero di elementi che va da 0 a $2^{10} - 1 = 1023$;
- `i_add`, segnale a 16 bit che rappresenta l'indirizzo di memoria del primo elemento della sequenza;

Uscite Oltre ai segnali in uscita relativi al collegamento con la memoria (contraddistinti dal prefisso `o_`), viene fornito un segnale in uscita al sistema da sviluppare:

- `o_done`, segnale che viene posto a 0 quando `i_rst`=1 e a 1 quando il sistema ha terminato di operare sulla sequenza di `i_k` elementi a partire dall'indirizzo di memoria `i_add`. A seguito di `o_done`=1 il segnale `i_start` viene abbassato ed è possibile porre nuovamente `o_done`=0, rendendo il sistema pronto per operare su una nuova sequenza;

Comportamento A seguito dell'inizializzazione con segnale di reset `i_rst`, al sistema viene fornito il segnale di start `i_start` = 1. Fino a quando `i_start` = 1, è garantito che `i_k` e `i_add` non cambino.

Se il sistema ha ricevuto in ingresso `i_k` = 0, nulla deve essere scritto sulla memoria.

Se il sistema ha ricevuto in ingresso `i_k` > 0, procederà a leggere e scrivere sulla memoria secondo il seguente criterio:

1. Viene letto dalla memoria il dato all'indirizzo specificato da `i_add`;
2. Se viene letto 0 come primo valore, si scrive il valore di credibilità 0 (`00000000`) all'indirizzo di memoria `i_add` + 1;
3. Se viene letto come primo valore un numero > 0, si scrive il valore di credibilità 31 (`00011111`) all'indirizzo `i_add` + 1;
4. Si prosegue leggendo l'indirizzo `i_add` + 2 dalla memoria se è minore di `i_add` + 2 · `i_k`, altrimenti si termina;
5. Se viene letto come valore un numero > 0, si scrive il valore di credibilità 31 (`00011111`) all'indirizzo successivo a quello da cui è avvenuta la lettura;
6. Se viene letto come valore 0, lo si sovrascrive con il precedente valore letto e si scrive il valore di credibilità precedente decrementato di 1 all'indirizzo successivo a quello da cui è avvenuta la lettura. Se il precedente valore di credibilità era 0, rimane 0;
7. Si prosegue leggendo i byte a indirizzi di memoria a distanza 2, fino a quando sono minori di `i_add` + 2 · (`i_k`), altrimenti si termina;
8. Si ritorna al passo 5.

Esempio `i_add` = `0x0002` `i_k` = `0x09`

Prima	
addr	word
0x0000	0x13
0x0001	0x00
0x0002	0x00
0x0003	0x00
0x0004	0x47
0x0005	0x00
0x0006	0x64
0x0007	0x00
0x0008	0xDA
0x0009	0x00
0x000A	0x37
0x000B	0x00
0x000C	0x00
0x000D	0x00
0x000E	0x00
0x000F	0x00
0x0010	0x00
0x0011	0x00
0x0012	0x00
0x0013	0x00
0x0014	0x00
0x0015	0x00
0x0016	0x00
0x0017	0x00
0x0018	0x00
0x0019	0x00
0x001A	0x23
0x001B	0x00
0x001C	0x59
0x001D	0x00
0x001E	0x33
0x001F	0x00
0x0020	0x00
0x0021	0x00
0x0022	0x00
0x0023	0x00

Dopo	
addr	word
0x0000	0x13
0x0001	0x00
0x0002	**0x00**
0x0003	**0x00**
0x0004	**0x47**
0x0005	**0x1F**
0x0006	**0x64**
0x0007	**0x1F**
0x0008	**0xDA**
0x0009	**0x1F**
0x000A	**0x37**
0x000B	**0x1F**
0x000C	**0x37**
0x000D	**0x1E**
0x000E	**0x37**
0x000F	**0x1D**
0x0010	**0x37**
0x0011	**0x1C**
0x0012	**0x37**
0x0013	**0x1B**
0x0014	0x00
0x0015	0x00
0x0016	0x00
0x0017	0x00
0x0018	0x00
0x0019	0x00
0x001A	0x23
0x001B	0x00
0x001C	0x59
0x001D	0x00
0x001E	0x33
0x001F	0x00
0x0020	0x00
0x0021	0x00
0x0022	0x00
0x0023	0x00

2 Architettura

Il progetto è realizzato ad alto livello con un approccio strutturale: tramite VHDL vengono specificati i moduli di cui è composto e come questi siano collegati tra loro e con ingressi e uscite esterne.

Ogni modulo implementa una parte delle funzionalità necessarie alla realizzazione del sistema, ed è realizzato con uno o più degli approcci tra dataflow, strutturale e comportamentale, in base alla complessità del modulo considerato.

I moduli utilizzati sono 8, dei quali è possibile vederne i collegamenti dallo schema in Figura 2.1.

2.1 Modulo 1 - State Manager

Il primo modulo (Figura 2.2) si occupa della gestione dello stato in cui l'intero sistema si può trovare. È realzzato con un approccio comportamentale, è quindi una collezione di process che implementano una macchina a stati.

Il primo process si occupa di aggiornare lo stato corrente in relazione al fronte di salita del clock, mentre il secondo process implementa la funzione di transizione e il terzo la funzione di uscita.

La macchina a stati è una macchina di Mealy, in quanto le uscite non dipendono dal solo stato corrente, ma anche dall'ingresso.

In Figura 2.3 sono riportati gli stati, e siccome la macchina a stati ha più di un ingresso, tale schema è da interpretarsi in maniera diversa dai classici diagrammi degli stati.

Il segnale `i_rst` di reset, asincrono, può portare in qualsiasi momento la macchina nello stato IDLE. Tutte le altre transizioni di stato avvengono invece sul fronte di salita del ciclo di clock.

Se un arco non ha etichette, la transizione di stato avviene sempre. Se un arco ha un'etichetta, la transizione di stato allo stato indicato avviene solo se sul fronte di salita i segnali indicati, con le opportune operazioni logiche, vengono valutati logicamente come veri. Altrimenti, se nessuna etichetta di un arco dello stato corrente è vera, lo stato della macchina non cambia.

Gli stati della macchina sono 6:

- **IDLE**: Lo stato in cui la macchina è portata a seguito di inizializzazione con `i_rst`, è uno stato passivo che indica la predisposizione della macchina a poter iniziare ad effettuare un'elaborazione;
- **READ**: Lo stato in cui si entra a seguito di un segnale `i_start` alto, oppure a seguito dello stato WRITECREDIBILITY. È uno dei quattro stati in cui la macchina sta effettuando l'elaborazione (READ, EVALREAD, WRITEVALUE, WRITECREDIBILITY). In questo stato si preparano i segnali necessari a poter richiedere alla memoria la lettura di un byte da un certo indirizzo;
- **EVALREAD**: Lo stato in cui si entra dopo READ se `i_finish` è basso. Siccome la memoria è sincrona, per poterne leggere il contenuto è necessario preparare i segnali all'interrogazione in un certo periodo di clock, per poi ottenere il risultato nel ciclo di clock successivo. In questo stato viene valutato se è necessario sovrascrivere il valore letto: per specifica deve essere fatto solo se il valore letto è 0. La valutazione è effettuata mediante

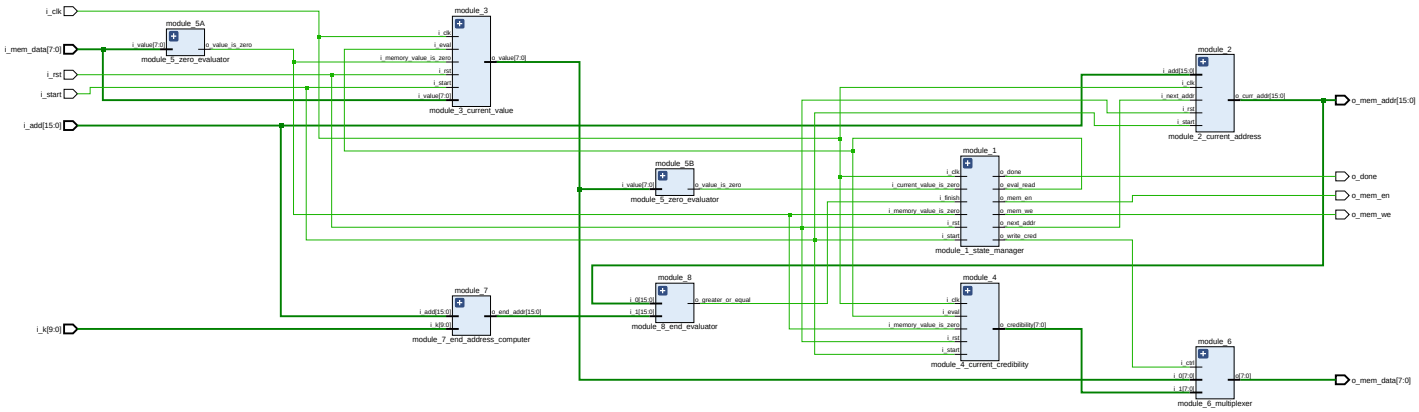

Figura 2.1: Schema architetturale del progetto

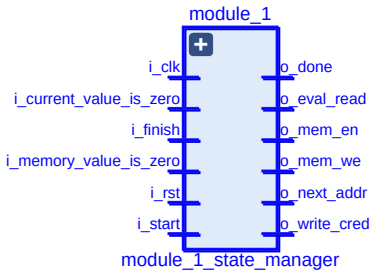

Figura 2.2: Interfaccia del modulo 1

i segnali `i_memory_value_is_zero` e `i_current_value_is_zero`, che devono essere contemporaneamente alto e basso per portare nello stato WRITEVALUE, altrimenti la transizione porta allo stato WRITECREDIBILITY. Sempre in base a questa condizione viene scelto se il prossimo indirizzo di memoria non deve cambiare oppure deve avanzare di una posizione. Questo stato viene utilizzato anche per segnalare ad altri moduli del sistema che la memoria sta esponendo un certo valore in lettura, permettendo ad esempio la sua memorizzazione;

- **WRITEVALUE**: Lo stato in cui si sovrascrive il valore 0 in memoria con un valore diverso da 0 memorizzato dal sistema. La transizione di questo stato è incondizionata verso WRITECREDIBILITY;
- **WRITECREDIBILITY**: Lo stato in cui si scrive il valore di credibilità in memoria;
- **DONE**: Lo stato in cui si segnala che il processo è terminato, utilizzando l'apposito segnale `o_done`. Si raggiunge questo stato dallo stato di READ, a seguito di un segnale `i_finish` alto che determina la fine dell'elaborazione, generato da un altro modulo del sistema. Tramite questo stato si può tornare in stato di IDLE a seguito del segnale `i_start` basso, rendendo il sistema nuovamente pronto a gestire una nuova elaborazione;

Il modulo, per gestire gli altri moduli, produce in uscita diversi segnali, riportati in Tabella 2.1.

stato	o_done	o_mem_we	o_mem_en	o_write_cred	o_next_addr	o_eval_read
IDLE	0	-	0	-	0	0
READ	0	0	1	-	0	0
EVALREAD	0	-	0	-	$\overline{\text{i\_memory\_value\_is\_zero}}$ or i_current_value_is_zero	1
WRITEVALUE	0	1	1	0	1	0
WRITECREDIBILITY	0	1	1	1	1	0
DONE	1	-	0	-	0	0

Tabella 2.1: Uscite del modulo 1

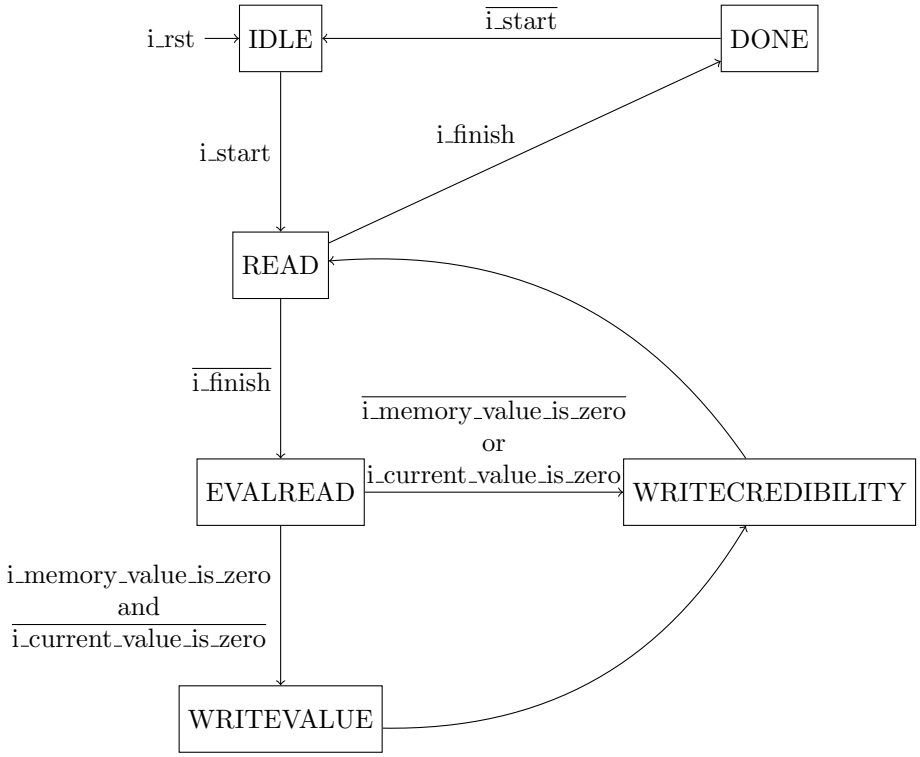

Figura 2.3: Diagramma degli stati del modulo 1

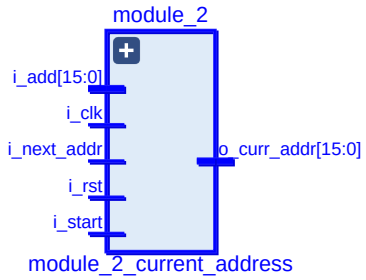

Figura 2.4: Interfaccia del modulo 2

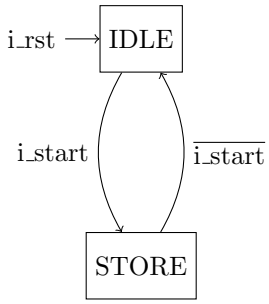

Figura 2.5: Diagramma degli stati del modulo 2

2.2 Modulo 2 - Current Address

Il secondo modulo (Figura 2.4) si occupa di inizializzare e gestire l'avanzamento dell'indirizzo di memoria corrente. È realzzato con un approccio comportamentale, è quindi una collezione di process che implementano una macchina a stati.

Il primo process si occupa di aggiornare lo stato corrente e l'indirizzo di memoria in relazione al fronte di salita del clock, mentre il secondo process implementa la funzione di transizione e la gestione dell'inizializzazione e dell'avanzamento dell'indirizzo di memoria corrente.

La macchina a stati è una macchina di Moore, in quanto l'uscita è l'indirizzo corrente.

I due possibili stati per la macchina sono uno stato di IDLE e uno di STORE, come mostrato in Figura 2.5.

Nello stato IDLE il modulo è inattivo, con indirizzo corrente zero. Passando dallo stato IDLE allo stato store, viene immagazzinato nell'indirizzo corrente il valore esposto da `i_add`. Nello stato STORE il modulo è attivo, e se il segnale `i_next_addr` è alto l'indirizzo corrente viene incrementato di una posizione sul fronte di salita del clock, altrimenti non subisce variazioni.

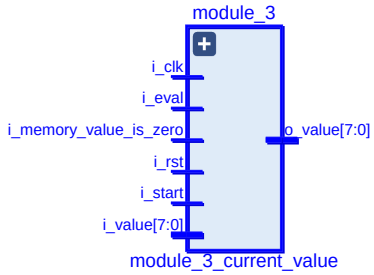

Figura 2.6: Interfaccia del modulo 3

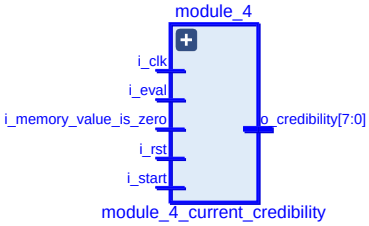

Figura 2.7: Interfaccia del modulo 4

2.3 Modulo 3 - Current Value

Il terzo modulo (Figura 2.6) si occupa di inizializzare e gestire l'ultimo valore letto dalla memoria diverso da zero. È realzzato con un approccio comportamentale, è quindi una collezione di process che implementano un registro di memoria a 8 bit.

Il primo process si occupa di aggiornare il valore corrente in relazione al fronte di salita del clock, mentre il secondo process implementa la gestione del valore corrente, sovrascrivendo il precedente solo quando `i_eval` è alto e il valore letto da memoria non è zero.

Se `i_start` è basso, il valore corrente è costantemente impostato a zero. Il modulo opera come descritto solo quando `i_start` è alto.

2.4 Modulo 4 - Current Credibility

Il quarto modulo (Figura 2.7) si occupa di inizializzare e gestire il valore di credibilità corrente. È realzzato con un approccio comportamentale, è quindi

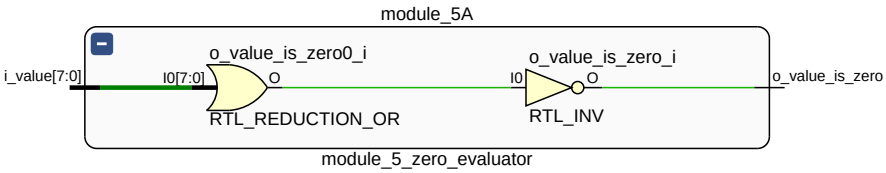

Figura 2.8: Interfaccia e architettura del modulo 5

una collezione di process che implementano un registro di memoria a 8 bit e la logica con la quale aggiornarne il contenuto.

Il primo process si occupa di aggiornare la credibilità corrente in relazione al fronte di salita del clock, mentre il secondo process implementa la gestione della credibilità corrente.

Se `i_start` è basso, la credibilità corrente è impostata a zero. Se invece `i_start` è alto, viene valutato il segnale `i_eval`: nel caso in cui sia basso, il valore di credibilità non cambia, mentre se è alto si considera il segnale `i_memory_value_is_zero`. Se il segnale `i_memory_value_is_zero` è alto, la credibilià viene decrementata di uno, fino ad un minimo di zero, altrimenti viene impostata a 31.

2.5 Modulo 5 - Zero Evaluator

Il quinto modulo, ricevuto in ingresso un segnle ad N (generic VHDL) bit, produce 1 in uscita sul segnale `o_value_is_zero` solo quando tutti i bit in ingresso sono 0.

È realizzato con un approccio dataflow, che è possibile vedere in Figura 2.8. È stata utilizzata la libreria use IEEE.STD_LOGIC_MISC per l'operazione or_reduce(), che effettua l'or di un vettore di bit (std_logic_vector). Senza questa

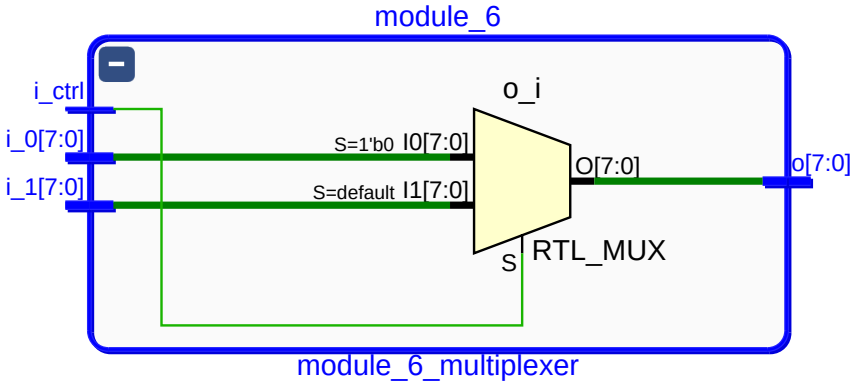

Figura 2.9: Interfaccia e architettura del modulo 6

libreria avrei creato il modulo per un N fissato e un numero fisso di or, oppure avrei utilizzato un process per mantenere il parametro generico N.

Viene utilizzato nel sistema due volte: la prima (Modulo 5A) riceve in ingresso il segnale `i_mem_data` ed è collegato in uscita a Modulo 1 e Modulo 3, la seconda (Modulo 5B) riceve in ingresso il segnale prodotto dal Modulo 3 (valore corrente) e porta l'uscita al Modulo 1.

2.6 Modulo 6 - Multiplexer

Il sesto modulo è un multiplexer che permette di selezionare quale ingresso di N (generic VHDL) bit portare in uscita su un vettore di N bit, usando un ingresso di selezione `i_ctrl`: per `i_ctrl` basso, viene portato in uscita `i_0`, mentre per `i_ctrl` alto `i_1`.

È realizzato con un approccio dataflow, che è possibile vedere in Figura 2.9.

Viene utilizzato nel sistema per selezionare cosa scrivere in memoria (segnale di controllo prodotto dal Modulo 1), se un valore precedentemente letto (ricevuto dal Modulo 3) nel caso all'ultima lettura della memoria sia stato letto 0 o una credibilità corrente (ricevuta dal Modulo 4). L'uscita è collegata direttamente a `o_mem_data`.

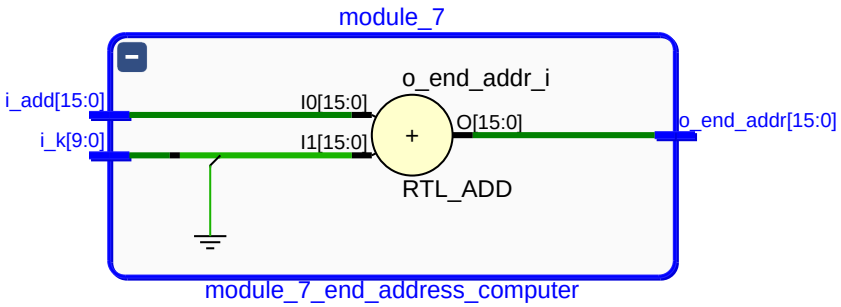

Figura 2.10: Interfaccia e architettura del modulo 7

2.7 Modulo 7 - End Address Computer

Il settimo modulo si occupa di calcolare l'indirizzo di memoria al quale il sistema deve interrompere l'elaborazione della sequenza.

È realizzato con un approccio dataflow, che è possibile vedere in Figura 2.10. Considerando che l'indirizzo finale è dato dalla formula $\texttt{i\_add} + 2\texttt{i\_k}$, prima viene effettuato uno shift logico a sinistra (equivalente a moltiplicare per due) dell'ingresso `i_k`, e poi questo viene sommato con i'ingresso `i_add`.

2.8 Modulo 8 - End Evaluator

L'ottavo modulo si occupa di valutare quando il sistema deve terminare l'elaborazione.

È realizzato con un approccio dataflow, che è possibile vedere in Figura 2.11. Il modulo effettua la sottrazione tra `i_0` e `i_1`, portando in uscita il bit più significativo negato. L'uscita ha quindi valore alto quando il risultato della sottrazione è positivo (MSB del risultato a 0), cioè quando `i_0` è maggiore o uguale a `i_1`. Se invece `i_0` è minore di `i_1`, l'uscita del modulo è 0.

Nel sistema l'ingresso `i_0` proviene dal Modulo 2 (indirizzo corrente), l'ingresso `i_1` proviene dal Modulo 7 (indirizzo finale), mentre l'uscita è collegata al Modulo 1, che la vede come segnale di ingresso `i_finish`.

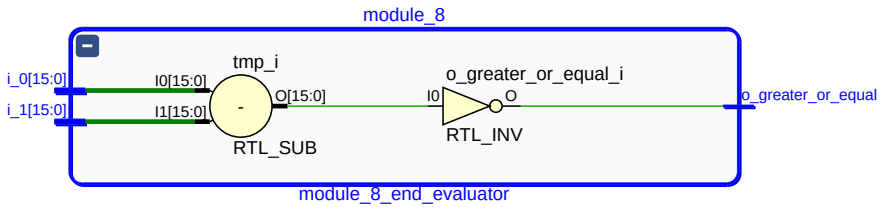

Figura 2.11: Interfaccia e architettura del modulo 8

3 Risultati sperimentali

3.1 Sintesi

La scheda utilizzata per effettuare la sintesi è quella di esempio riportata sulle regole del progetto: Artix-7 FPGA xc7a200tfbg484-1. Il progetto viene sintetizzato senza errori, e dopo aver impostato come constraint il clock a 20 ns si ottengono i seguenti risultati:

```
>report_timing

[...]
Timing Report

Slack (MET) : 16.312ns  (required time - arrival time)
[...]

>report_utilization

[...]
1. Slice Logic
--------------

+-------------------------+------+
|        Site Type        | Used |
+-------------------------+------+
| Slice LUTs*             |   69 |
|   LUT as Logic          |   69 |
|   LUT as Memory         |    0 |
| Slice Registers         |   39 |
|   Register as Flip Flop |   39 |
|   Register as Latch     |    0 |
| F7 Muxes                |    0 |
| F8 Muxes                |    0 |
+-------------------------+------+
[...]
```

Dai quali possiamo desumere che il percorso critico del componente hardware ha una latenza di $20 - 16.312 = 3.688$ ns, che quindi rispetta la specifica di operazione con ciclo di clock di 20 ns, e che il numero di Latch sintetizzati è pari a 0, in quanto spesso vengono sintetizzati Latch quando si utilizza un approccio comportamentale e non si specifica un'uscita per ogni ramo del processo.

3.2 Simulazioni

Di seguito sono riportate le simulazioni effettuate, tutte superate sia in pre-sintesi che in post-sintesi funzionale. Le simulazioni sono ottenute modificando la simulazione di esempio, per andare a verificare il comportamento del componente in casi non verificati ma previsti dalla specifica. Per ogni testbench sono quindi riportate le modifiche rispetto al primo test tramite comando:

```
diff -Naru project_tb.vhd project_tb_X.vhd
```

3.2.1 Test bench 1

Questo è il caso di test fornito, e dopo un'attenta analisi del codice, queste sono le funzionalità e casi testati:

- Mentre `i_rst` è alto, viene assicurato che `i_done` rimanga basso.
- Dopo che il segnale `i_rst` è tornato basso, ma prima che il segnale `i_start` diventi alto, viene verificato che il segnale `o_done` non venga portato alto.
- Al termine dell'elaborazione (quando `o_done` viene portato alto), si verifica che non siano contemporaneamente alti `o_mem_en` e `o_mem_we`. Non deve essere infatti possibile scrivere in memoria al termine dell'elaborazione.
- Si verifica che al termine dell'elaborazione il contenuto della memoria tra `i_add` e `i_add`$+2$`i_k`-1 sia stato modificato secondo specifica, sovrascrivendo i valori letti con la lettura precedente se questi erano 0 e scrivendo l'opportuno valore di credibilità.
- Si verifica che non appena il segnale `i_start` viene portato basso, il segnale `i_done` sia ancora alto. Nel caso di test proposto questo viene verificato correttamente, anche se il messaggio di report indicato

  `"TEST FALLITO o_done !=0 after reset before start"`

  è sbagliato, dovrebbe essere

  `"TEST FALLITO o_done !=1 after start set to 0 at elaboration end"`

Con questo test vengono testate le funzionalità di base del modulo hardware, nel caso in cui la sequenza di elaborazione includa come primo valore letto un valore diverso da 0, e ci siano successivamente alcuni valori da leggere uguali a 0, che diminuiscono il valore di credibilità (senza mai arrivare a 0). Non viene inoltre esplorata la possibilità di richiedere un'elaborazione ulteriore al termine della prima elaborazione.

3.2.2 Test bench 2 - Doppia elaborazione

Questo caso di test estende il caso precedentemente definito dal test bench 1 (3.2.1), chiedendo al sistema prima un'elaborazione a partire da `i_add` = `0x04D2` e `i_k` = `0xE`, per poi chiedere una successiva elaborazione di una sequenza a partire da `i_add` = `0x05D2` e `i_k` = `0xE`. Tra le due elaborazioni intercorre una pausa di 50 ns, nei quali il segnale `i_start` è mantenuto basso e vengono caricati i valori specificati di `i_add` e `i_k`.

```
1 --- ./project_tb.vhd
2 +++ ./project_tb_2.vhd
3 @@ -1,14 +1,14 @@
4 -- TB EXAMPLE PFRL 2023-2024
5 +- TB 2 PFRL 2023-2024
```

```
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9   use ieee.numeric_std.all;
10  use std.textio.all;
11
12 -entity project_tb is
13 -end project_tb;
14 +entity project_tb_2 is
15 +end project_tb_2;
16
17 -architecture project_tb_arch of project_tb is
18 +architecture project_tb_arch_2 of project_tb_2 is
19      constant CLOCK_PERIOD : time := 20 ns;
20      signal tb_clk : std_logic := '0';
21      signal tb_rst, tb_start, tb_done : std_logic;
22 @@ -32,6 +32,7 @@
23      signal memory_control : std_logic := '0';
24
25      constant SCENARIO_ADDRESS : integer := 1234;
26 +    constant SCENARIO_ADDRESS_2 : integer := 1490;
27
28      component project_reti_logiche is
29          port (
30 @@ -135,6 +136,14 @@
31              wait until rising_edge(tb_clk);
32          end loop;
33
34 +        for i in 0 to SCENARIO_LENGTH*2-1 loop
35 +            init_o_mem_addr<= std_logic_vector(
   to_unsigned(SCENARIO_ADDRESS_2+i, 16));
36 +            init_o_mem_data<= std_logic_vector(
   to_unsigned(scenario_input(i),8));
37 +            init_o_mem_en  <= '1';
38 +            init_o_mem_we  <= '1';
39 +            wait until rising_edge(tb_clk);
40 +        end loop;
41 +
42          wait until falling_edge(tb_clk);
43
44          memory_control <= '1';  -- Memory controlled by
               the component
45 @@ -152,6 +161,21 @@
46
47          tb_start <= '0';
48
49 +        tb_add <= std_logic_vector(to_unsigned(
   SCENARIO_ADDRESS_2, 16));
50 +        tb_k   <= std_logic_vector(to_unsigned(
   SCENARIO_LENGTH, 10));
```

```
51 +
52 +         wait until falling_edge(tb_done);
53 +
54 +         tb_start <= '1';
55 +
56 +         while tb_done /= '1' loop
57 +             wait until rising_edge(tb_clk);
58 +         end loop;
59 +
60 +         wait for 5 ns;
61 +
62 +         tb_start <= '0';
63 +
64           wait;
65
66       end process;
67 @@ -183,6 +207,22 @@
68           wait until falling_edge(tb_start);
69           assert tb_done = '1' report "TEST FALLITO o_done
                 !=0 after reset before start" severity
                failure;
70           wait until falling_edge(tb_done);
71 +
72 +         wait until rising_edge(tb_start);
73 +
74 +         while tb_done /= '1' loop
75 +             wait until rising_edge(tb_clk);
76 +         end loop;
77 +
78 +         assert tb_o_mem_en = '0' or tb_o_mem_we = '0'
    report "TEST FALLITO o_mem_en !=0 memory should not be
     written after done." severity failure;
79 +
80 +         for i in 0 to SCENARIO_LENGTH*2-1 loop
81 +             assert RAM(SCENARIO_ADDRESS_2+i) =
    std_logic_vector(to_unsigned(scenario_full(i),8))
    report "TEST FALLITO @ OFFSET=" & integer'image(i) & "
     expected= " & integer'image(scenario_full(i)) & "
    actual=" & integer'image(to_integer(unsigned(RAM(i))))
     severity failure;
82 +         end loop;
83 +
84 +         wait until falling_edge(tb_start);
85 +         assert tb_done = '1' report "TEST FALLITO o_done
     !=0 after reset before start" severity failure;
86 +         wait until falling_edge(tb_done);
87
88           assert false report "Simulation Ended! TEST
                PASSATO (EXAMPLE)" severity failure;
89       end process;
```

3.2.3 Test bench 3 - Re-inizializzazione a `i_rst=0`

Questo caso di test estende il caso precedentemente definito dal test bench 1 (3.2.1), considerando che il segnale `i_rst` possa essere portato a 1 prima che il modulo completi l'elaborazione. Scegliendo opportunamente un tempo di 540 ns tra il fronte di salita del segnale `i_start` e il fronte di salita del segnale `i_rst`, è possibile fermare l'elaborazione dopo la scrittura del settimo valore di credibilità, andando poi a verificare che la prima parte della sequenza sia stata elaborata correttamente, mentre la seconda non abbia subito modifiche. Alla salita del segnale `i_rst` il segnale `i_start` viene portato basso.

Avvertenza: questo test dipende dalla specifica implementazione qui realizzata, in quanto interrompe la sequenza con un reset tenendo conto dei nanosecondi necessari al componente per elaborare la sequenza (3 stati per la lettura di un valore diverso da 0, 3 stati per la lettura di un valore nullo).

```
1  --- ./project_tb.vhd
2  +++ ./project_tb_3.vhd
3  @@ -5,10 +5,10 @@
4   use ieee.numeric_std.all;
5   use std.textio.all;
6
7  -entity project_tb is
8  -end project_tb;
9  +entity project_tb_3 is
10 +end project_tb_3;
11
12 -architecture project_tb_arch of project_tb is
13 +architecture project_tb_arch_3 of project_tb_3 is
14       constant CLOCK_PERIOD : time := 20 ns;
15       signal tb_clk : std_logic := '0';
16       signal tb_rst, tb_start, tb_done : std_logic;
17 @@ -27,7 +27,7 @@
18       type scenario_type is array (0 to SCENARIO_LENGTH
             *2-1) of integer;
19
20       signal scenario_input : scenario_type := (128, 0,
             64, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
             100,  0, 1,  0, 0,  0, 5,  0, 23,  0, 200,  0,
             0,  0 );
21 -     signal scenario_full  : scenario_type := (128, 31,
     64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100,
     31, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30 );
22 +     signal scenario_full  : scenario_type := (128, 31,
     64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100,
     0, 1,  0, 0,  0, 5,  0, 23,  0, 200,  0,  0,  0 );
23
24       signal memory_control : std_logic := '0';
25
26 @@ -144,13 +144,16 @@
27
28           tb_start <= '1';
```

```
29
30 -         while tb_done /= '1' loop
31 -             wait until rising_edge(tb_clk);
32 -         end loop;
33 -
34 -         wait for 5 ns;
35 +         wait for 540 ns;
36
37          tb_start <= '0';
38 +
39 +         tb_rst <= '1';
40 +
41 +         -- Wait some time for the component to reset...
42 +         wait for 50 ns;
43 +
44 +         tb_rst <= '0';
45
46          wait;
47
48 @@ -170,20 +173,17 @@
49
50          wait until rising_edge(tb_start);
51
52 -         while tb_done /= '1' loop
53 -             wait until rising_edge(tb_clk);
54 -         end loop;
55 +         wait until tb_rst = '1';
56 +         wait for 25 ns;
57 +         assert tb_done = '0' report "TEST FALLITO o_done
       !=0 during reset" severity failure;
58 +         wait until tb_rst = '0';
59
60 -         assert tb_o_mem_en = '0' or tb_o_mem_we = '0'
      report "TEST FALLITO o_mem_en !=0 memory should not be
       written after done." severity failure;
61 +         assert tb_o_mem_en = '0' or tb_o_mem_we = '0'
      report "TEST FALLITO o_mem_en !=0 memory should not be
       written after reset." severity failure;
62
63          for i in 0 to SCENARIO_LENGTH*2-1 loop
64              assert RAM(SCENARIO_ADDRESS+i) =
                    std_logic_vector(to_unsigned(
                    scenario_full(i),8)) report "TEST FALLITO
                     @ OFFSET=" & integer'image(i) & "
                    expected= " & integer'image(scenario_full
                    (i)) & " actual=" & integer'image(
                    to_integer(unsigned(RAM(i)))) severity
                    failure;
65          end loop;
66
```

```
67 -         wait until falling_edge(tb_start);
68 -         assert tb_done = '1' report "TEST FALLITO o_done
       !=0 after reset before start" severity failure;
69 -         wait until falling_edge(tb_done);
70 -
71           assert false report "Simulation Ended! TEST
                PASSATO (EXAMPLE)" severity failure;
72       end process;
```

3.2.4 Test bench 4 - Nessuna modifica in memoria fuori dai valori specificati

Questo caso di test estende il caso precedentemente definito dal test bench 1 (3.2.1), andando a verificare che al termine dell'elaborazione non solo le celle di memoria interessate fossero state aggiornate, ma anche a verificare che nessun'altra cella di memoria sia stata sovrascritta con un valore diverso da 0 (nel test le celle della memoria sono inizializzate a 0).

```
1  --- ./project_tb.vhd
2  +++ ./project_tb_4.vhd
3  @@ -1,14 +1,14 @@
4  -- TB EXAMPLE PFRL 2023-2024
5  +-- TB 4 PFRL 2023-2024
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9   use ieee.numeric_std.all;
10  use std.textio.all;
11
12 -entity project_tb is
13 -end project_tb;
14 +entity project_tb_4 is
15 +end project_tb_4;
16
17 -architecture project_tb_arch of project_tb is
18 +architecture project_tb_arch_4 of project_tb_4 is
19      constant CLOCK_PERIOD : time := 20 ns;
20      signal tb_clk : std_logic := '0';
21      signal tb_rst, tb_start, tb_done : std_logic;
22 @@ -179,6 +179,13 @@
23          for i in 0 to SCENARIO_LENGTH*2-1 loop
24              assert RAM(SCENARIO_ADDRESS+i) =
                    std_logic_vector(to_unsigned(
                    scenario_full(i),8)) report "TEST FALLITO
                     @ OFFSET=" & integer'image(i) & "
                    expected= " & integer'image(scenario_full
                    (i)) & " actual=" & integer'image(
                    to_integer(unsigned(RAM(i)))) severity
                    failure;
25          end loop;
```

```
26 +
27 +         for i in 0 to SCENARIO_ADDRESS-1 loop
28 +             assert RAM(i) = "00000000" report "TEST
       FALLITO @ ADDRESS=" & integer'image(i) & "IS NOT 0"
       severity failure;
29 +         end loop;
30 +         for i in SCENARIO_ADDRESS+SCENARIO_LENGTH*2 to
       65535 loop
31 +             assert RAM(i) = "00000000" report "TEST
       FALLITO @ ADDRESS=" & integer'image(i) & "IS NOT 0"
       severity failure;
32 +         end loop;
33
34           wait until falling_edge(tb_start);
35           assert tb_done = '1' report "TEST FALLITO o_done
                 !=0 after reset before start" severity
                failure;
```

3.2.5 Test bench 5 - Primo valore letto dalla sequenza 0

Questo caso di test estende il caso precedentemente definito dal test bench 1 (3.2.1), andando a considerare il comportamento del modulo quando il primo valore letto dalla memoria è 0. In questo caso il modulo lascia tale valore invariato, e scrive come valore di credibilità nella cella di memoria successiva 0. Per verificare l'effettiva scrittura del valore di credibilità, impostiamo tale cella al valore 1. Vengono quindi modificati i primi due numeri di scenario_input in 0, 1 e di scenario_full in 0, 0. Il resto dei valori della sequenza resta invece invariato.

```
1  --- ./project_tb.vhd
2  +++ ./project_tb_5.vhd
3  @@ -1,14 +1,14 @@
4  -- TB EXAMPLE PFRL 2023-2024
5  +-- TB 5 PFRL 2023-2024
6
7   library ieee;
8   use ieee.std_logic_1164.all;
9   use ieee.numeric_std.all;
10  use std.textio.all;
11
12 -entity project_tb is
13 -end project_tb;
14 +entity project_tb_5 is
15 +end project_tb_5;
16
17 -architecture project_tb_arch of project_tb is
18 +architecture project_tb_arch_5 of project_tb_5 is
19      constant CLOCK_PERIOD : time := 20 ns;
20      signal tb_clk : std_logic := '0';
21      signal tb_rst, tb_start, tb_done : std_logic;
```

```
@@ -26,8 +26,8 @@
     constant SCENARIO_LENGTH : integer := 14;
     type scenario_type is array (0 to SCENARIO_LENGTH
         *2-1) of integer;

-    signal scenario_input : scenario_type := (128, 0, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100, 0, 1, 0, 0, 0, 5, 0, 23, 0, 200, 0, 0, 0 );
-    signal scenario_full  : scenario_type := (128, 31, 64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100, 31, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30 );
+    signal scenario_input : scenario_type := (0, 1, 64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100, 0, 1, 0, 0, 0, 5, 0, 23, 0, 200, 0, 0, 0 );
+    signal scenario_full  : scenario_type := (0, 0, 64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100, 31, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30 );

     signal memory_control : std_logic := '0';
```

3.2.6 Test bench 6 - Decremento valore di credibilità fino a 0

Questo caso di test estende il caso precedentemente definito dal test bench 1 (3.2.1),considerando una differente sequenza, costituita da un valore iniziale diverso da 0 che porta la credibilità a 31, seguito da soli 0 come valori di lettura, che portano la credibilità fino a 0 e permettono di verificare che questa non decrementi ulteriormente effettuando underflow, ma una volta raggiunto il valore minimo rimanga invece fissa a 0. Per realizzarlo, è stato mantenuto il primo valore delle sequenze scenario_input e scenario_full, con rispettivo valore di credibilità, estendendo le sequenze fino ad un totale di 35 coppie valore-credibilità, con tutte le altre coppie poste a 0. La credibilità nella sequenza finale deve decrementare di 1 fino a 0, e deve essere scritto il primo valore letto (128) per 34 volte.

```
--- ./project_tb.vhd
+++ ./project_tb_6.vhd
@@ -1,14 +1,14 @@
-- TB EXAMPLE PFRL 2023-2024
+- TB 6 PFRL 2023-2024

 library ieee;
 use ieee.std_logic_1164.all;
 use ieee.numeric_std.all;
 use std.textio.all;

-entity project_tb is
-end project_tb;
+entity project_tb_6 is
+end project_tb_6;

```

```
17 -architecture project_tb_arch of project_tb is
18 +architecture project_tb_arch_6 of project_tb_6 is
19       constant CLOCK_PERIOD : time := 20 ns;
20       signal tb_clk : std_logic := '0';
21       signal tb_rst, tb_start, tb_done : std_logic;
22 @@ -23,11 +23,11 @@
23       type ram_type is array (65535 downto 0) of
             std_logic_vector(7 downto 0);
24       signal RAM : ram_type := (OTHERS => "00000000");
25
26 -     constant SCENARIO_LENGTH : integer := 14;
27 +     constant SCENARIO_LENGTH : integer := 35;
28       type scenario_type is array (0 to SCENARIO_LENGTH
             *2-1) of integer;
29
30 -     signal scenario_input : scenario_type := (128, 0,
       64, 0,   0,  0,  0,  0,  0,  0,  0,  0,  0, 100,
       0, 1,  0, 0,  0, 5,  0, 23,  0, 200,  0,   0,  0 );
31 -     signal scenario_full  : scenario_type := (128, 31,
       64, 31, 64, 30, 64, 29, 64, 28, 64, 27, 64, 26, 100,
       31, 1, 31, 1, 30, 5, 31, 23, 31, 200, 31, 200, 30 );
32 +     signal scenario_input : scenario_type := (128,  0,
         0,  0,   0,  0,   0,  0,   0,  0,   0,  0,   0,  0,
         0,  0,   0,  0,   0,  0,   0,  0,   0,  0,   0,  0,
         0,  0,   0,  0,   0,  0,   0,  0,   0,  0,   0,  0,
         0,  0,   0,  0,   0,  0,   0, 0,   0, 0,   0, 0,
       0, 0,   0, 0,   0, 0,   0, 0,   0, 0,   0, 0,   0, 0,
         0, 0,   0, 0,   0, 0 );
33 +     signal scenario_full  : scenario_type := (128, 31,
       128, 30, 128, 29, 128, 28, 128, 27, 128, 26, 128, 25,
       128, 24, 128, 23, 128, 22, 128, 21, 128, 20, 128, 19,
       128, 18, 128, 17, 128, 16, 128, 15, 128, 14, 128, 13,
       128, 12, 128, 11, 128, 10, 128, 9, 128, 8, 128, 7,
       128, 6, 128, 5, 128, 4, 128, 3, 128, 2, 128, 1, 128,
       0, 128, 0, 128, 0, 128, 0 );
34
35       signal memory_control : std_logic := '0';
```

4 Conclusioni

Scomponendo il modulo hardware con un approccio strutturale in differenti moduli, è stato possibile realizzare moduli di complessità inferiore che, interagendo tra loro, vanno a soddisfare le specifiche fornite per il progetto. Le specifiche sono state valutate inoltre con opportuni test bench in pre-sintesi e post-sintesi funzionale, potendo quindi verificare il funzionamento del componente in una simulazione dell'ambiente in cui potrebbe essere impiegato.