UNIVERSITÀ DEGLI STUDI DI UDINE

DIPARTIMENTO POLITECNICO DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria Elettronica

Laboratorio di Calcolatori Elettronici: Dispositivo *sumavg*

Giulio Marziali
Mat. 135422

ANNO ACCADEMICO 2023/2024

...

Indice

Capitolo 1

Introduzione

1.1 Obiettivo del Progetto

L'obiettivo del presente documento è quello di affrontare la progettazione di un componente che sia in grado di interfacciarsi a una memoria esterna e calcolare la media della somma di due vettori di dati.
Dati i puntatori *ptr1* e *ptr2* agli indirizzi di memoria a partire dai quali risiedono due vettori composti da *len* elementi, il componente dovrebbe calcolare:

$$res = avg(ptr1[i] + ptr2[i]) \tag{1.1}$$

I dati nei vettori sono espressi in formato **Q16.16**, fixed point con 16 bit per la parte intera e 16 bit per la parte frazionaria.

1.2 Specifiche di Progetto

Il componente, che dovrà essere in grado di interfacciarsi con una memoria esterna, sarà, a tal proposito, parte integrante di un sistema, il cui schema a blocchi è fornito a priori.

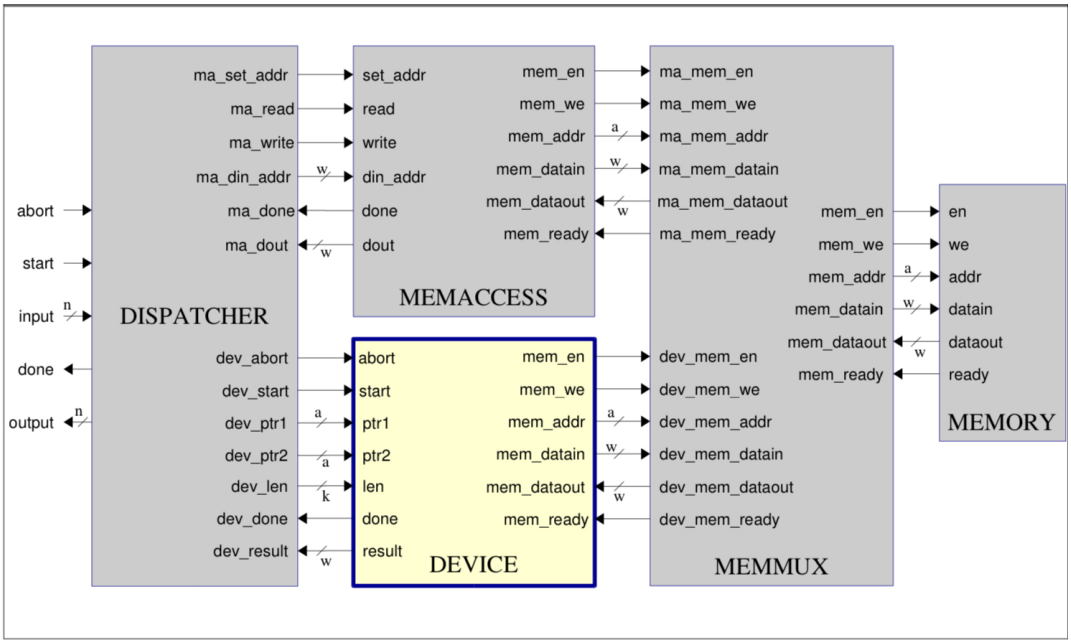

Figura 1.1: Architettura del sistema.

La Figura 1.1 presenta l'intero sistema che dialogherà con la porta seriale, all'interno del quale risultano individuabili i componenti:

- *dispatcher*: si tratta di un componente dedicato a interpretare l'input e selezionare, di conseguenza, l'operazione da svolgere (accesso in memoria oppure computazione di *sumavg*);
- *memaccess*: questo componente gestisce le operazioni di lettura e scrittura in memoria;
- *memmux*: è un'interfaccia dalle proprietà simili al multiplexer, tale che riesca a gestire gli accessi in memoria nei due casi specifici, qualora provengano dal dispositivo *sumavg*, oppure da *memaccess*, assicurandosi che i due accessi non avvengano contemporaneamente;

- *sumavg*: il componente che si intende progettare.

Ciascun componente è stato suddiviso in una control unit, sviluppata come macchina a stati finiti tramite un opportuno ASM chart, e in un datapath, insieme delle unità funzionali ottenuto direttamente dall'ASM chart. Per una descrizione più dettagliata del funzionamento del componente *sumavg* si rimanda alla lettura della sezione relativa al diagramma ASM.

Capitolo 2

Progettazione

2.1 Diagrammi ASM

2.1.1 Dispatcher, memaccess e memmux

Si ritiene opportuno richiamare, inizialmente, i diagrammi ASM relativi ai componenti *dispatcher*, *memaccess* e *memmux*, già precedentemente disponibili, in quanto forniti a corredo delle specifiche di progetto. Essi sono risultati necessari per la stesura del codice VHDL implementativo dei tre componenti in questione.

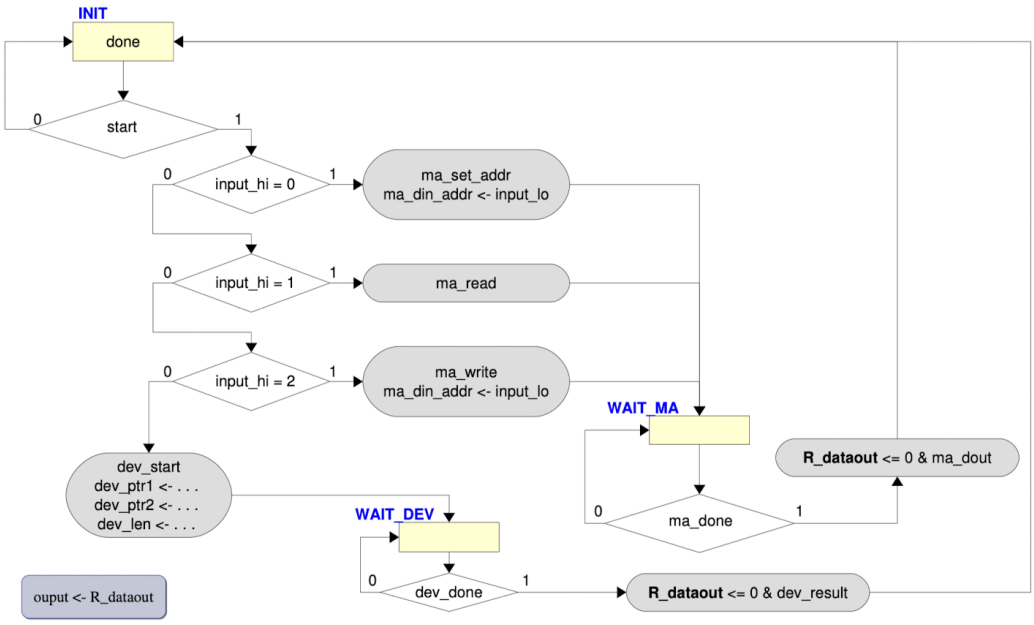

Figura 2.1: ASM chart del componente *dispatcher*.

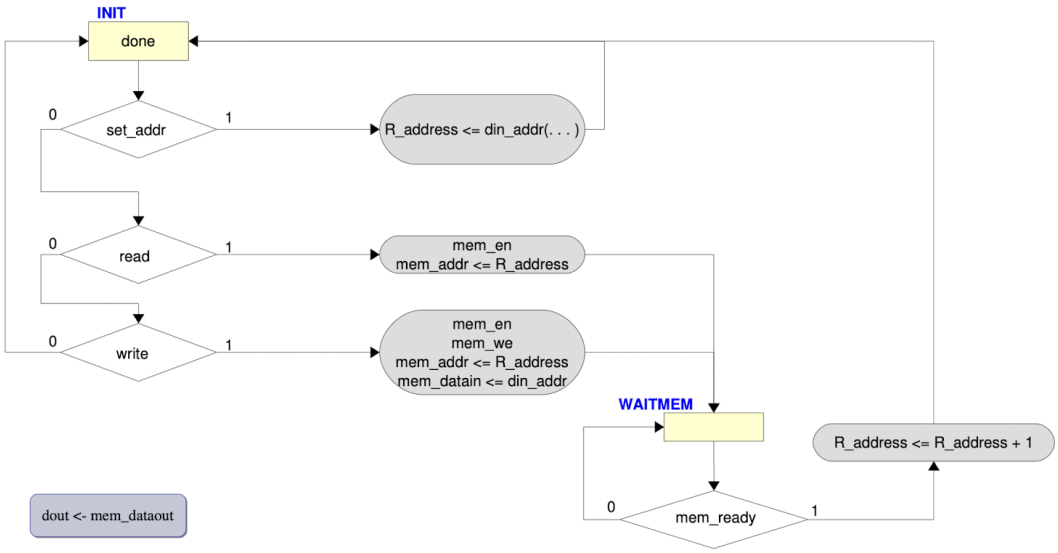

Figura 2.2: ASM chart del componente *memaccess*.

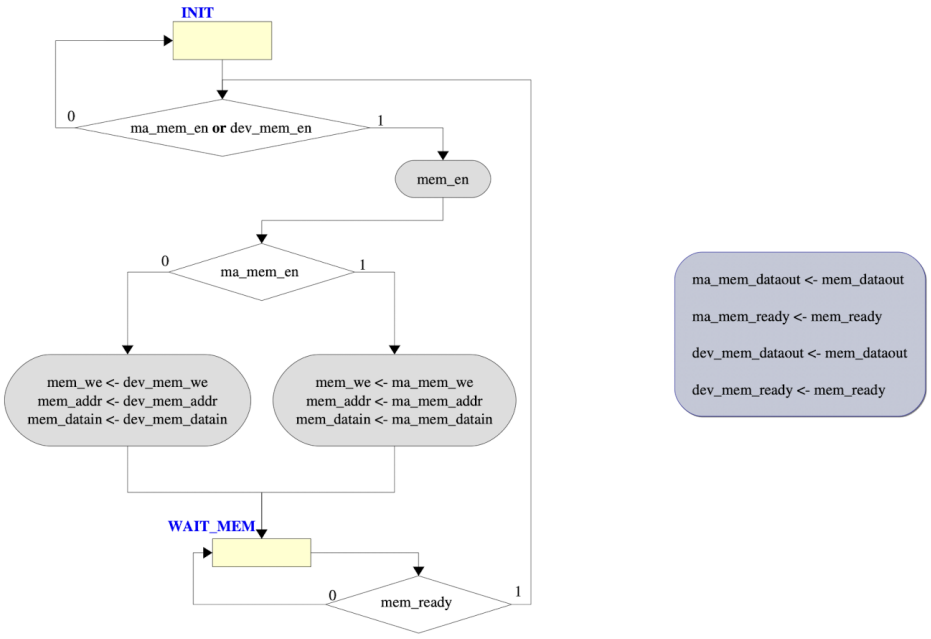

Figura 2.3: ASM chart del componente *memmux*.

2.1.2 Sumavg

In Figura 2.4 è illustrato il diagramma ASM, progettato per lo sviluppo del dispositivo *sumavg*.
L'idea alla base dello schema raffigurato nasce essenzialmente dalla necessità di individuare tre macroblocchi che dovrebbero rispettivamente occuparsi di:

- predisporre i dati per la computazione;
- accumulare i risultati intermedi delle somme in un registro;
- ottenere il risultato finale tramite una divisione.

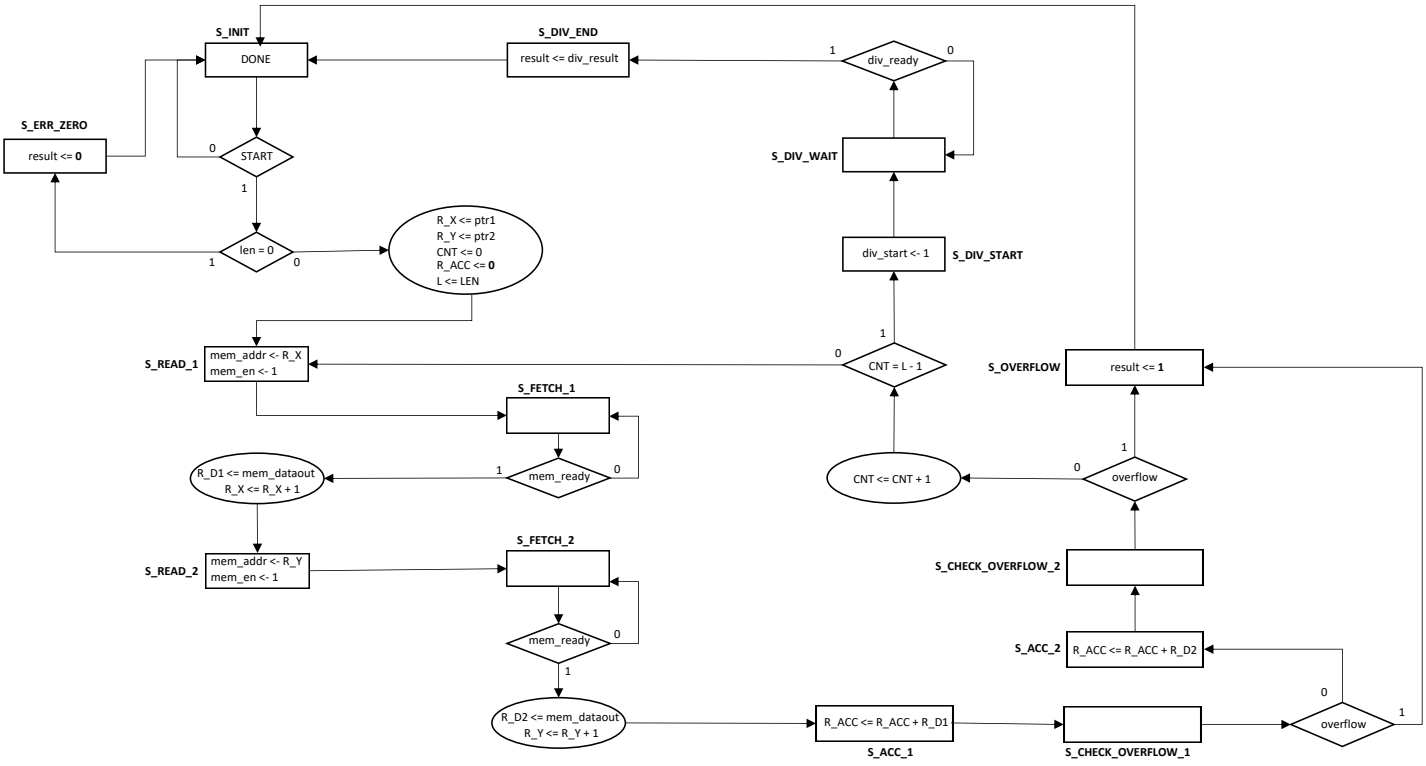

Figura 2.4: ASM chart del dispositivo *sumavg*.

Nella fase di inizializzazione, il valore logico alto del segnale start determina l'inizio dell'elaborazione da parte del dispositivo. Il primo controllo necessario è riguardante la lunghezza dei vettori di dati da caricare dalla memoria: qualora la lunghezza estrapolata dalla parola di input risultasse nulla, l'operazione di divisione finale non sarebbe definita, per cui viene previsto uno stato apposito e una soluzione descritta nel Paragrafo 2.1.3. Qualora venga rilevata una lunghezza maggiore di zero per i vettori, l'algoritmo di controllo procede e vengono caricati i valori *ptr1*, *ptr2* e *len* nei registri X, Y e L, rispettivamente. Il contenuto dei registri *CNT* e *R_ACC* viene azzerato, in preparazione all'operazione di somma e media.

Nello stato *S_READ_1*, viene fornito alla memoria il primo indirizzo *ptr1*, contenuto nel registro X, ed il segnale *mem_en* viene attivato, abilitando l'operazione di lettura dalla memoria.

Per quanto riguarda lo stato *S_FETCH_1*, il diagramma implementa la logica per prelevare gli elementi del vettore dalla memoria e inserirli, ad ogni iterazione, in un registro apposito. Il segnale *mem_ready* viene controllato prima di procedere nelle operazioni di lettura, garantendo la sincronizzazione con il modulo di memoria. L'incremento del puntatore X, una volta eseguita la lettura, è funzionale all'aggiornamento dell'indirizzo in modo da favorire l'estrazione del dato successivo. Il sistema prevede un comportamento analogo per il caricamento del dato dal secondo indirizzo di memoria.

Nel loop di accumulo *S_ACC_1* ed *S_ACC_2* sono gli stati che regolano le somme delle componenti vettoriali, a valle di ciascuno dei quali viene effettuato il necessario controllo di overflow, secondo quanto descritto nel Paragrafo 2.1.4. I dati contenuti nei registri preposti vengono sommati ed accumulati di volta in volta nel registro *R_ACC*. L'incremento del registro *CNT* è necessario per tenere traccia del numero di elementi elaborati. Il controllo della condizione *CNT = L - 1* dopo ogni passo di accumulo, consente la verifica di quanti

elementi del vettore siano stati elaborati. Se *CNT* è minore di *L-1*, si ritorna allo stato *S_READ_1*, preparandosi a leggere la coppia successiva di elementi, altrimenti, la condizione che ferma il ciclo è il raggiungimento del valore *L-1* nel conteggio.

L'ultimo macroblocco del diagramma ASM si apre con lo stato *S_DIV_START*, nel quale viene avviata la divisione tramite l'opportuno segnale di controllo: al termine della fase di accumulo, è necessaria una logica per lasciare al divisore il tempo opportuno per ottenere il risultato. Lo stato *S_DIV_WAIT* è stato inserito per gestire l'attesa nel corso dell'operazione, mentre lo stato *S_DIV_END* termina la divisione e fornisce il risultato attraverso l'uscita *result* del dispositivo.

Quando il divisore ha finito, il registro *R_ACC* (contenente la somma degli elementi dei vettori) è stato diviso per il valore *LEN*, completando così il calcolo della media. Una volta eseguita la divisione, non resta che ritornare allo stato iniziale e segnalare la fine della computazione, attivando il segnale *done*.

Riguardo alla gestione dei due casi singolari di overflow e divisione per zero, si è resa necessaria una modifica del flusso di controllo mediante l'introduzione di opportuni stati e segnali dedicati.

2.1.3 Gestione della divisione per zero

Per quanto concerne il caso in cui venga rilevata dall'input una lunghezza dei vettori nulla, nello stato *S_INIT*, il segnale di stato *len_zero* assume valore alto e notifica alla FSM della control-unit di entrare nello stato *S_ERR_ZERO*, bloccando di fatto il caricamento di dati e l'elaborazione del dispositivo, ma, a maggior ragione, impedendo l'inizio della divisione la quale risulterebbe indefinita. Contestualmente a ciò, tramite l'invio di un segnale di controllo al datapath, si è scelto di forzare il risultato dell'elaborazione del

dispositivo ad assumere il valore speciale esadecimale "00000000".

2.1.4 Gestione dell'overflow

Dopo ogni somma all'interno del loop di accumulo, qualora si riscontri una condizione tale da provocare il raggiungimento di un numero non rappresentabile in formato Q16.16, il flag *R_acc_carry*, connesso direttamente al segnale di stato *overflow*, viene posto a valore logico alto. Il massimo numero rappresentabile, espresso in notazione decimale, risulta 2147483647, mentre il minimo numero rappresentabile è -2147483648. Per evidenziare nel flusso di controllo il caso di overflow correlato, si è previsto uno stato apposito *S_OVERFLOW*.
L'algoritmo tratta la problematica attraverso l'invio di un opportuno segnale al datapath, la cui ricezione impone il risultato dell'elaborazione al valore speciale rappresentato da 32 bit dal valore logico '1' (in esadecimale, la lettura dell'output del sistema sarà "00FFFFFF").

2.2 Codice in linguaggio VHDL

In questa sezione del testo, si riassumerà le parti salienti del codice sviluppato, in linguaggio VHDL, con il proposito di chiarire le scelte effettuate in fase di progetto.
Ciascun dispositivo è stato ottenuto sfruttando la canonica suddivisione in control unit e datapath, fatta eccezione per la memoria, implementata tramite il file *memory.vhdl*, reso disponibile dal docente.
La versione integrale del codice viene consegnata in allegato alla relazione di laboratorio.

2.2.1 Dispatcher, memaccess e memmux

Per la scrittura del codice relativo ai dispositivi *dispatcher*, *memaccess* e *memmux*, si è fatto riferimento ai diagrammi ASM, raccolti nel capitolo precedente.

2.2.2 Sumavg

In Figura 2.4, il diagramma ASM del dispositivo *sumavg* evidenzia quali siano gli stati e le loro transizioni, da specificarsi nell'opportuno processo, all'interno della control unit. Per il dispositivo si è prevista la seguente interfaccia:

```
entity sumavg is
generic (
   W_BITS             : integer := 32;
   A_BITS             : integer := 12;
   K_BITS             : integer := 8
);
port (
    CLK               : in std_logic;
    rst_n             : in std_logic;
       -- inputs
    start             : in std_logic;
```

```
        abort             : in std_logic;
        ptr1              : in std_logic_vector(A_BITS-1 downto 0);
        ptr2              : in std_logic_vector(A_BITS-1 downto 0);
        len               : in std_logic_vector(K_BITS-1 downto 0);
        mem_dataout       : in std_logic_vector(W_BITS-1 downto 0);
        mem_ready         : in std_logic;
           -- outputs
        done              : out std_logic;
        result            : out std_logic_vector(W_BITS-1 downto 0);
        mem_en            : out std_logic;
        mem_we            : out std_logic;
        mem_addr          : out std_logic_vector(A_BITS-1 downto 0);
        mem_datain        : out std_logic_vector(W_BITS-1 downto 0)
);
end sumavg;
```

I generici *N_BITS*, *W_BITS*, *A_BITS* e *K_BITS* sono stati inseriti per dotare il codice della necessaria modularità.

2.2.3 Il sistema

Per il collegamento dei vari componenti del sistema, si è optato per una descrizione strutturale, ottenuta facendo riferimento alla Figura 1.1. Compatibilmente con ciò che ci si aspetti, l'intero sistema presenta un'interfaccia di questo tipo:

```
entity system is
   generic (
      N_BITS              : integer := 40;
      W_BITS              : integer := 32;
      A_BITS              : integer := 12;
      K_BITS              : integer := 8
   );
   port (
```

```
      CLK                  : in std_logic;
      rst_n                : in std_logic;
      ---
      sys_abort            : in std_logic;
      sys_start            : in std_logic;
      sys_input            : in std_logic_vector(N_BITS-1 downto 0);
      sys_done             : out std_logic;
      sys_output           : out std_logic_vector(N_BITS-1 downto 0)
   );
end system;
```

2.2.4 Tester

Per poter fornire gli ingressi e testare il sistema, si è utilizzato un tester con la seguente interfaccia:

```
   entity tester is
   generic (
      W_BITS               : natural := 32;
      A_BITS               : natural := 12;
      K_BITS               : natural := 8;
      N_BITS               : natural := 40
   );
   port (
      CLK                  : in std_logic;
      rst_n                : in  std_logic;
      abort                : out std_logic;
      start                : out std_logic;
      input_data           : out std_logic_vector(N_BITS - 1 downto 0);
      done                 : in  std_logic;
      output_data          : in  std_logic_vector(N_BITS - 1 downto 0);
      finished             : out std_logic
   );
```

```
end entity;
```

2.2.5 Testbench

Nel testbench sono stati istanziati l'intero sistema ed il tester, grazie alla definizione degli opportuni package.
Il periodo di clock di 10 ns è stato ricavato dalle specifiche della FPGA, scelta per la successiva sintesi del dispositivo. Il tempo di reset previsto risulta pari a 50ns.

2.3 Simulazione RTL del Sistema

Per la simulazione a livello RT del sistema, si è deciso di optare per il software ModelSim SE-64 10.7. Il codice sviluppato è stato compilato con successo, dopodiché, è stato possibile procedere al banco di simulazione, così da verificare il corretto comportamento del sistema.
I file ottenuti nel corso delle simulazioni vengono consegnati in allegato alla relazione.
Prima di procedere nella trattazione, si desidera richiamare il fatto che sia stato utilizzato un tester per impartire gli ingressi al sistema, e valutarne il funzionamento sotto una molteplicità di stimoli e condizioni.

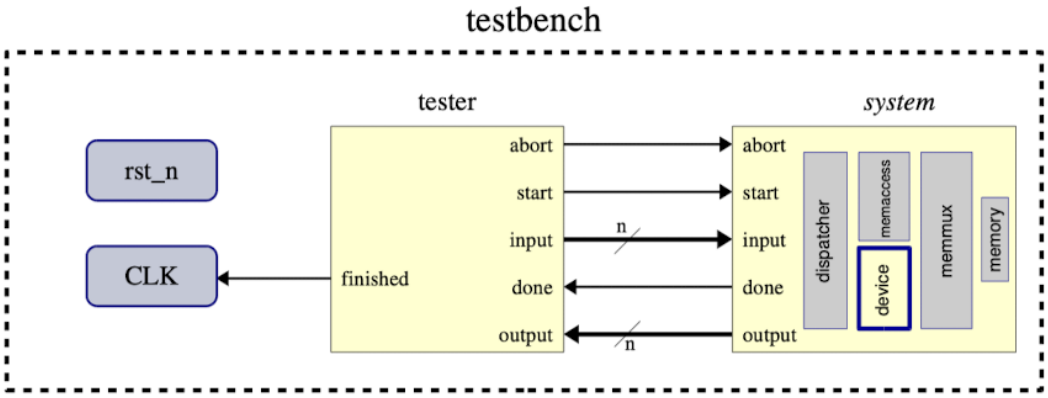

Figura 2.5: Architettura di test per la simulazione del sistema.

2.3.1 Dispatcher, memaccess e memmux

Nell'ottica della simulazione dei dispositivi *dispatcher*, *memaccess* e *memmux*, dobbiamo accertarci che il comportamento desiderato sia rispettato, ovvero che sia possibile accedere rispettivamente alla memoria oppure al dispositivo, in funzione del valore rilevato presso *input_hi*, gli 8 bit alti della parola di ingresso. A tal proposito, si vuole dimostrare la correttezza del comportamento in merito ai tre casi, rispettivamente, di impostazione dell'indirizzo, accesso in scrittura ed accesso in lettura:

- In presenza del comando "SET ADDRESS", *input_hi* ha valore 0 ed il dispositivo *dispatcher* attiva l'opportuno percorso verso la memoria per

impostare l'indirizzo contenuto nel range dei 12 bit iniziali della parola di ingresso.

Nella Figura 2.6 si riporta l'andamento delle forme d'onda simuate, in riferimento al funzionamento indicato. Come si evince dalla figura, nell'esempio si è scelto come indirizzo il valore esadecimale "00A", ovvero l'indirizzo binario "000000001010";

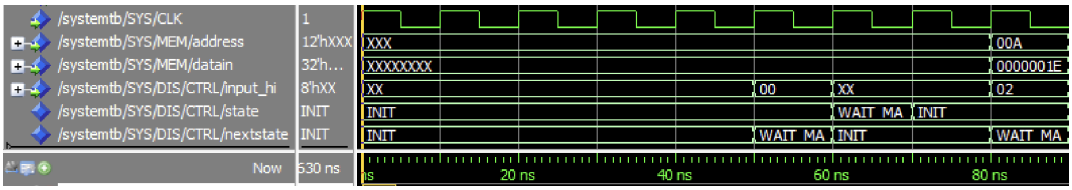

Figura 2.6: Simulazione del sistema nel caso del comando SET ADDRESS.

- In presenza del comando "WRITE", *input_hi* ha valore 2 ed il dispositivo *dispatcher* deve riuscire a scrivere in memoria il contenuto dei 32 bit bassi dell'input, presso l'indirizzo impostato. Per verificare che l'operazione sia andata a buon fine, sarà opportuno eseguire successivamente una lettura presso l'indirizzo stesso. Nella Figura 2.7 le forme d'onda indicano un esempio di scrittura, nel quale viene memorizzato il dato esadecimale "0000001E" presso l'indirizzo impostato "00A";

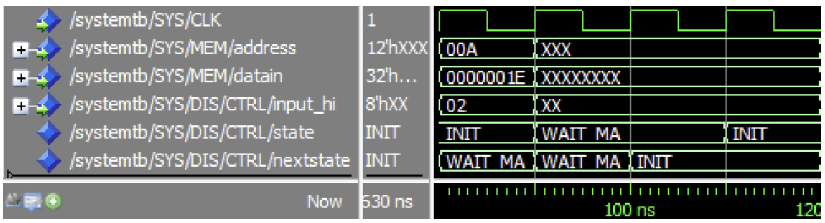

Figura 2.7: Simulazione del sistema nel caso del comando WRITE.

- In presenza del comando "READ", *input_hi* ha valore 1 ed il dispositivo *dispatcher* dovrebbe leggere dalla memoria il dato in corrispondenza dell'indirizzo impostato. Per verificare che l'operazione sia andata a buon fine, si è deciso di eseguire una scrittura all'indirizzo prima di eseguirne la lettura. Nella Figura 2.8 è indicato un esempio di lettura, nel quale,

dopo aver impostato l'indirizzo "00A" con l'operazione apposita, viene letto il dato memorizzato e il canale *dataout* risulta riportare il dato scritto nel corso della precedente operazione.

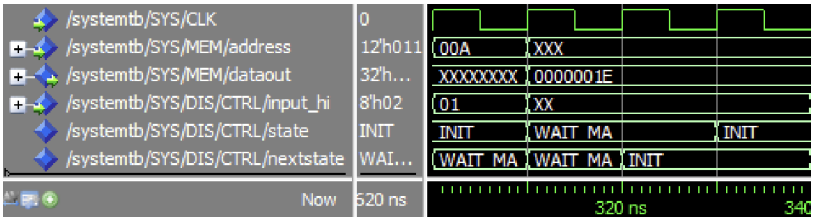

Figura 2.8: Simulazione del sistema nel caso del comando READ.

Il dispositivo *memaccess* viene attivato, qualora si desideri accedere alla memoria. *Memmux*, invece, si occupa di dirigere gli accessi alla memoria da parte di *memaccess* e del dispositivo, ed accertare che essi siano mutuamente esculsivi. La simulazione nei tre casi dei rispettivi comandi ha fornito i risultati sperati, provando che:

- L'impostazione dell'indirizzo fornisce correttamente i dati alla memoria, attraverso l'indirizzo *mem_addr*;
- L'operazione di lettura comunica l'indirizzo di interesse ed attiva il segnale di controllo *mem_en*, che permette di attivare gli accessi in lettura;
- L'operazione di scrittura attiva anche il segnale di controllo *mem_we* per gli accessi in scrittura e fornisce i dati alla memoria tramite il vettore di 32 bit *mem_datain*.

2.3.2 Sumavg

Circa la simulazione del dispositivo *sumavg*, viene proposto un esempio concreto, e quanto più completo possibile, nel quale si vuole accedere alla computazione del dispositivo.
Gli ingressi impartiti al tester risultano i seguenti:

```
constant CMD_SA          : integer := 0;
constant CMD_RD          : integer := 1;
constant CMD_WR          : integer := 2;
constant CMD_GO          : integer := 100;
constant NODATA          : integer := -1;

type array_of_integers is array (natural range <>) of integer;
constant COMMANDS        : array_of_integers := (CMD_SA, CMD_WR, CMD_WR, CMD_WR, CMD_SA, CMD_WR, CMD_WR, CMD_WR, CMD_GO, CMD_GO, CMD_GO);
constant DATA            : array_of_integers := (10, 15204352, 63111168, 117178368, 16, 50331648, 82116608, 2147483247, 17, 18, 10);
constant ADDR1           : array_of_integers := (NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, 10, 16, 10);
constant ADDR2           : array_of_integers := (NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, 16, 16, 16);
constant LEN             : array_of_integers := (NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, NODATA, 3, 2, 0);
```

Figura 2.9: La figura illustra i valori degli ingressi impartiti al tester.

In Figura 2.10 sono raccolte le forme d'onda e le rispettive risposte del sistema, relativamente agli ingressi ricevuti.

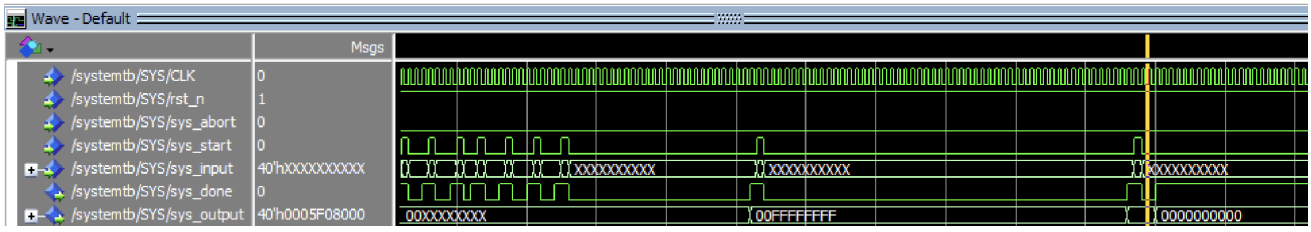

Figura 2.10: Simulazione della computazione di *sumavg*, la figura illustra output del sistema, in riferimento al dispositivo.

La prima richiesta, rivolta al sistema per una computazione da parte del dispositivo, viene eseguita tramite la parola di input "640301000A". Pertanto il sistema riceve un comando per il quale *input_hi* assume un valore differente da 0, 1 e 2, cosicché l'accesso non sia direttamente alla memoria, ma al dispositivo.
La lunghezza richiesta dei vettori per il calcolo di *sumavg* è 3, mentre gli indirizzi di *ptr1* e *ptr2* per eseguire il fetch dei dati sono, rispettivamente, 10 e 16: infatti, nei 12 bit meno significativi ritroviamo il valore esadecimale "00A",

mentre dal 13esimo bit al 24esimo ritroviamo il valore esadecimale "010".
La Figura 2.11 intende dimostrare che la memoria sia acceduta il numero di volte corretto, dipendentemente dal segnale *len*. Da un'analisi delle forme d'onda illustrate, emergono inoltre i conteggi delle iterazioni prima e successivamente alla commutazione del segnale done.

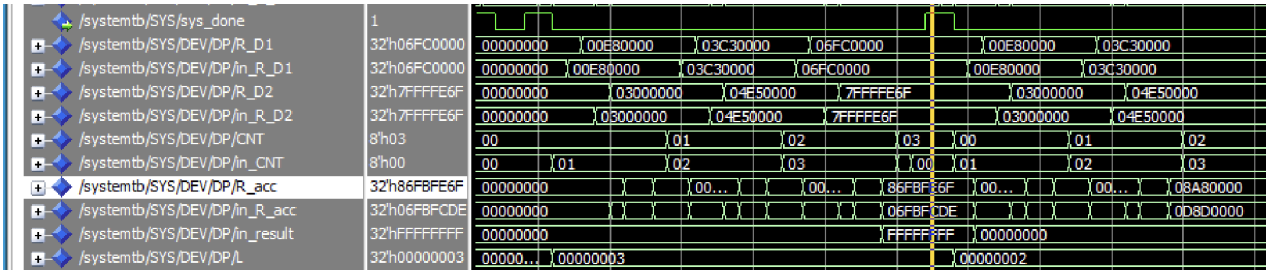

Figura 2.11: Simulazione della computazione di *sumavg*, la figura illustra le forme d'onda relative al caricamento dei dati dalla memoria e all'avanzamento del processo di accumulo.

In questo esempio, i dati caricati dagli indirizzi di memoria selezionati erano stati scritti precedentemente facendo uso del sistema stesso. I valori esadecimali, corrispondenti a quelli decimali presentati in Figura 2.9, risultano, rispettivamente, "00E80000", "03C30000" e "06FC0000" per il primo vettore, mentre vengono caricati i dati "03000000", "04E50000" e "7FFFFE6F" per il secondo vettore.
Se tutto procedesse senza impedimenti, al termine della fase di accumulo, la quantità risulante aggiornata nel registro *R_ACC* sarebbe divisa per la lunghezza 3. È evidente che la somma del terzo elemento del secondo vettore, all'ultimo passo del loop di accumulo, faccia ricadere nella problematica di overflow. La risposta del sistema nel caso evidenziato consiste nel fornire in output la sequenza di 32 bit dal valore binario '1' (valore esadecimale "00FFFFFF" su 40 bit).

Esaminiamo ora il secondo ingresso proposto al sistema, ovvero la parola di input "640201000A". È possibile notare che l'unica differenza con il caso pre-

cedente sia la lunghezza dei vettori, da 3 ridotta a 2. Come conseguenza di ciò, la fase di accumulo prevede soltanto due iterazioni e il terzo dato causante overflow non viene acceduto. In tal caso, il risulato è compatibile con le attese e riporta l'uscita esadecimale "0004540000".

Il terzo stimolo impartito al sistema vuole evidenziarne il comportamento, contestualmente alla ricezione della parola di input "640001000A", ossia in caso di lunghezza nulla. L'operazione di divisione non risulta dunque possibile e, pertanto, il dispositivo attiva un segnale di controllo, concepito per trattare questa circostanza assegnando all'output una sequenza nulla, come si può rilevare dalla Figura 2.12.

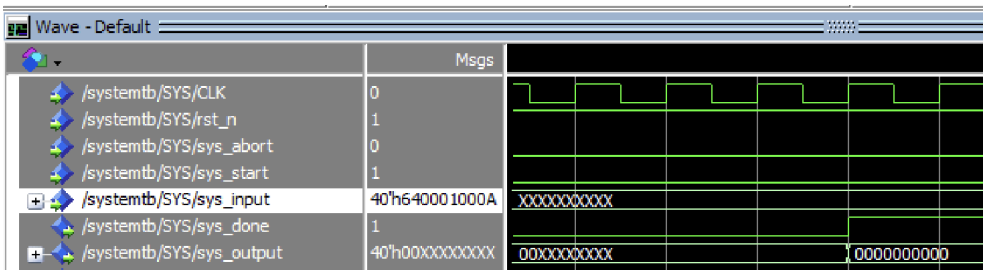

Figura 2.12: Simulazione della computazione di *sumavg*, la figura illustra la gestione di input facenti riferimento a vettori di lunghezza nulla.

Di seguito si raccoglie alcune transizioni di stato della control-unit di *sumavg* nel contesto degli esempi proposti:

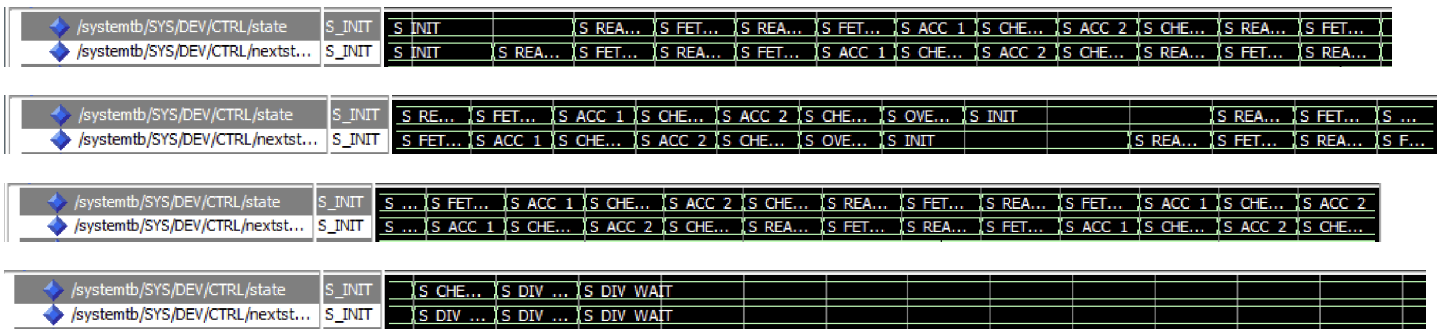

Figura 2.13: Le figure proposte illustrano alcuni transizioni di stato della control-unit del dispositivo *sumavg*.

Capitolo 3

Fase di Sintesi

3.1 Scelta della scheda

Dal momento che il grado di impellenza richiesto dall'applicazione non prevede particolare velocità di esecuzione, per il processo di sintesi la scelta è ricaduta su FPGA.

Il dispositivo designato si chiama Xilinx Spartan6 - XC6SLX45 FPGA, che rappresenta un ottimo compromesso tra densità logica, risorse per l'elaborazione di segnali ed equipaggiamento di features. La scheda è idonea ad una molteplicità di applicazioni embedded e industriali, basate su tecnologie di elaborazione low-power.

3.2 Presentazione dei risultati

La scelta del tool di sintesi, invece, è ricaduta su Xilinx ISE Design-Suite 12.1.

Per realizzare la sintesi del sistema è risultato necessario collegare strutturalmente il sistema stesso e un modulo di interfaccia, come da Figura 3.1.

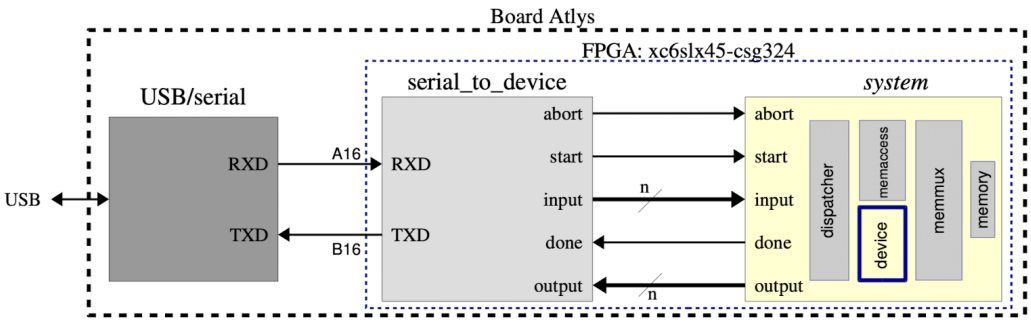

Figura 3.1: Collegamento strutturale sistema-interfaccia, per il processo di sintesi.

Nonostante l'insieme dei log e file, generati dal processo di sintesi, sia consegnato in allegato nella sua totalità, si desidera comunque presentare in Figura 3.2 alcuni dei risultati ottenuti:

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	683	54,576	1%
Number used as Flip Flops	683		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	754	27,288	2%
Number used as logic	623	27,288	2%
Number using O6 output only	397		
Number using O5 output only	29		
Number using O5 and O6	197		
Number used as ROM	0		
Number used as Memory	1	6,408	1%
Number used as Dual Port RAM	0		
Number used as Single Port RAM	0		
Number used as Shift Register	1		
Number using O6 output only	1		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used exclusively as route-thrus	130		
Number with same-slice register load	129		
Number with same-slice carry load	1		
Number with other load	0		
Number of occupied Slices	221	6,822	3%
Number of LUT Flip Flop pairs used	755		
Number with an unused Flip Flop	247	755	32%
Number with an unused LUT	1	755	1%
Number of fully used LUT-FF pairs	507	755	67%
Number of unique control sets	32		
Number of slice register sites lost to control set restrictions	76	54,576	1%
Number of bonded IOBs	4	218	1%
Number of RAMB16BWERs	8	116	6%
Number of RAMB8BWERs	0	232	0%
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of BUFG/BUFGMUXs	1	16	6%
Number used as BUFGs	1		

Figura 3.2: Riassunto dell'utilizzo di risorse, ottenuto nel processo di sintesi.

Dalla tabella si deduce che la quantità di flip-flop utilizzata per la sintesi ammonti a 683, una quantità estremamente bassa in confronto alla totalità di quelli messi a disposizione dalla scheda.