Politecnico di Milano

Prova Finale Reti Logiche

Anno accademico 2022-2023

Elia Pontiggia
Matricola 956690 – Codice Persona 10716792

Introduzione

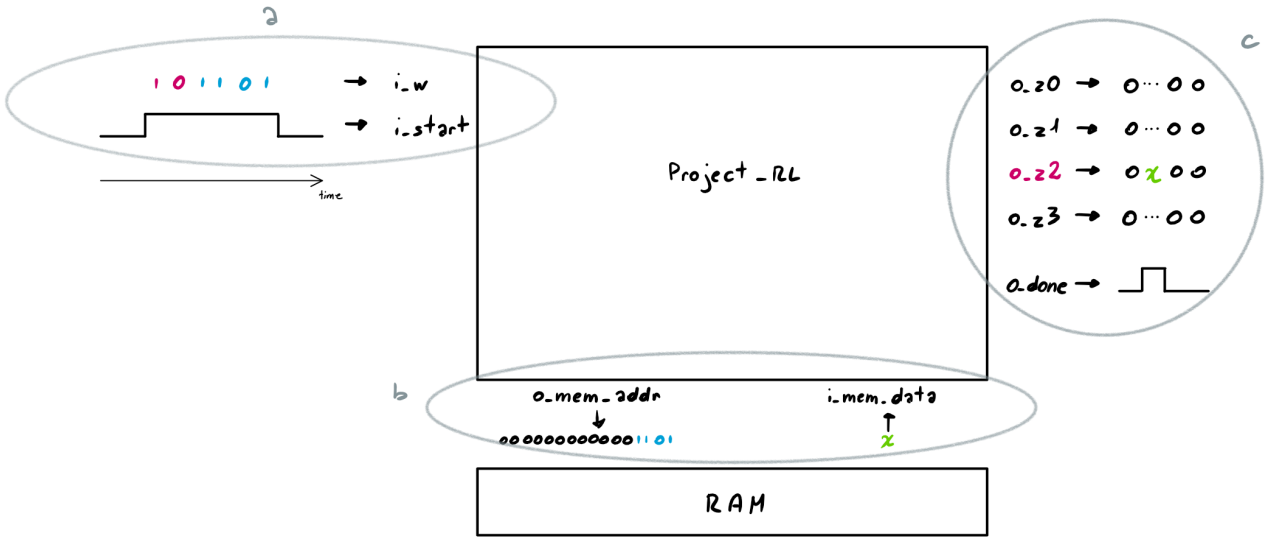
Scopo del progetto è quello di realizzare una macchina con un ingresso principale [`i_w`] e quattro uscite principali [`o_z0 o_z1 o_z2 o_z3`] ognuna da 8 bit, eseguendo un processo che si può riassumere in tre passi:

a. Riceve in input serialmente in notazione big-endian l'indirizzo di una delle uscite su due bit e un indirizzo di memoria di massimo 16 bit
b. Legge da una memoria RAM il dato (8 bit) memorizzato all'indirizzo dato in input, eventualmente espandendolo a sinistra con `0`
c. Sulla porta letta durante la fase (a) espone il dato appena letto; sulle altre porte espone lo stesso dato esposto alla lettura dell'input precedente. Se su una porta non è ancora stato elaborato alcun dato, mostrerà `00000000`

Oltre agli I/O già citati e all'input per il segnale di clock, il circuito presenta:

- I segnali I/O utili per il corretto scambio di dati con la RAM
- Il segnale input `i_start`, che rimane asserito il tempo durante il quale i bit che vengono campionati da `i_w` rappresentano un input
- Il segnale output `o_done`, che verrà asserito al massimo per un ciclo di clock per ogni sequenza di input. Nello stesso ciclo in cui esso sarà al livello alto, le quattro porte di output mostreranno il dato da esse memorizzato
- Il segnale input `i_rst`, segnale di reset sincrono che azzera i dati memorizzati nelle quattro porte di output ed eventualmente interrompe l'ultima elaborazione dell'input (più avanti discuteremo del comportamento della macchina in base all'istante in cui viene asserito questo segnale)

Un esempio di funzionamento della macchina, seppur in maniera molto semplificata, è rappresentato nella figura seguente:



Architettura

Lo schema del circuito globale è rappresentato in coda alla relazione. Per chiarezza, è stato fatto uso di alcuni abusi di notazione:

- Sono state omesse le linee rappresentanti il segnale di reset dato in input da `i_rst`, collegato alla porta `clr` di tutti i flip-flop e di tutti i registri
- Per ogni componente SR (sia flip-flop che registro), le linee S e R sono state rappresentate come una unica linea
- Non viene rappresentato il segnale di clock che, ovviamente, è connesso opportunamente a tutti i componenti di memoria
- I moduli combinatori vengono rappresentati black-box
- Nel caso in cui ogni bit di un array di linee debba essere messo in `and` con un unico segnale, è stata rappresentata una sola porta `and` e non un numero pari alla cardinalità dell'array

Per indicare i moduli che compongono il circuito, verranno utilizzati i colori con cui sono rappresentati gli stessi nello schema

Verde: monostable_circuit

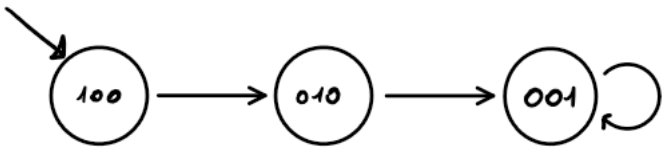
Rappresenta un circuito il cui obiettivo è quello di portare sull'uscita un impulso nel momento in cui comincia e finisce l'input, ovvero rileva i fronti di salita e di discesa del segnale `i_start`

Nel progetto, è stato implementato con un FF-D (che "ritarda" di un ciclo di clock il segnale `i_start`), il cui output è in `xor` con il segnale `i_start`

Blu: input_splitter

L'obiettivo del modulo è di distinguere i bit presenti sul bus `data_in` in base a se essi rappresentano un bit di indirizzo dell'uscita oppure un bit dell'indirizzo da leggere dalla RAM

È stato implementato costruendo una macchina a stati simile ad un contatore one-hot, il cui diagramma è rappresentato nell'immagine seguente



In particolare:

- Stato 100: il bit su `data_in` rappresenta il MSB dell'indirizzo della porta di uscita
- Stato 010: rappresenta l'LSB della porta di uscita
- Stato 001: si sta leggendo un bit dell'indirizzo di memoria

Il reset della macchina avviene sul fronte di discesa del ciclo di clock in cui si è campionato il fronte di salita del segnale `i_start` (in altre parole, il reset è dato da `monostable and i_start`)

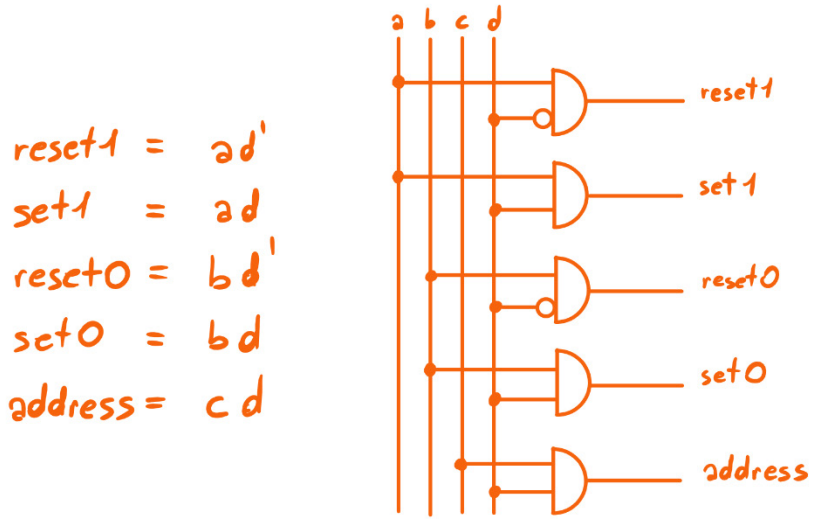
Arancio: logic_block_a

È il modulo combinatorio che si occupa di elaborare il bit letto da `i_w` e, sulla base delle informazioni lette dal modulo blu, eseguire l'istruzione corretta, che sia settare/resettare l'opportuno FF che memorizza quale delle 4 uscite andrà utilizzata, oppure semplicemente propagare il bit di indirizzo di memoria

Benché il funzionamento del modulo sia simile a quello di un decoder, per l'implementazione si è scelto di sintetizzare le 6 funzioni logiche corrispondenti alle 6 uscite del blocco, rappresentate dalla seguente tabella di verità (condensata)

a	b	c	d	
exit_msb	exit_lsb	address_bit	w	effect
1	0	0	0	reset exit 1
1	0	0	1	set exit 1
0	1	0	0	reset exit 0
0	1	0	1	set exit 0
0	0	1	0	propaga 0
0	0	1	1	propaga 1
others				–

E, di conseguenza, ottimizzate come



Azzurro: serial_reg

È il registro seriale SIPO a scorrimento singolo che permette di realizzare una caratteristica fondamentale del circuito, ovvero che gli indirizzi in input vengano dati un bit alla volta in big endian (più significativo → meno siginificativo) e infine permette di avere l'LSB automaticamente nella posizione corretta dei 16 bit preceduto da soli `0`, senza necessitare di ulteriori rielaborazioni. Esso viene azzerato quando è dato il reset tramite `i_rst` e quando viene resettata la macchina a stati nel modulo blu

L'implementazione del modulo vede 16 FF-D collegati a cascata; il primo ha come D il bit `addr` uscente dal modulo arancio. Ogni bit dell'uscita andrà messo in and con il segnale `o_mem_en` per poter leggere correttamente dalla memoria. Ogni FF è stato dichiarato tramite port map e non con un processo per evitare problemi di timing in fase di test post-sintesi

Violetto: exit_selector

Si occupa di memorizzare i primi due bit dell'input **`i_w`** e quindi di conoscere l'uscita su cui bisognerà mostrare il dato

Viene implementato con una coppia di FF-SR, ognuno dei quali riceve il comando opportuno dal modulo arancione, già trattato

Giallo: logic_block_b

È responsabile di prendere in input il dato letto dalla memoria e instradarlo (tramite due array, quello di set e di reset) verso il registro che fa riferimento alla singola uscita corretta

Il segnale **`interEF`** triggera il processo che descrive il modulo. Prima esso compila i vettori di set - reset bit a bit con i valori opportuni leggendo il dato da **`i_mem_data`**, e successivamente, sulla base dei valori passati dal modulo violetto, si assegnano ai canali di uscita corretti solo quando **`interEF`** è posto ad 1 con un assegnamento condizionale

Lime: out_reg

Sono quattro moduli, tutti uguali, che memorizzano il dato letto da memoria (passato dal modulo giallo) e lo mostrano sull'uscita corrispondente durante il ciclo di clock successivo al livello alto del segnale **`canWeExit`**, collegato a **`o_done`** da un FF-D

Per l'implementazione si sono utilizzati per ogni modulo un registro parallelo SR. Ogni bit in uscita viene messo in **`and`** con il segnale **`canWeExit`** e, prima dell'uscita, lo stream passa per un registro parallelo di tipo D. Quest'ultimo componente è stato aggiunto per limitare il più possibile la diversità nella propagazione dei segnali in ingresso alle porte **`and`**, dovuta all'elevato fan-out di **`delayE`** (mostrato chiaramente nello schema)

Nero: delay

Rappresentano dei FF-D con il solo scopo di "rallentare" il segnale in input per permettere la coordinazione temporale tra i vari moduli, sia esso un segnale che arriva da **`i_w`** o che arriva dal modulo verde

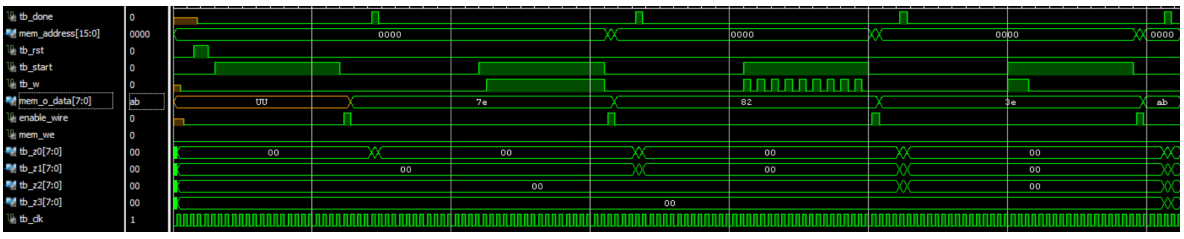
Risultati sperimentali

Test

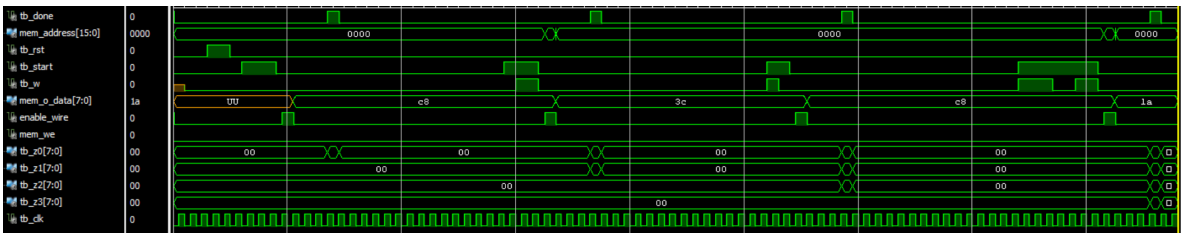
Si possono individuare due gruppi di test da effettuare, per ognuno è riportato il frammento di test specifico che dimostra che la macchina si comporta come desiderato

Il primo copre le possibili combinazioni di input dati / output attesi dalla macchina

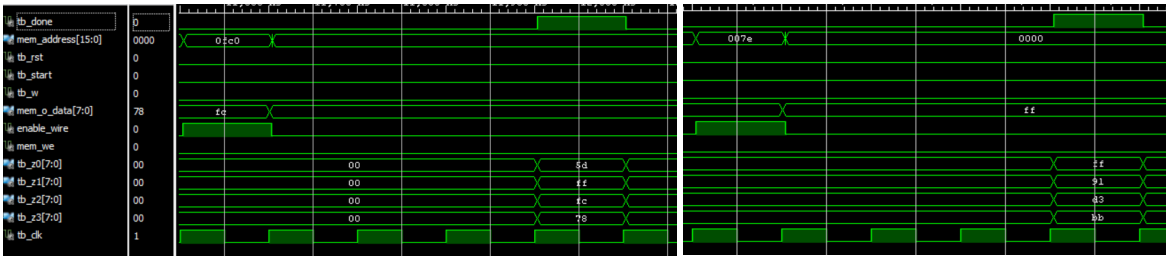
1. Caso più semplice: il test verifica che la macchina reagisca correttamente quando vengono dati in input indirizzi completi (16bit), vengono testati in particolare i due casi limite, ovvero l'indirizzo **`0000000000000000`** e **`1111111111111111`**



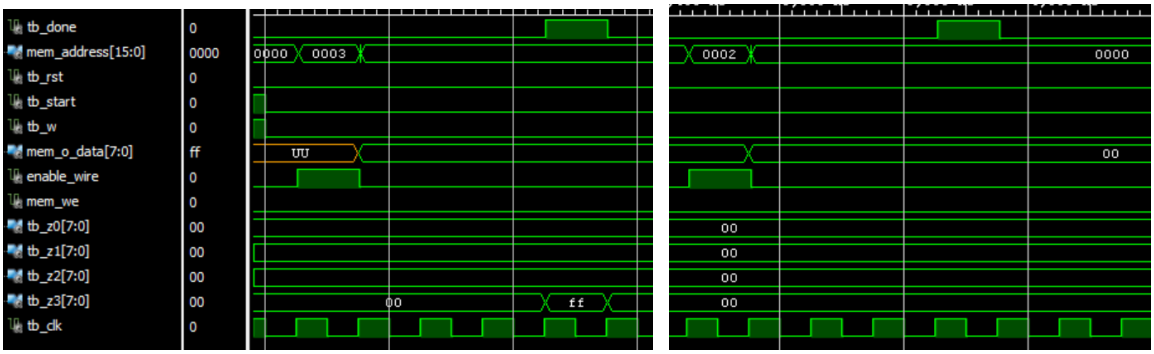
2. Non è necessario che gli indirizzi siano dati completi; si vuole quindi osservare il comportamento della macchina nei casi opposti al test sopra; i casi limite sottoposti (parte di indirizzo) sono quindi `0`, `1`, indirizzo vuoto



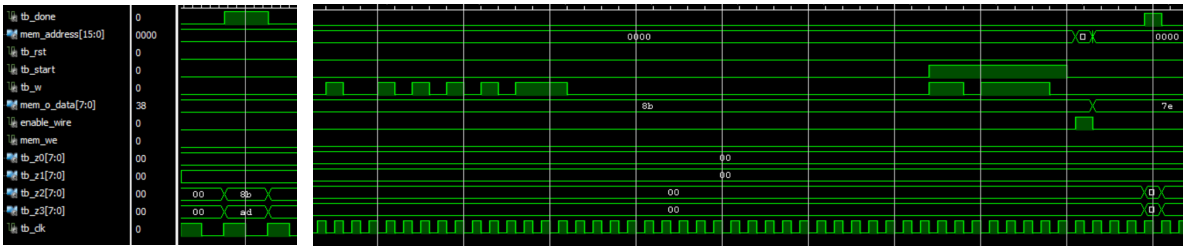
3. Ci si vuole accertare che la macchina sia in grado di sovrascrivere i dati precedentemente immagazzinati nei registri di output



4. Si testano ora i casi limite non per l'input ma per l'output, si danno quindi in input alla macchina prima un indirizzo di memoria in cui è salvato il dato `11111111`. Successivamente, sullo stesso canale di uscita, viene richiesto un indirizzo in cui risiede il dato `00000000`

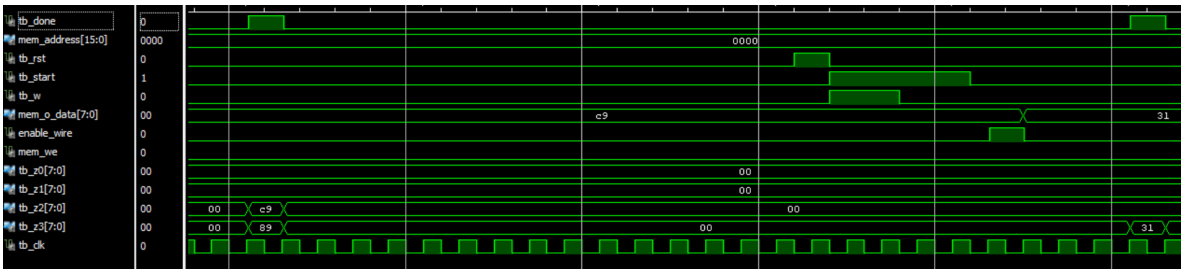


5. Come ultimo dei test sull'input si vuole verificare che effettivamente non vengano considerati i bit su `i_w` se `i_start` ha valore `0`

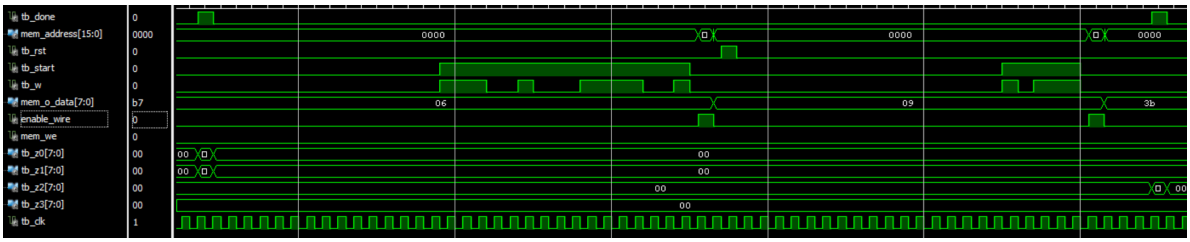


Verificato il corretto funzionamento della macchina a fronte di ingressi "lineari", si vuole ora verificare che il circuito reagisca correttamente ai segnali di reset, qualsiasi sia l'istante di tempo in cui vengono dati

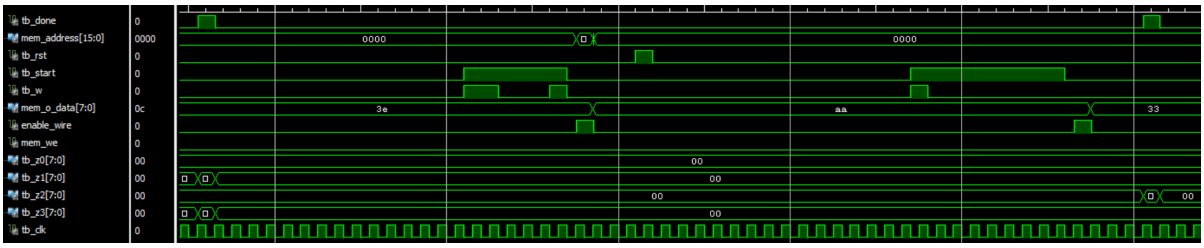
6. Caso più semplice: il segnale di reset viene dato in un momento in cui la macchina non sta elaborando l'input e sta dunque aspettando la salita del segnale `i_start`



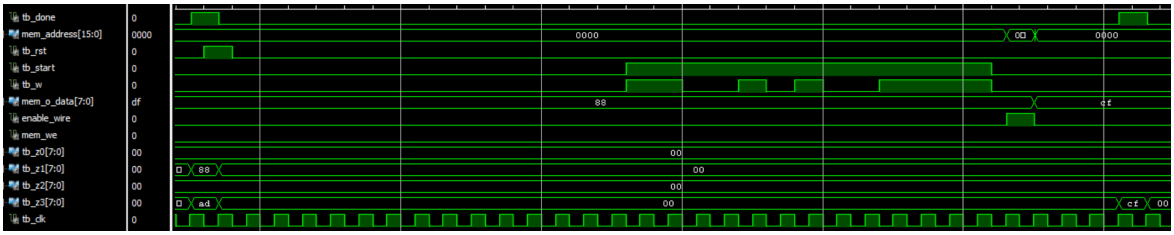
7. L'impulso di reset viene dato in uno dei 4 cicli di clock che intercorrono la discesa di `i_start` e la salita di `o_done`, durante i quali la macchina sta elaborando l'input e salvando il dato restituito dalla RAM



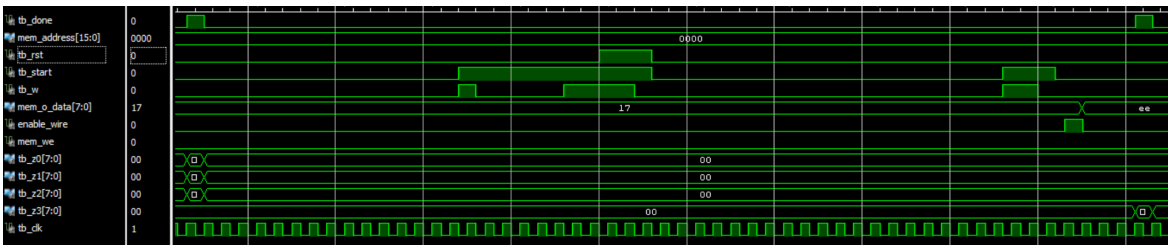
8. È un caso limite del test numero 6: il reset viene dato durante il ciclo di clock immediatamente antecedente al ciclo in cui salirebbe il segnale `o_done`, verificando che i registri d nei moduli lime facciano in tempo ad effettuare il reset



9. In questo caso, invece, il segnale di reset sale durante lo stesso ciclo di clock in cui è alto il segnale `o_done` (un ciclo successivo al test numero 7), verificando che, durante quel periodo, le uscite rimangano con i valori corretti, mentre dalla sequenza successiva la macchina sia correttamente resettata



10. Caso opposto al precedente e contemporaneamente altro caso limite del test numero 6; il segnale reset viene dato mentre il segnale `i_start` è alto, assicurandosi che l'input che sta venendo campionato venga ignorato



Report di sintesi

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	71	0	134600	0.05
LUT as Logic	71	0	134600	0.05
LUT as Memory	0	0	46200	0.00
Slice Registers	107	0	269200	0.04
Register as Flip Flop	107	0	269200	0.04
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Si nota che, rispetto ad una prima stesura del progetto, in cui si implementava l'intero circuito come un'unica macchina a stati governata da un singolo processo, vengono usati molti meno registri (107 contro circa 200) e, in questo caso, nessuno di essi è latch, caratteristica che aumenta la robustezza del circuito

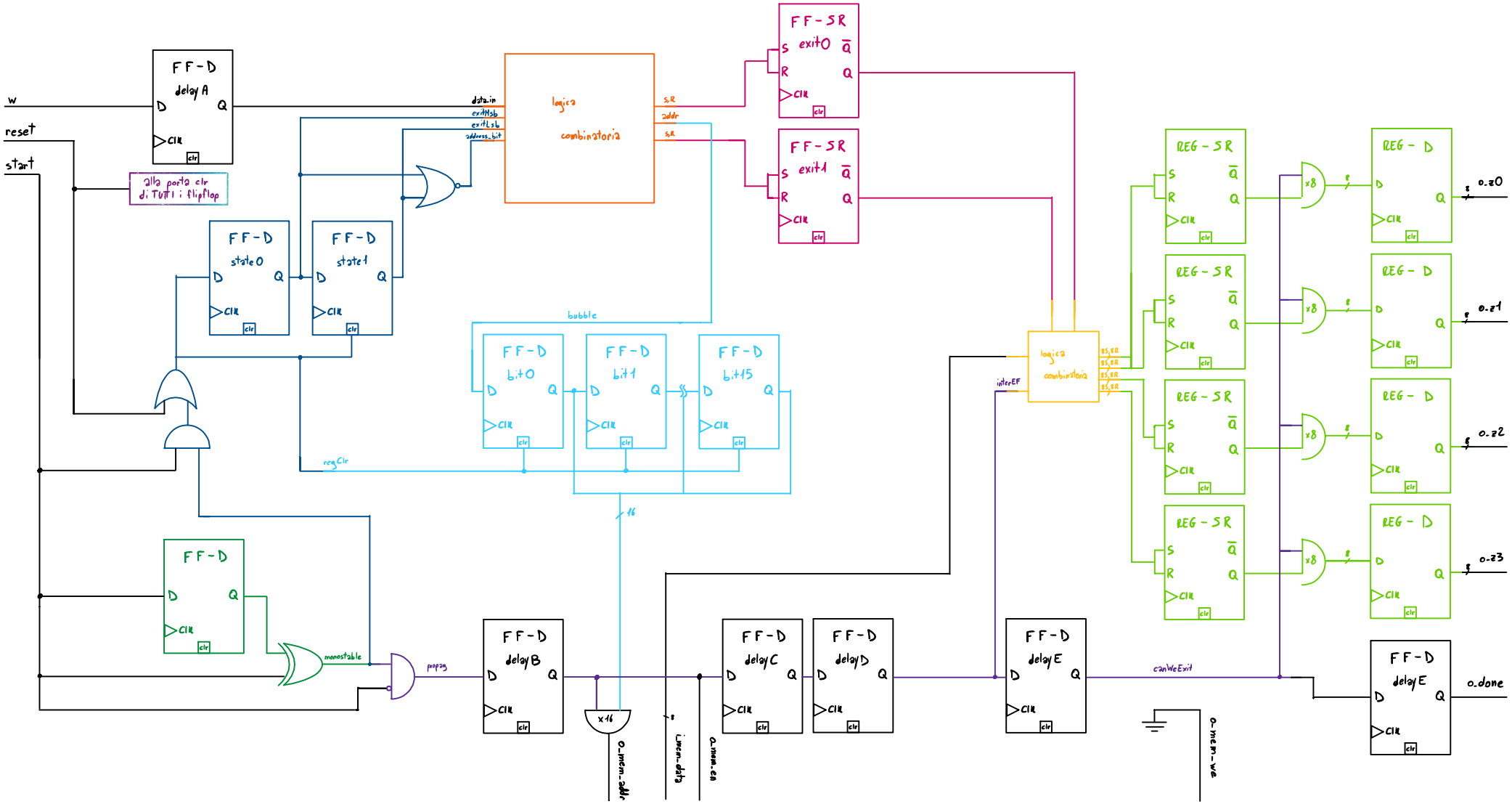
Conclusione

Il progetto presentato è correttamente sintetizzabile e supera tutti i test sottoposti (inclusi i testbench forniti come esempio) sia in behavioral simulation che in functional e timing post-synthesis simulation

Il progetto, inoltre, è stato sottoposto sia ai test sopra descritti che ad una serie di batteria di test (non riportati in questa relazione) generati casualmente da uno script python, sia leggeri che piuttosto pesanti, al fine di testarne ulteriormente la robustezza

Si vuole anche porre particolare attenzione al fatto che il tempo che intercorre tra il fronte di discesa del segnale `i_start` e il fronte di salita del segnale `o_done` non dipende dalla lunghezza dell'input dato bensì corrisponde esattamente a cinque cicli di clock, rendendo così più semplice una eventuale interazione tra il circuito e altri componenti

Infine, pur non essendo richiesto dalla specifica, il circuito è stato anche implementato e sottoposto alle relative simulazioni con successo per ogni testbench citato in questa relazione. Dimostra così che il progetto sarebbe in grado di funzionare correttamente su una fpga calcolando anche i ritardi su bus reali e non solamente ideali


w
reset
start
FF-D
delay A
Q
CLR
clr
alla porta clr di TUTTI i flipflop
data_in
exitMsb
exitLsb
address_bit
logica
combinatoria
s,R
addr
s,R
FF-SR
exit0
exit1
S
R
Q
FF-D
state0
state1
bubble
FF-D
bit0
bit1
bit15
reqClr
16
FF-D
delay B
FF-D
delay C
FF-D
delay D
FF-D
delay E
monostable
popRq
x16
o_mem_addr
i_mem_data
o_mem_en
o_mem_we
canWeExit
interEF
logica
combinatoria
REG-SR
REG-D
x8
o_z0
o_z1
o_z2
o_z3
o_done