Politecnico di Milano

Progetto di Reti Logiche 2021/2022

Scaglione Prof. Salice

Edoardo Fullin (Codice Persona 10677606)

5 aprile 2022 (revisione 13 giugno 2022)

Indice

1 Introduzione

1.1 Descrizione del progetto

Lo scopo del progetto è il design, tramite linguaggio VHDL, di un modulo hardware sintetizzabile per FPGA. Tale modulo deve essere in grado di applicare ad una sequenza di parole in input un codice convoluzionale 1/2, che può essere implementato attraverso la macchina a stati finiti (FSM) in figura 1.

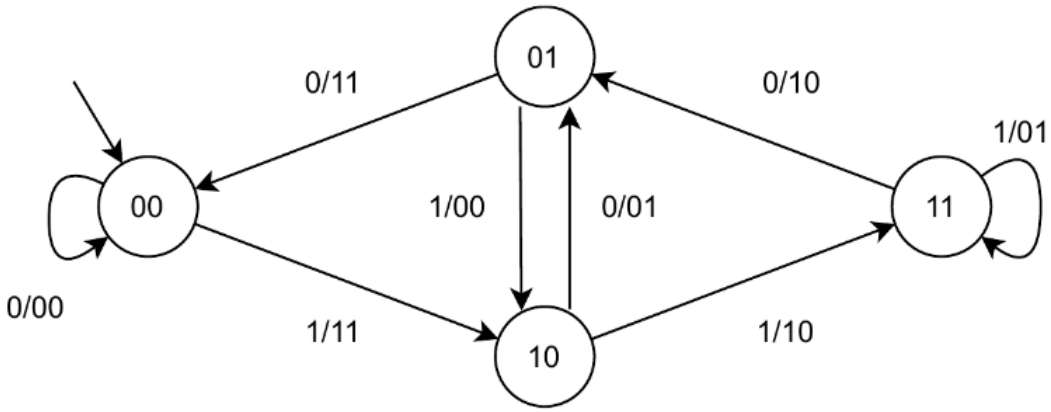

Figura 1: FSM convoluzionatore 1/2

Il modulo deve quindi restituire in output, scrivendo in memoria, le uscite della macchina a stati finiti, bit per bit.

1.2 Esempio di funzionamento

La FSM prende in input la parola `11001011`, che deve essere serializzata come 1 al tempo 0, 1 al tempo 1, 0 al tempo 2 e così via. La macchina quindi, partendo dallo stato di reset 00 andrà nello stato 10 alla lettura del primo 1, producendo in output 11; passerà poi nello stato 11 producendo 10 e successivamente nello stato 01 producendo 10 e infine nello stato 00 producendo 11. Il primo byte di output, relativo al primo nibble della parola di input, sarà quindi 11101011, che dovrà essere scritto in memoria.
Per il secondo nibble di input e per i successivi si applica lo stesso principio di funzionamento sopra descritto. Si nota quindi facilmente per ogni nibble in input viene prodotto un byte in output (la lunghezza dell'output è doppia di quella dell'input).

1.3 Specifiche del modulo da realizzare

Il modulo hardware da descrivere è sincronizzato su un clock esterno, non include la memoria su cui viene effettuato input/output e deve implementare la seguente interfaccia verso l'esterno:

```
entity project_reti_logiche is
    port (
        i_clk : in std_logic;
        i_rst : in std_logic;
        i_start : in std_logic;
        i_data : in std_logic_vector(7 downto 0);
        o_address : out std_logic_vector(15 downto 0);
        o_done : out std_logic;
        o_en : out std_logic;
        o_we : out std_logic;
        o_data : out std_logic_vector (7 downto 0)
    );
end project_reti_logiche;
```

In particolare:

- `i_clk` è il segnale di clock, generato dall'esterno
- `i_rst` è il segnale di reset
- `i_start` è il segnale di inizio sequenza
- `i_data` è il byte in arrivo dalla memoria RAM
- `o_address` è l'indizzo di memoria da cui si vuole leggere/scrivere
- `o_done` è il segnale di fine computazione
- `o_en` è il segnale di enable per la memoria RAM
- `o_we` è il segnale di enable per la scrittura in memoria RAM
- `o_data` è il byte da scrivere in memoria RAM

2 Architettura

L'architettura del componente è composta principalmente da due sottomoduli: una macchina a stati di controllo e da un datapath che gestisce il flusso dati. La FSM di controllo ha il compito di generare i segnali di controllo che attivano, disattivano o modificano il comportamento dei sottomoduli del datapath in base allo stato attuale dell'esecuzione. Il datapath gestisce l'intero flusso di dati, dalla lettura dalla memoria di input, alla serializzazione, alla bufferizzazione del risultato e alla sua scrittura in memoria.
Vengono illustrati più in dettaglio i singoli componenti.

2.1 Datapath

Il datapath è il componente che si occupa della gestione del flusso dati, incluso l'input e l'output da e verso la memoria RAM. Nella attuale implementazione è stata usata una solo entity VHDL, che può essere vista come l'insieme di più moduli collegati tra loro come illustrato dallo schema in figura 2. I componenti del datapath sono pilotati dai segnali di controllo generati dalla macchina a stati che controlla il flusso di esecuzione.

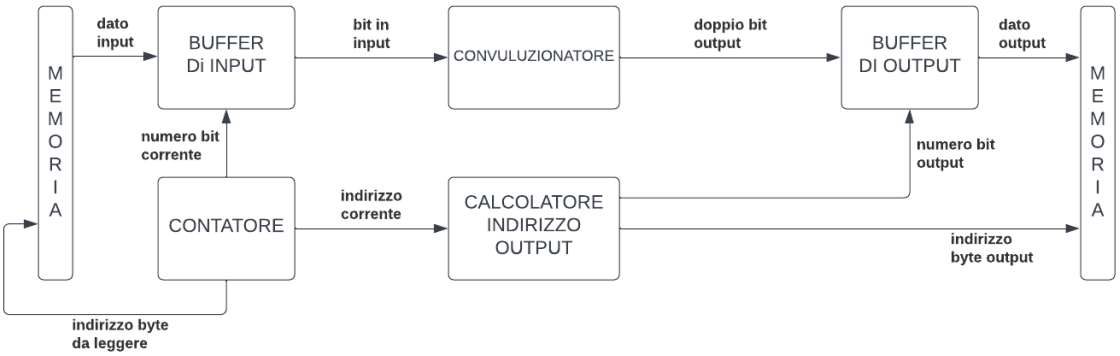

Figura 2: Schema ad alto livello del componente datapath

2.1.1 Input Buffer

Il buffer di input è realizzato con un registro (entity `serializer_register`) che, in presenza del segnale `sr_byte_load` memorizza il dato attualmente presente sul bus `i_data` e lo tiene memorizzato. Il registro ha un parametro aggiuntivo `i_sel` di lunghezza 3 bit (0-7) che permette di specificare quale singolo bit interno al byte memorizzato si vuole in ouput.

2.1.2 Contatore

Per quanto riguarda il contatore la scelta di progetto è stata quella di utilizzare un unico contatore a 11 bit. Il contatore memorizza l'indice del bit corrente relativo alla sequenza. In questo modo, è possibile ottenere il numero del bit corrente all'interno del byte corrente leggendo i 3 bit meno significativi ed il numero del byte corrente all'interno della sequenza leggendo gli 8 più significativi.

In questo modo è possibile utilizzare un solo contatore che automaticamente e contemporaneamente tiene traccia di entrambe le posizioni, automaticamente passando al byte successivo quando viene terminata la lettura del byte corrente.
Il contatore è pilotato da un mux a due ingressi che in base agli ingressi fa fare al registro le seguenti azioni:

- "00" - mantiene nel registro il valore corrente
- "01" - avanza di un bit
- "10" - avanza di un byte (8 bit)
- "11" - resetta a 0 il contatore

2.1.3 Convoluzionatore

Il convoluzionatore (`state_machine.vhd`) è una semplice macchina di Mealy dotata di enable e reset che realizza il convoluzionatore 1/2 come da specifica.

2.1.4 Calcolatore Indirizzo di Output

Il calcolatore dell'indirizzo di output è una ALU completamente combinatoria che, a partire dal numero del bit corrente (in arrivo dal contatore), permette il calcolo dell'indirizzo di output.
Visto che il convoluzionatore per ogni bit di input restituisce sempre 2 bit, il numero del bit in output sarà sempre $2 * curr\_pos + sm\_w\_sel$, dove `curr_pos` è il numero del bit corrente in arrivo dal contatore mentre `sm_w_sel` è un segnale di controllo generato dalla macchina a stati che è attivo (ad 1) se e solo se si vuole scrivere il secondo bit di output dal convoluzionatore. Nel caso in cui la scrittura in memoria sia abilitata, viene anche sommata la costante `1000` a cui viene sottratto `2` visto che un byte in input (quindi due in output) sarebbe usato dal numero di byte della sequenza che non deve essere riportato in output.

A causa del fatto che l'indirizzo di output deve essere (almeno) un bit più lungo di quello di input si è deciso di strutturare tutto il componente per lavorare direttamente su indirizzi da 16 bit (che è la dimensione richiesta dalla memoria), è quindi richiesto il padding sull'indirizzo corrente restituito dal contatore per renderlo anch'esso da 16 bit.

```
-- calcolo della posizione di ouput (2 * currpos [+ 1])
with sm_w_sel select
    out_pos <= std_logic_vector(shift_left(unsigned("00000" & curr_pos), 1)) when '0',
               std_logic_vector(shift_left(unsigned("00000" & curr_pos), 1) + 1) when '1',
               (others => 'X') when others;

-- output inidirizzo memoria
with writesel select
    o_addr <= std_logic_vector(unsigned("000" & out_pos(15 downto 3)) + 998) when '1',
           std_logic_vector(unsigned(X"00" & curr_pos(10 downto 3))) when '0',
           (others => '0') when others;
```

2.1.5 Buffer di Output

Il buffer di output è un componente sequenziale/combinatorio che memorizza temporaneamente il risultato della computazione, in attesa della scrittura in memoria. Il componente si rende necessario a causa dell'indirizzamento al byte della memoria.

Il buffer, quando abilitato alla scrittura, prende uno dei due bit in arrivo dal convoluzionatore tramite un mux pilotato dal segnale `sm_w_sel`, lo shifta a sinistra del numero di posizioni in arrivo dal calcolatore dell'indirizzo di output e lo scrive nella posizione corretta mettendolo in bitwise OR con il valore attuale.

2.2 Macchina a stati

Il modulo è composto da una macchina a 14 stati il cui compito è quello di generare i segnali di controllo in grado di attivare o disattivare le parti del datapath atte a svolgere la funzione richiesta in quello specifico punto dell'esecuzione.

La macchina a stati è descritta come in figura 3, quando non è presente una label sulle transizioni si intende che la transizione è incondizionata oppure, in presenza di altre transizioni in uscita dallo stesso stato, quando la condizione associata all'altra transizione è falsa. E' inoltre presente, ma non riportata nel diagramma, una transizione da tutti gli stati verso `SINIT` quando `i_rst = 1`

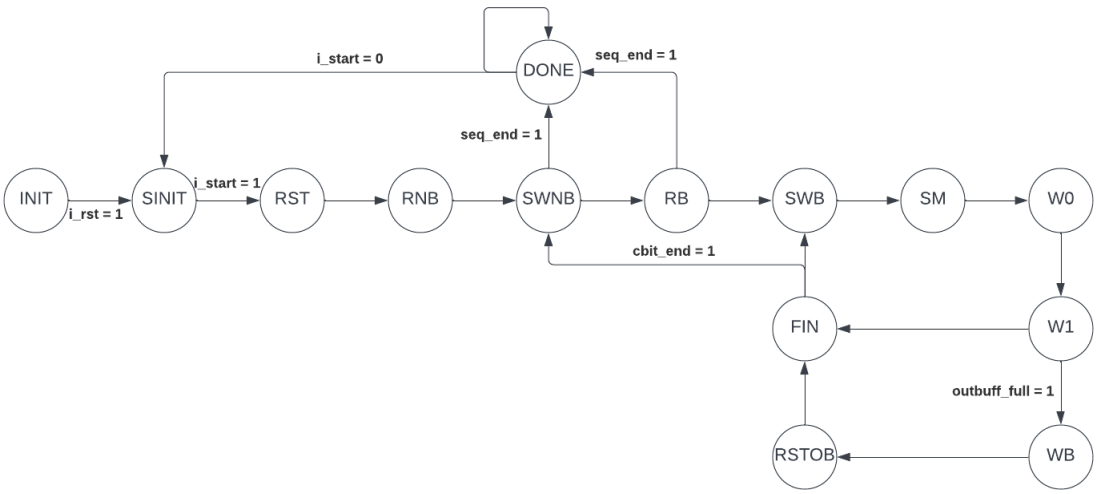

Figura 3: FSM di controllo

Di seguito si riporta la descrizione dei vari stati della macchina di controllo, oltre allo stato dei segnali di controllo.

2.2.1 INIT

Stato iniziale in cui si trova la macchina prima dell'avvio dell'esecuzione.
I segnali di controllo sono tutti disattivati

2.2.2 SINIT

Stato in cui si trova la macchina prima dell'avvio di una nuova sequenza
I segnali di controllo sono tutti disattivati

2.2.3 RST

Reset di tutti i registri interni al datapath per rendere il datapath pronto per una nuova sequenza.
I segnali di controllo attivi sono:

- `sm_rst` che resetta stato corrente e uscita corrente del convoluzionatore
- `curr_mux` ad 11 che resetta a la posizione corrente
- `outbuff_rst` che resetta il buffer di output
- `outbuff_load` che è necessario per resettare il buffer di output

2.2.4 RNB

Lettura del numero di byte dalla RAM ad indirizzo 0.
I segnali di controllo attivi sono:

- `nbytes_load` che attiva il registro che salva il numero di byte da processare
- `curr_mux` ad 10 che fa avanzare la posizione corrente al prossimo byte

2.2.5 SWNB

Stato di stallo in attesa della lettura dalla memoria, verifica lo stato corrente ed il termine della sequenza.
I segnali di controllo sono tutti disattivati

2.2.6 RB

Lettura del byte corrente e memorizzazione nel buffer di input.
I segnali di controllo attivi sono:

- `sr_byte_load` che attiva la lettura del registro che contiene il byte

2.2.7 SWB

Stato di stallo in attesa che il byte venga letto con successo.
I segnali di controllo attivi sono:

- `sr_byte_load` che attiva la lettura del registro che contiene il byte

2.2.8 SM

Il bit corrente è ora presente all'ingresso del convoluzionatore, che viene abilitato e viene generato l'output relativo a quel bit.
I segnali di controllo attivi sono:

- `sr_ena` che attiva il registro serializzatore
- `sm_ena` che attiva il convoluzionatore

2.2.9 W0 e W1

I bit di output vengono scritti nell'output buffer.
Al termine della scrittura si verifica il segnale `outbuff_full` per decidere se è necessario scrivere il buffer di output nella memoria RAM. I segnali di controllo sono:

- `sm_w_sel` rispettivamente a 0 in W0 e 1 in W1 che indica se viene scritto il bit 0 o il bit 1 del convoluzionatore
- `outbuff_load` che abilita la scrittura nel buffer di output

2.2.10 WB

Scrittura del buffer di output in memoria RAM.
I segnali di controllo attivi sono:

- `writesel (o_we)` che attiva il la scrittura in memoria e genera l'indirizzo corretto

2.2.11 RSTOB

Reset del buffer di output.
I segnali di controllo attivi sono:

- `outbuff_rst` che resetta il buffer di output
- `outbuff_load` che è necessario per resettare il buffer di output

2.2.12 FIN

Il singolo bit è stato processato, viene avanzato il contatore.
Si verifica il segnale `cbit_end` per decidere se è necessario passare alla lettura del prossimo byte oppure del prossimo bit. I segnali di controllo attivi sono:

- `curr_mux` a 01 che avanza il contatore di una posizione

2.2.13 DONE

La sequenza è terminata, in attesa di i_start o i_rst per iniziare una nuova sequenza.
I segnali di controllo attivi sono:

- `o_done` a 1 che segnala il termine della computazione

3 Sintesi

3.1 Report Utilization

Il modulo è correttamente sintetizzabile, con la versione corrente di Vivado ed è implementabile su hardware FPGA xc7a200tfbg484 usando 54 Lookup Tables (LUT) e 53 Flip-Flop.
In particolare, i 53 flip flop (1 bit a flip flop) sono così ditribuiti:

- 4 bit per la macchina a stati di controllo
- 2 bit per lo stato corrente del convoluzionatore
- 2 bit per l'uscita corrente del convoluzionatore
- 8 bit per il buffer di output
- 11 bit per la posizione corrente
- 8 bit per il byte corrente
- 8 bit per il numero di bytes da leggere
- 10 bit per segnali di controllo vari

La sintesi non inferisce latch.

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	54	0	0	134600	0.04
LUT as Logic	54	0	0	134600	0.04
LUT as Memory	0	0	0	46200	0.00
Slice Registers	53	0	0	269200	0.02
Register as Flip Flop	53	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Questi dati sono stati ottenuti con `Vivado v.2021.2 (lin64) Build 3367213 Tue Oct 19 02:47:39 MDT 2021` su OS Fedora Linux 35 in data 27 Maggio 2022.

3.2 Report Timing

Utilizzando un constraint con clock 15 ns, Vivado riporta uno Slack (MET) di 11.675 ns ed un Data Path delay di 2.943 ns, quindi c'è margine per alzare la frequenza del sistema.

Contenuto del file di constraint:

```
create_clock -period 15 -name clock -waveform {0 5} [get_ports i_clk]
```

4 Implementazione

La implementation riesce a ottimizzare ulteriormente il sistema, portando il numero di LUT a 51.
Dopo la implementation lo Slack scende a 11.660ns ed il datapath delay sale a 3.317 ns `logic 1.240ns 37.382%, route 2.077ns 62.618%`

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	51	0	0	134600	0.04
LUT as Logic	51	0	0	134600	0.04
LUT as Memory	0	0	0	46200	0.00
Slice Registers	53	0	0	269200	0.02
Register as Flip Flop	53	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

5 Simulazione e casi di test

Il componente è stato simulato su una varietà di casi di test, che comprendono anche gli edge-case più comuni.
Viene riportato l'andamento di alcuni segnali in alcune parti della simulazione per una selezione dei casi testati.
Sono stati effettuati i seguenti casi di test, dei quali per alcuni viene riportata parte della simulazione:

- test con reset durante l'esecuzione
- test con sequenza di lunghezza zero
- test con sequenza di lunghezza massima (0xff)
- test con cambio della sequenza al termine dell'esecuzione
- test con sequenze generate casualmente di lunghezza casuale

5.0.1 Test con sequenza vuota

Nel caso la sequenza sia vuota (RAM[0] = 0), dopo la scrittura del numero di bytes nell'apposito registro, il datapath alzerà immediatamente il segnale `seq_end` che porterà la macchina a stati nello stato di DONE senza mai scrivere nulla in memoria.

```
seq_end <= '1' when unsigned(curr_pos(10 downto 3)) > nbytes else '0';
```

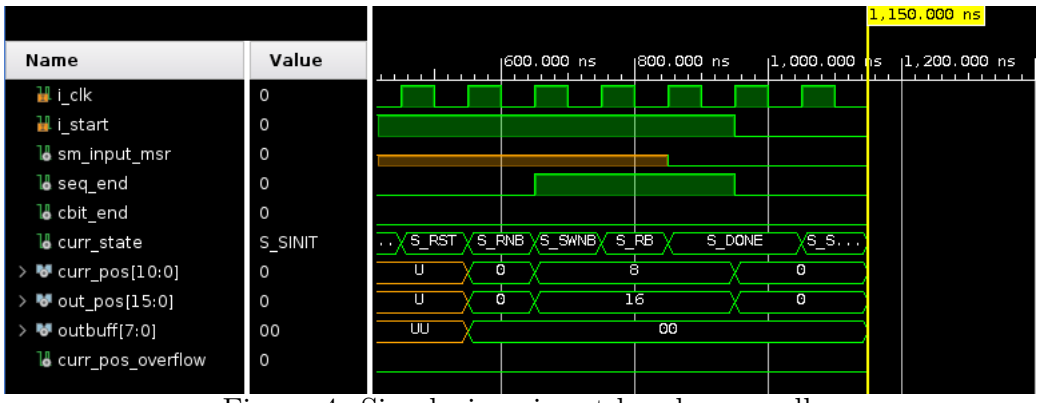

Figura 4: Simulazione input lunghezza nulla

5.0.2 Test con sequenza di lunghezza massima

Il test con sequenza si lunghezza massima è stato quello con il maggior numero di problemi.
Questo perchè dopo aver processato l'ultimo byte il registro `curr_pos` va in overflow nello stato `FIN` tornando a zero. Questo comporta che quando la macchina a stati torna nello stato `SWNB` con il registro `curr_pos` che vale zero, essendo minore di `nbytes` l'esecuzione riprende dall'inizio.
Per risolvere il problema è stato quindi introdotto un segnale `curr_pos_overflow` che viene alzato in maniera sequenziale quando il registro `curr_pos` passa da 255 a 0 ed il segnale che pilota il contatore è in posizione "+1" (01)
Il segnale di fine sequenza è quindi stato modificato per essere attivo anche in caso di overflow.

```
seq_end <= '1' when unsigned(curr_pos(10 downto 3)) > nbytes or
                    curr_pos_overflow = '1' else '0';
```

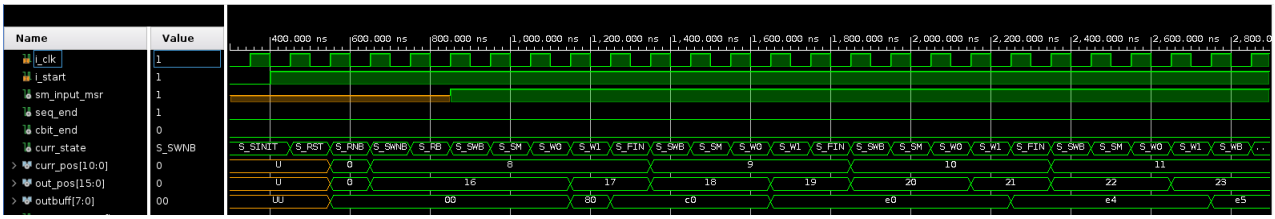

Figura 5: Parte iniziale della simulazione con input di lunghezza massima

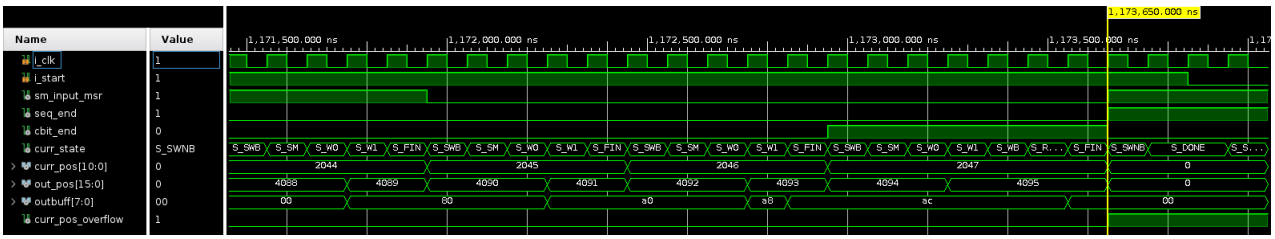

Figura 6: Parte finale della simulazione con input di lunghezza massima, si vede curr_pos andare in overflow e l'alzamento del segnale di overflow

5.0.3 Test sequenza con reset

Quando viene alzato il segnale di reset la FSM di controllo passa sempre nello stato SINIT, ricominciando l'esecuzione.
Quando poi viene alzato il segnale di start la FSM di controllo passa nello stato `RST` che provvedere al reset di tutti i componenti interni del datapath e del convoluzionatore.

```
if i_rst = '1' then
    curr_state <= S_SINIT;
```

5.0.4 Test con sequenze multiple in fila

Alla fine della sequenza la FSM di controllo alza il segnale `o_done`, nel caso in cui `i_start` venga abbassato la macchina si porta nello stato SINIT e si prepara ad accettare una nuova sequenza.

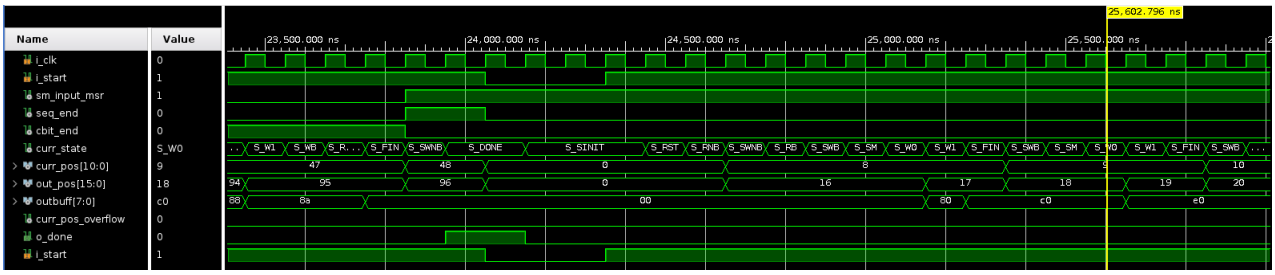

Figura 7: Simulazione fine di una sequenza e inizio della successiva

6 Scelte progettuali e conclusioni

Si è scelto di scrivere il codice VHDL "a più basso livello" possibile in modo che la architettura risultante sia il più possibile simile a quella progettata. Si sarebbe potuto, in alternativa, ricorrere a più strutture `case` ed `if - elsif` al posto di usare MUX e segnali aggiuntivi.

Potrebbe essere possibile rimuovere alcuni stati della macchina a stati, come `SWNB` però questo porterebbe a maggiore complessità di controllo e a nessun risparmio sul numero di bit in quanto passerebbero da 14 a 13 o 12.
Si avrebbe però un risparmio temporale su sequenze molto lunghe.

6.1 Osservazioni

Durante la revisione finale della relazione e del progetto, che è stata effettuata a giugno 2022 su una macchina ed un sistema operativo diversi rispetto a quelli che erano stati utilizzato per lo sviluppo del progetto le settimane precendenti si è notato un leggero scostamento dei valori relativi alla componentistica utilizzata. In particolare il numero di LUT utilizzate è passato da 53 a 62. Avendo a disposizione (tramite git) i log ed i report precendenti ho appurato che il codice sorgente non è cambiato se non per il fatto che le varie entity, originariamente in file separati, sono state messe in un unico file. Questa relazione non è stata modificata e indica i valori precedenti, sono però state aggiunte alcune informazioni circa l'ambiente che è stato utilizzato per ottenere i dati presenti.