Corso di Reti Logiche
Prof. William Fornaciari - Anno scolastico 2019/2020
Politecnico di Milano

Prova Finale (Progetto di Reti logiche)

Rei Barjami (Codice Persona 10588572 - Matricola 887825)
Federica Bucchieri (Codice Persona 10601322 - Matricola 886930)

30/03/2020

Indice

1 Introduzione **1**

2 Architettura **2**

2.1 Macchina a stati 2

2.1.1 Descrizione degli stati 3

2.2 Datapath 5

3 Risultati sperimentali **7**

4 Simulazioni **8**

4.0.1 Test Bench 1: minimo valore codificabile 8

4.0.2 Test Bench 2: massimo valore codificabile 8

4.0.3 Test Bench 3: reset 9

4.0.4 Test Bench 4: reset al termine dell'esecuzione 9

4.0.5 Test Bench 5: doppia Working-Zone 10

4.0.6 Test Bench 6: troncamento 10

5 Conclusioni **11**

1. Introduzione

Scopo di questo progetto è l'implementazione di un metodo di codifica basato su un meccanismo noto come "WORKING ZONE". Dato in ingresso un indirizzo binario, tale valore viene confrontato con gli indirizzi di ogni Working Zone (d'ora in avanti abbreviato come "WZ") e codificato come appartenente a tale WZ, se appartiene ad almeno uno degli intervalli proposti, altrimenti viene riportato in uscita senza alterazioni.

Una working-zone è definita -come da specifica- come un intervallo di indirizzi di dimensione fissa che parte da un indirizzo base.

Nel nostro caso, le WZ a disposizione sono 8 e comprendono un intervallo di 4 indirizzi. Gli indirizzi base delle WZ sono forniti in fase di inizializzazione della macchina insieme al valore da codificare. E' possibile che all'interno della stessa esecuzione, a seguito di un restart della macchina, tali valori cambino prima di fornire la codifica dell'indirizzo richiesto.

La codifica avviene come segue :

- Se l'indirizzo appartiene ad una WZ l'output sarà il seguente - `Wz_Bit & Wz_Num & Wz_Offset`
  Il termine `Wz_Bit` è un bit che viene posto a 1 quando l'indirizzo da codificare appartiene ad una WZ, altrimenti è posto a 0. Il successivo termine `Wz_Num` indica il numero della WZ di appartenenza, nel nostro caso essendo 8 le WZ, 3 bit sono sufficienti allo scopo. Ultimo, `Wz_Offset` esprime l'offeset dell'indirizzo rispetto all'indirizzo di base della working zone, con codifica one-hot espressa su 4 bit. L'`output` quindi sarà codificato su 8 bit.
- Se l'indirizzo non appartiene ad una WZ l'output sarà il seguente - `Wz_Bit & Addr`
  In questo caso il termine `Wz_Bit` viene posto a 0 mentre il termine `Addr` rappresenta l'indirizzo da codificare espresso su 7 bit. Anche in questo caso quindi l'`output` atteso ha dimensione 8 bit.

Esempio

Valore presente in Working-Zone

```
Addr  ->  33
Wz_0  ->  0
Wz_1  ->  10
Wz_2  ->  20
Wz_3  ->  30
Wz_4  ->  40
Wz_5  ->  50
Wz_6  ->  60
Wz_7  ->  70

Output  ->  1-011-1000
```

Valore non presente in nessuna Working-Zone

```
Addr  ->  33
Wz_0  ->  0
Wz_1  ->  10
Wz_2  ->  20
Wz_3  ->  40
Wz_4  ->  50
Wz_5  ->  60
Wz_6  ->  70
Wz_7  ->  80

Output  ->  0-0100001
```

2. Architettura

Per implementare questo metodo di codifica, si è scelto di operare costruendo una Macchina a stati finiti e di implementare insieme ad essa il relativo Datapath.

2.1 Macchina a stati

La macchina costruita è composta da 25 stati. La Figura 2.1 presenta uno schema dell'architettura proposta, mettendo in risalto i principali passaggi di stato ed i segnali rilevanti che vengono attivati in un determinato stato. L'interfaccia del componente è la seguente, come da specifica:

```
entity project_reti_logiche is
      Port (
         i_clk :  in STD_LOGIC;
         i_start :  in STD_LOGIC;
         i_rst :  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;
```

Allo scopo è stato definito il tipo personalizzato S per la rappresentazione degli stati e sono stati utilizzati i seguenti segnali, utili alla comprensione del funzionamento della macchina a stati:

```
signal RegAddr_load :  STD_LOGIC;
signal RegTemp_load :  STD_LOGIC;
signal wz_num :  STD_LOGIC_VECTOR (2 downto 0);
signal wz_bit :  STD_LOGIC;
signal cur_state, next_state, temp_state :  S;
```

2.1.1 Descrizione degli stati

I primi quattro stati della macchina sono stati di inizializzazione che vengono ripetuti ad ogni restart della macchina.

- `S0:` Stato di partenza e di reset della macchina in cui tutti i segnali sono posti a 0. Il passaggio allo stato successivo avviene al momento della ricezione del segnale `i_start=1`.
- `S1:` Viene abilitata la lettura dalla memoria ponendo il segnale `o_en=1`.
- `S2:` `o_address` viene posto a `8` in modo tale da leggere come primo valore il contenuto dell'indirizzo 8 che conterrà `Addr` da codificare.
- `Address_Setting:` In questo stato `RegAddr_load=1` e ci si prepara a ricevere il valore di `Addr` dalla memoria.
- `Address_Initializing:` Il valore di `Addr` arriva al datapath e viene immagazzinato dentro `RegAddr`.

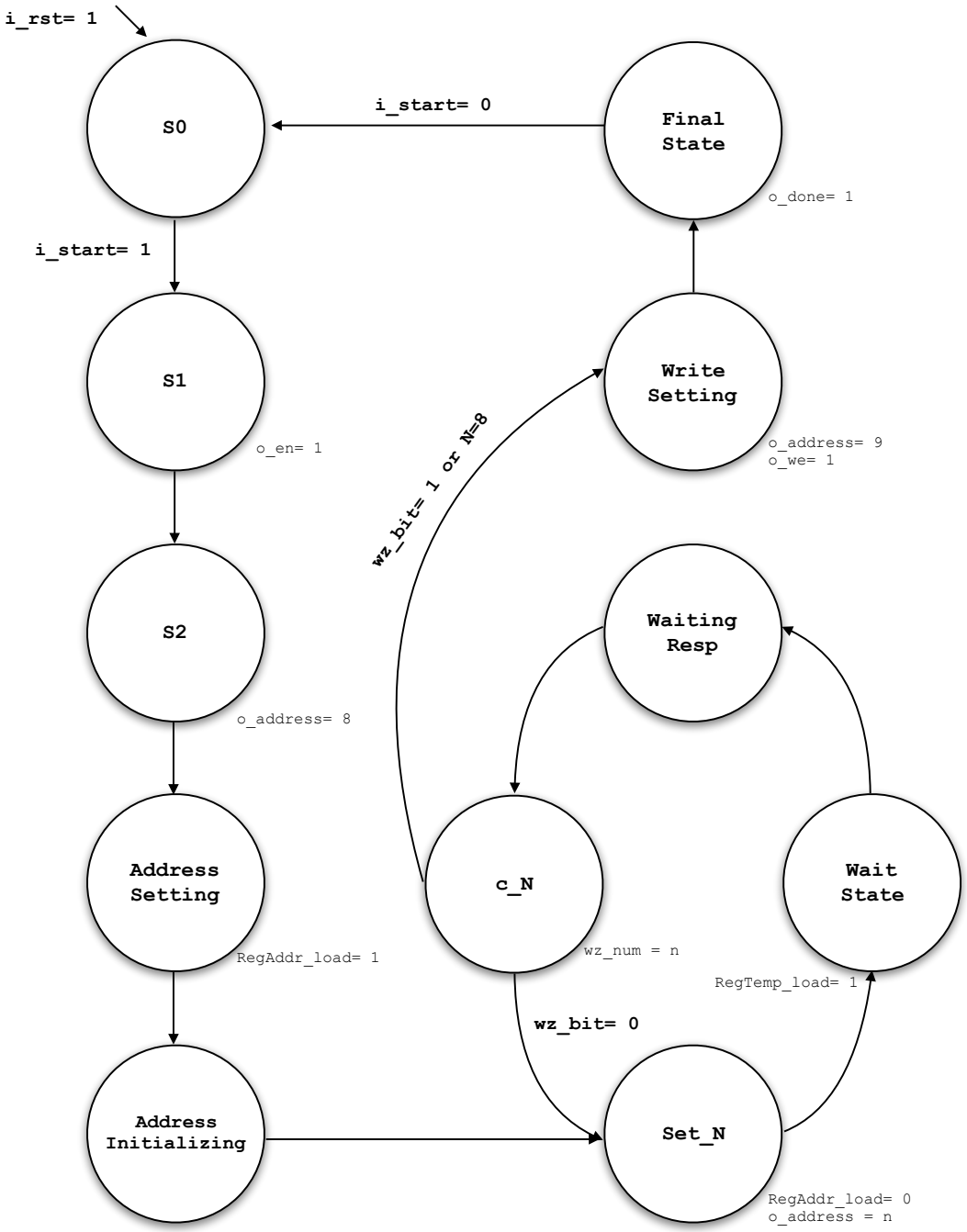
I successivi quattro stati si ripetono per ogni Working-Zone quindi per praticità si è deciso di rappresentarli con la dicitura `"N"`. Essi si svolgono ciclicamente e la condizione di uscita dal ciclo è univoca tranne che per l'ultimo stato `c_8` che prevede in ogni caso la prosecuzione dell'esecuzione ed il passaggio allo stato `Write_Setting`.

- `Set_N:` Viene settato il valore di `o_address` di interesse così da richiedere alla memoria l'indirizzo di base di ogni WZ.
- `Wait_State:` Il valore richiesto alla memoria arriva alla macchina a stati grazie al segnale `RegTemp_load=1` e diventa utilizzabile all'interno dell'architettura.
- `Waiting_Response:` Il valore richiesto alla memoria arriva ora anche al Datapath che risponde settando `wz_bit` a seconda che `Addr` appartenga o meno alla WZ considerata in questo ciclo d'esecuzione.
- `c_N:` Stato di Controllo che verifica il valore di `wz_bit` e decide quindi quale sarà lo stato successivo dell'esecuzione. Se `wz_bit=1` si passa allo stato di `Write_Setting`, altrimenti si passa a `Set_N+1` ricominciando il ciclo.

Gli ultimi due stati rappresentano la fase di scrittura del risultato in memoria.

- `Write_Setting:` Si richiede alla memoria di poter scrivere l'output elaborato impostando `o_we=1` e settando `o_address=9`. Viene riportato contemporaneamente `RegTemp_load=0`.
- `Final_State:` L'output viene effettivamente scritto in memoria quindi si dichiara la fine dell'esecuzione con `o_done=1`. Quindi si ritorna allo stato di partenza quando il segnale `i_start` viene riportato a zero.

Figura 2.1: Schema semplificato della Macchina a stati



2.2 Datapath

La nostra implementazione fa uso di un Datapath a supporto delle operazioni che avvengono durante i vari stati del ciclo della Macchina a stati. La Figura 2.2 presenta uno schema esplicativo dell'insieme delle unità di calcolo proposte. L'interfaccia del modulo progettato è la seguente:

```
entity DataPath is
      Port(
         i_clk :  in STD_LOGIC;
         i_res :  in STD_LOGIC;
         wz_num:  in STD_LOGIC_VECTOR (2 downto 0);
         i_data :  in STD_LOGIC_VECTOR (7 downto 0);
         o_data :  out STD_LOGIC_VECTOR (7 downto 0);
         RegAddr_load:  in STD_LOGIC;
         RegTemp_load:  in STD_LOGIC;
         wz_bit :  inout STD_LOGIC
         );
end Datapath;


architecture Behavioral of DataPath is
         signal RegAddr :  STD_LOGIC_VECTOR (7 downto 0);
         signal RegTemp:  STD_LOGIC_VECTOR (7 downto 0);
         signal sub:  STD_LOGIC_VECTOR (7 downto 0);
         signal wz_offset:  STD_LOGIC_VECTOR (3 downto 0);
         signal concat:  STD_LOGIC_VECTOR ( 7 downto 0);
```

La computazione si svolge nel seguente modo:

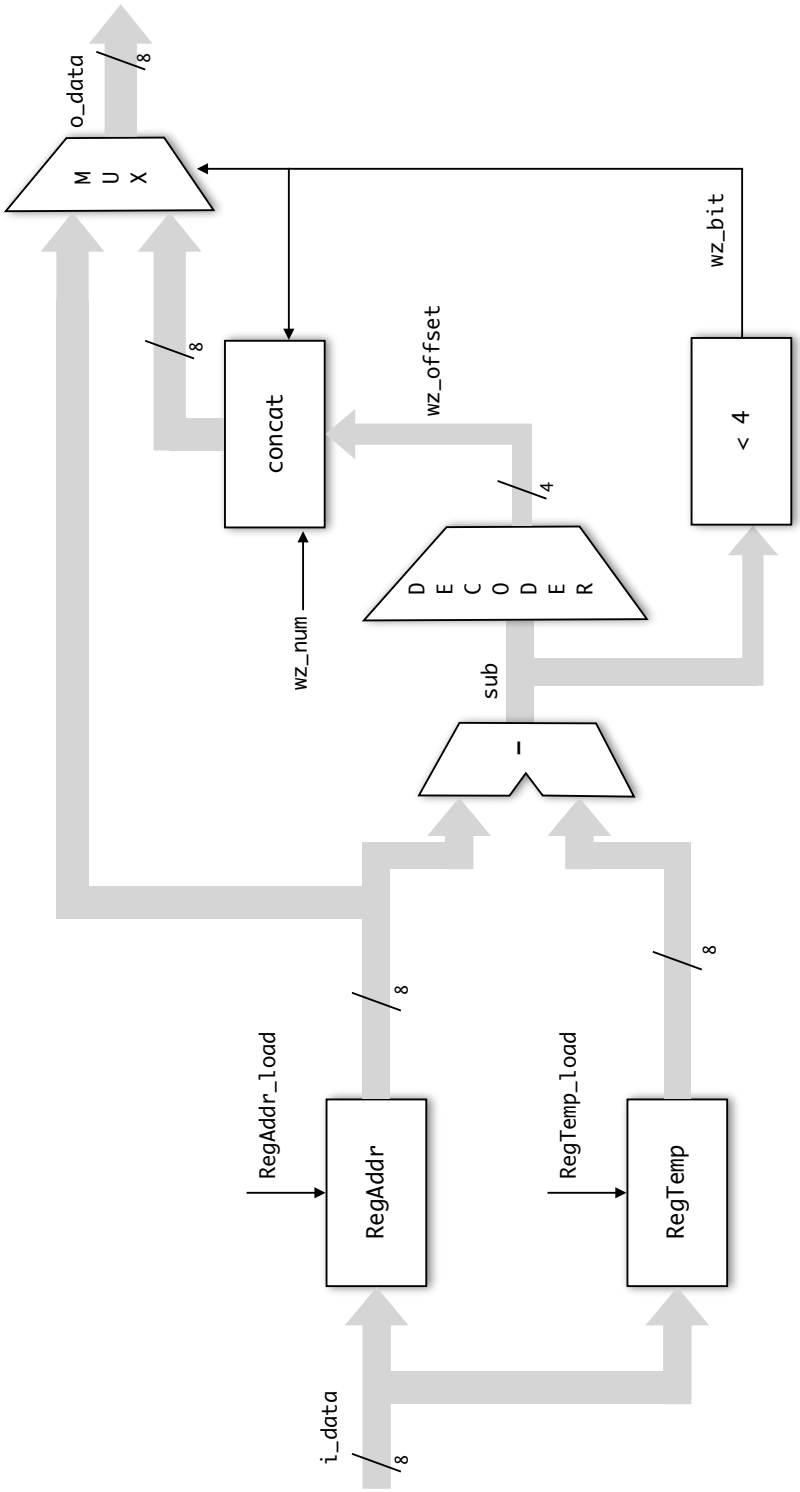
Dopo aver inizializzato `RegAddr` e `RegTemp`, i due registri vengono sfruttati per salvare i valori rispettivamente di `Addr` e dell'indirizzo base della Working Zone considerata ad ogni giro. Il caricamento avviene grazie ai segnali di load (`RegAddr_load` e `RegTemp_load`) gestiti interamente dalla macchina a stati. In questa fase, l'ottavo bit di ogni indirizzo in ingresso viene posto a `'0'` per evitare valori in ingresso che violino le specifiche superando la lunghezza di 7 bit previsti.

Ogni indirizzo di base contenuto in `RegTemp` viene confrontato con il valore contenuto in `RegAddr` tramite una sottrazione il cui risultato è salvato nel registro `sub`. Se il risultato è inferiore a `4`, l'indirizzo da codificare `Addr` appartiene alla WZ in considerazione e di conseguenza `wz_bit` viene posto a `1`. Altrimenti il valore di tale segnale `inout` che comunica con la Macchina a stati resta posto a `0`.

Grazie all'analisi del valore di `sub` viene anche definito il contenuto di `wz_offset` tramite un Decoder. Si ottengono così tutti gli elementi necessari per la computazione dell'output:

Un multiplexer, grazie al valore di selezione fornito da `wz_bit` fornisce in uscita all'interno del segnale `o_data` il valore di `RegAddr` quando si ha `wz_bit=0` altrimenti per `wz_bit=1` fornisce in uscita l'indirizzo contenuto in `concat`, formato dalla concatenazione di `wz_bit & wz_num  wz_offset`. Tale risultato viene quindi trasmesso al componente principale e scritto poi in memoria.

Figura 2.2: Schema del Datapath



3. Risultati sperimentali

La macchina da noi progettata è risultata correttamente sintetizzabile in modo da permetterci di svolgere correttamente i test anche in post sintesi. In particolare sono state svolte soltanto le simulazioni di tipo *behavioural* in pre sintesi e, in post sintesi di tipo *functional*.

Riteniamo utile confrontare due tempi di esecuzione significativi: il tempo di esecuzione per la codifica del minimo valore codificabile `(0)` ed del massimo valore codificabile `(127)`.

1. Tempo di esecuzione *(behavioural)* per codificare il valore `0000000` con appartenenza alla `WZ_0` : **1550 ns**.
2. Tempo di esecuzione *(behavioural)* per codificare il valore `1111111` con appartenenza alla `WZ_7`: **4350 ns**.

Per avere un parametro che quantifichi rispetto al tempo totale a disposizione quale sia il tempo del path peggiore, bisogna considerare il *Worst Negative Slack* riportato nella tabella **Design Timing Summary** del componente sintetizzato.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	96,008 ns	Worst Hold Slack (WHS):	0,163 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	25	Total Number of Endpoints:	25	Total Number of Endpoints:	21

All user specified timing constraints are met.

Figura 3.1: Design Timing Summary

4. Simulazioni

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il test bench di esempio, sono stati definiti 9 ulteriori casi di test con l'intento di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione. Particolare attenzione è stata posta nell'individuare i corner case. Di seguito viene fornita una breve descrizione dei test bench più importanti utilizzati e per quelli più significativi viene anche mostrato l'effettivo corretto funzionamento.

4.0.1 Test Bench 1: minimo valore codificabile

Questo test valuta il corretto funzionamento della macchina quando il valore da codificare appartiene alla prima Working-Zone e ha distanza dal suo indirizzo pari a 0 (`wz_offset="0001"`). Il test è stato estremizzato settando tutti gli indirizzi base delle WZ al valore 127, mentre WZ_0 = '0'. L'input fornito è il minimo valore codificabile dal componente.

INPUT: `0000000` (0)
OUTPUT ATTESO: `1-000-0001` (129)

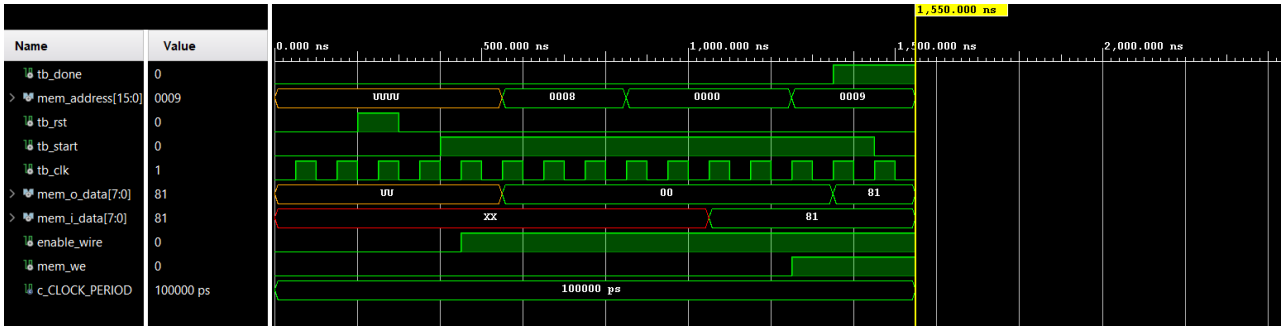


Figura 4.1: Grafico dell'andamento del test bench 1.

4.0.2 Test Bench 2: massimo valore codificabile

Questo test valuta il corretto funzionamento della macchina quando il valore da codificare appartiene all'ultima Working-Zone e ha distanza dal suo indirizzo pari a 0 (`wz_offset="0001"`). Il test è stato estremizzato settando tutti gli indirizzi base delle WZ a '0', mentre WZ_7 = '127'. L'input fornito è il massimo valore codificabile dal componente.

INPUT: `1111111` (127)
OUTPUT ATTESO: `1-111-0001` (241)

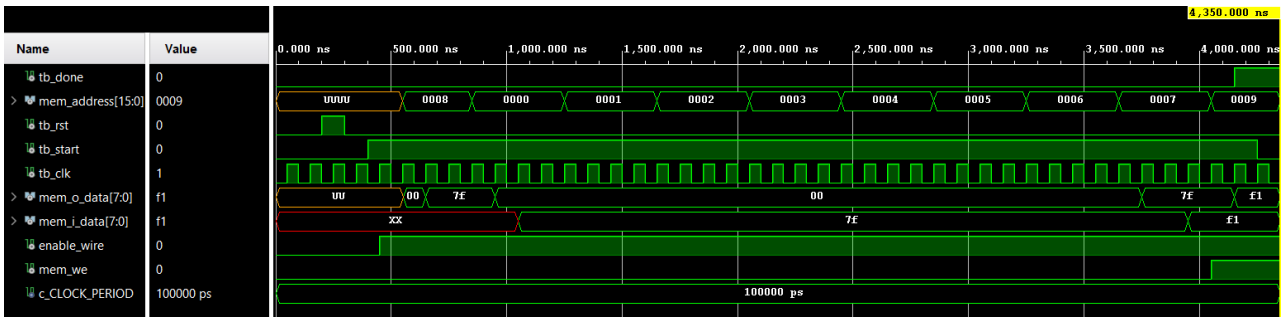


Figura 4.2: Grafico dell'andamento del test bench 2.

4.0.3 Test Bench 3: reset

Questo test valuta il corretto funzionamento della macchina quando viene inviato un segnale di `reset` ad esecuzione avviata. Il valore degli indirizzi base delle WZ e del valore da codificare sono casuali. In particolare, l'indirizzo scelto appartiene alla WZ_4 che ha base 62, l'esecuzione viene avviata ponendo `i_start=1` e dopo `300ns` viene dato `i_rst=1`. L'esecuzione riprende con un ulteriore `i_start=1` e prosegue fino a completare la codifica.

INPUT: `1000001` (65)
OUTPUT ATTESO: `1-100-1000` (200)

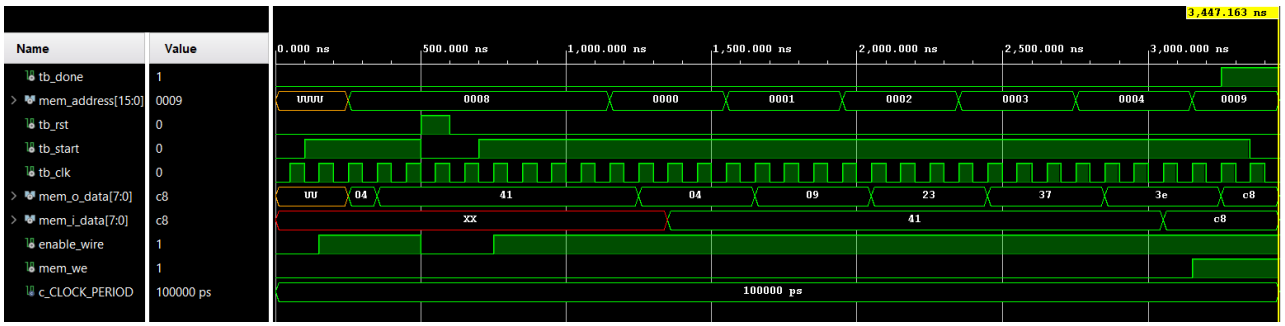


Figura 4.3: Grafico dell'andamento del test bench 3.

4.0.4 Test Bench 4: reset al termine dell'esecuzione

Questo test valuta il corretto funzionamento della macchina quando viene inviato un segnale di `reset` a codifica terminata, dopo aver posto il segnale `o_done=1`. Il valore degli indirizzi base delle WZ e del valore da codificare sono casuali. In particolare, l'indirizzo scelto appartiene alla WZ_6 che ha base 93.

INPUT: `1011110` (94)
OUTPUT ATTESO: `1-110-0010` (226)

4.0.5 Test Bench 5: doppia Working-Zone

Questo test valuta il corretto funzionamento della macchina quando il valore da codificare appartiene a due WZ consecutive. L'input da codificare appartiene sia alla WZ_3 (55) che alla WZ_4 (56) ed è pari a 57. Si è scelto di utilizzare come criterio di appartenenza l'ordine delle WZ, quindi in questo caso ci aspettiamo che venga codificato come appartenente alla WZ_3.

INPUT: `0111001` (57)
OUTPUT ATTESO: `1-011-0100` (180)

4.0.6 Test Bench 6: troncamento

Questo test valuta il corretto funzionamento della macchina quando l'input fornito supera il numero di bit consentito. Dando in input l'indirizzo 140, esso viene troncato al settimo bit, diventando così 12 in binario (1-0001100). Il valore troncato appartiene alla WZ_1 e viene quindi codificato di conseguenza.

INPUT: `10001100` (140)
OUTPUT ATTESO: `1-001-1000` (152)

5. Conclusioni

Nello svolgere l'incarico assegnatoci, le scelte progettuali che sono state attuate, sono state indirizzate all'ottimizzazione ed alla semplicità del codice.

Dal punto di vista dell'*ottimizzazione*, la gestione del ciclo degli stati di computazione è svolto in maniera sequanziale partendo dalla prima Working-Zone e producendo in itinere il risultato di codifica. In questo modo, una volta identificata l'eventuale WZ di appartenenza la computazione termina senza verificare tutte le restanti, riducendo quindi il tempo di esecuzione complessivo.

Dal punto di vista della *semplicità*, le scelte dei moduli del datapath sono ricadute su componenti semplici: un sottrattore a due ingressi, un decoder a quattro uscite, due multiplexer a due ingressi e l'utilizzo di tre soli registri, più il quarto utilizzato per la concatenazione dei segnali necessari alla produzione dell'output.

Inoltre, i risultati ottenuti in fase di testing del componente sono risultati tutti positivi sia in modalità behavioural, sia in post sintesi in modalità functional, quindi ci riteniamo soddisfatti della *robustezza* del codice proposto.