PROGETTO DI RETI LOGICHE – Prova finale

Prof. Fabio Salice – Anno Accademico 2022 / 23

Gentile Mirko

Cod. persona 10681978

Matricola 961246

Indice

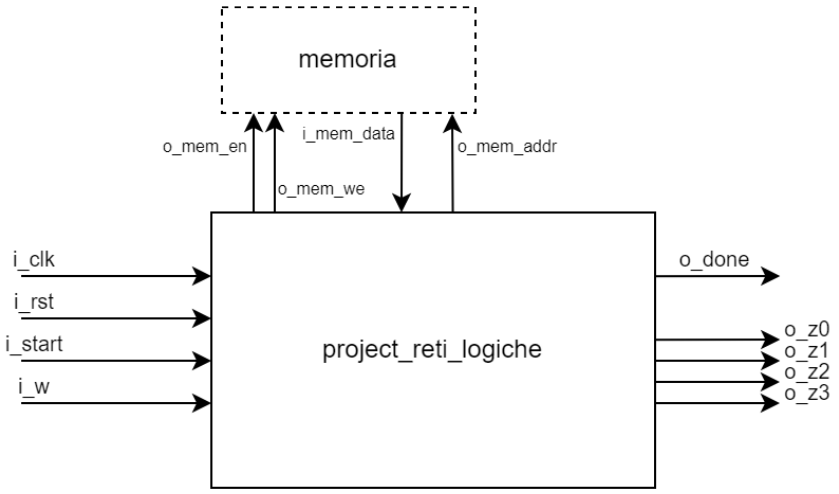
1. Introduzione

1.1 Spiegazione specifica di progetto

Obiettivo della prova finale di quest'anno era progettare, in linguaggio VHDL, una componente hardware che compiesse delle specifiche operazioni.

Più specificatamente, dati in ingresso i segnali di clock, start, reset, w(rite), compito dello studente è quello di acquisire il segnale seriale in ingresso, scomporlo in canale di uscita ed indirizzo, fornire tale indirizzo a memoria (che in questa prova è da considerarsi esterna alla componente), e riportare il segnale dati di ritorno da memoria sul canale selezionato.

Di seguito una raffigurazione della componente vista esternamente:

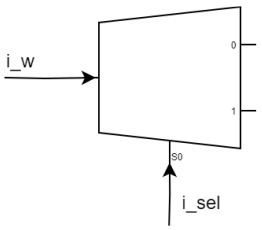


Il segnale di abilitazione scrittura (*o_mem_we*) non viene utilizzato poiché non sono necessarie operazioni di scrittura su disco.

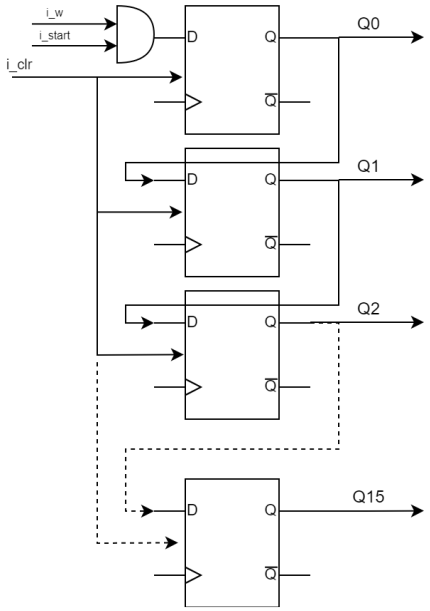
2. Architettura

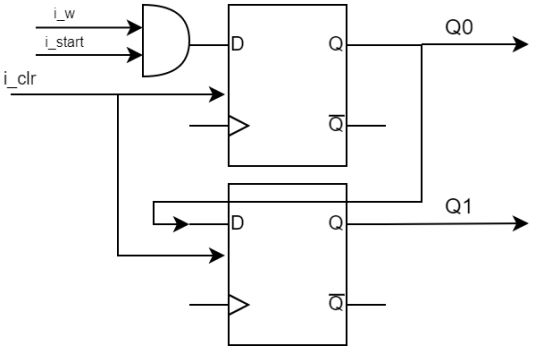
2.1 Schema funzionale

Di seguito analizziamo brevemente gli snodi principali alla base del datapath (possono non coincidere con la effettiva sintesi del circuito)



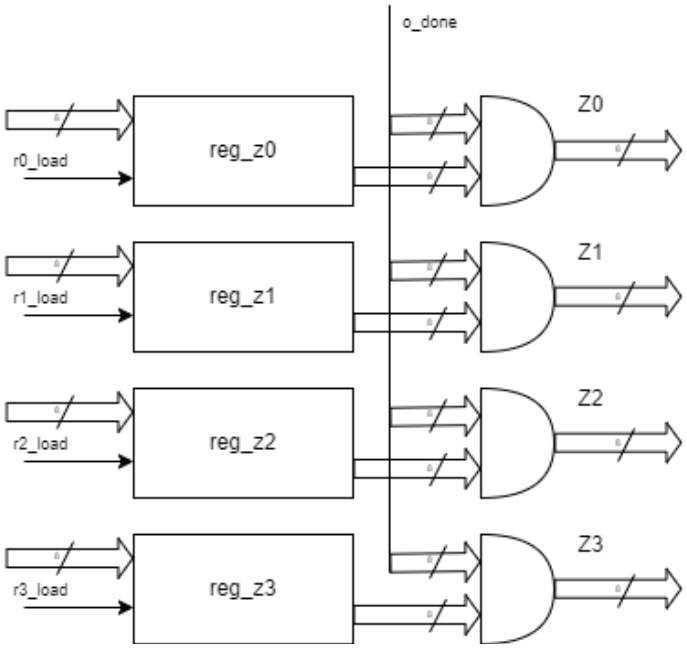
Il segnale di input viene smistato su 2 canali (‘0’ contenuto del messaggio, ‘1’ canale Zn di uscita). La pilotante *i_sel* viene fornita dalla FSM.





In uscita al demux troviamo i due shifter di tipo SIPO (serial input parallel out). I segnali di ingresso sono messi in AND logico con il segnale *i_start*, per campionare solo i bit effettivamente utili.

Un ennesimo segnale *i_clr* si occupa di ripulire i due shifter. Viene alzato in due casi: al reset, o simultaneamente a *o_done*.



Un ulteriore demux si occupa di indirizzare il dato ricevuto da memoria sul canale corretto. Ciascuno dei canali di output della componente presenta un registro in modo chè l'ultimo dato possa essere salvato. Le uscite dei registri sono messe in AND logico con il segnale *o_done*, replicato su 8bit, affinchè siano visibili solo quando quest'ultimo è alto. Questa parte, nel pratico, è stata codificata con un semplice if statement.

2.2 Macchina a stati

Di seguito una rapida descrizione di ogni stato. Il nome degli stati non è propriamente incrementale poichè alcuni di questi son stati rimossi durante la progettazione.

S0 (idle):

Stato iniziale della macchina, che resta in attesa di *i_start* alto. Raggiungibile tramite reset della macchina, o dopo la fine di una precedente trasmissione.

S2 (primo bit canale di output):

Stato di inizio trasmissione, il primo bit del canale di uscita viene acquisito in questo stato.

S3:

Il secondo bit del canale di uscita viene acquisito in questo stato. Si è scelto di non implementare un loop poiché il num. di bit da acquisire è prefissato, cioè 2.

S5 (primo bit di indirizzo, se presente):

Qui inizia la vera e propria acquisizione del messaggio, che ha dimensione variabile. Dunque si sceglie di implementare un loop, ad eccezione del primo bit, al fine di alzare il segnale *r1_load*.

S7 (acquisizione indirizzo):

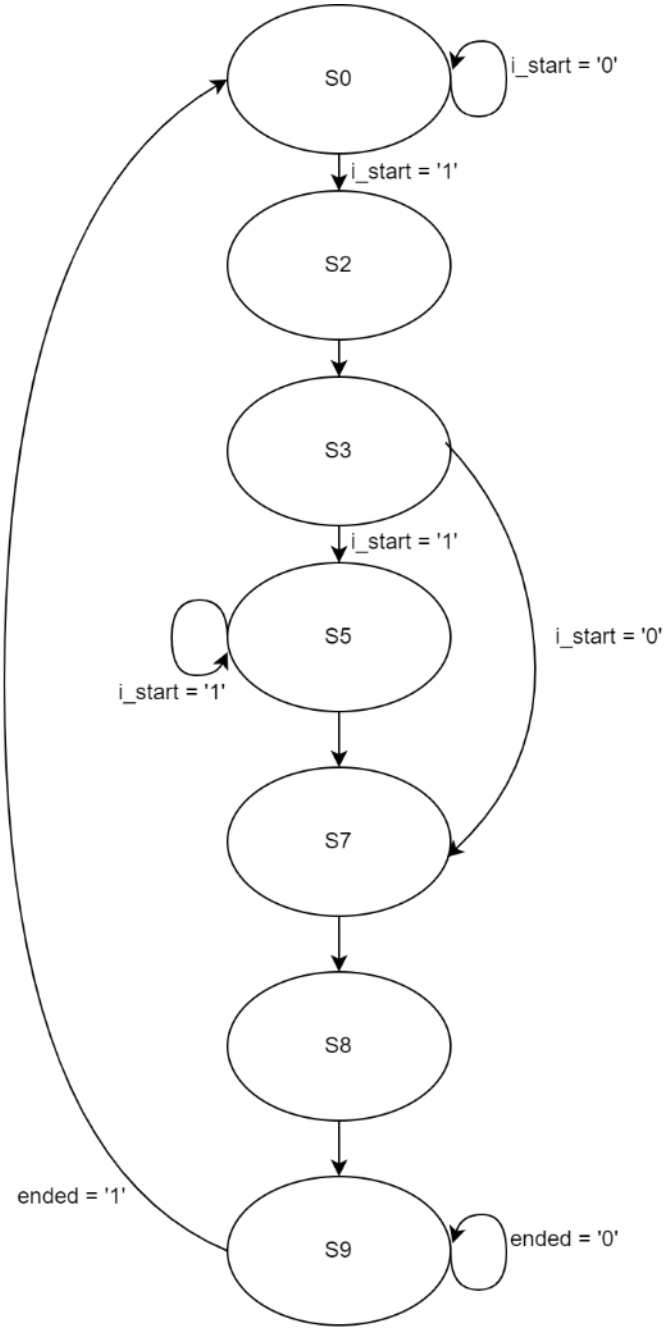
Loop di acquisizione messaggio.

S8 (busy waiting):

Questo stato serve a coordinarsi con il datapath. Alza il segnale *flag*, che verrà utilizzato da quest'ultimo.

S9 (ritorno):

Stato finale della macchina, che segna il termine della trasmissione. Quando il datapath fornisce il segnale *ended* alto, lo stato alza *o_done* per un ciclo, e ritorna allo stato iniziale.



2.3 Descrizione dei moduli

Si è diviso il codice del progetto in due moduli principali:

project_reti_logiche

```
entity project_reti_logiche is
    port (
        i_clk : in std_logic;
        i_rst: in std_logic;
        i_start : in std_logic;
        i_w : in std_logic;
        o_z0 : out std_logic_vector(7 downto 0);
        o_z1 : out std_logic_vector(7 downto 0);
        o_z2 : out std_logic_vector(7 downto 0);
        o_z3 : out std_logic_vector(7 downto 0);
        o_done : out std_logic;
        o_mem_addr : out std_logic_vector(15 downto 0);
        i_mem_data : in std_logic_vector(7 downto 0);
        o_mem_we : out std_logic;
        o_mem_en : out std_logic
    );
end project_reti_logiche;
```

Entità top-level, contiene la FSM. Tramite dei segnali, si occupa del coordinamento del datapath. Le porte in entrata/uscita sono fornite da specifica e non modificabili.

datapath

```
entity datapath is
    port ( i_clk:  in STD_LOGIC;
           i_str:  in STD_LOGIC;
           i_rst:  in STD_LOGIC;
           i_clr:  in STD_LOGIC;
           i_sel:  in STD_LOGIC;
           i_flag: in STD_LOGIC;
           i_w:    in STD_LOGIC;
           i_mem_data: in STD_LOGIC_VECTOR (7 downto 0);
           r0_load: in STD_LOGIC;
           r1_load: in STD_LOGIC;
           o_r0: out STD_LOGIC_VECTOR (15 downto 0);
           --r2_load: in STD_LOGIC;
           --r2_sel:  in STD_LOGIC;
           --o_end: out STD_LOGIC
           o_z0: out STD_LOGIC_VECTOR (7 downto 0);
           o_z1: out STD_LOGIC_VECTOR (7 downto 0);
           o_z2: out STD_LOGIC_VECTOR (7 downto 0);
           o_z3: out STD_LOGIC_VECTOR (7 downto 0);
           o_end: out STD_LOGIC
    );
end datapath;
```

Componente istanziata dal livello superiore, contiene tutta la parte di elaborazione, ovvero gli shifter, i registri, i demux. Gestisce inoltre i segnali da/a memoria.

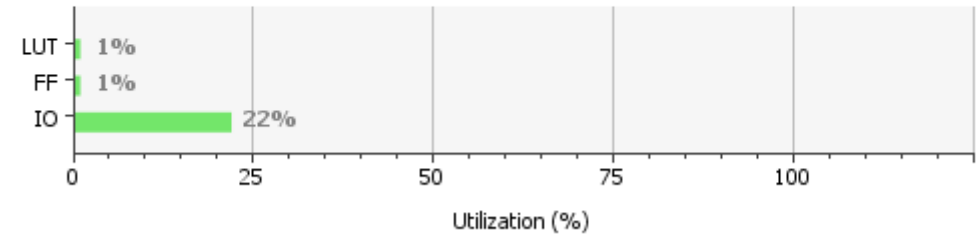
I segnali commentati fanno riferimento ad un registro r2 che non è stato utilizzato.

3. Risultati sperimentali

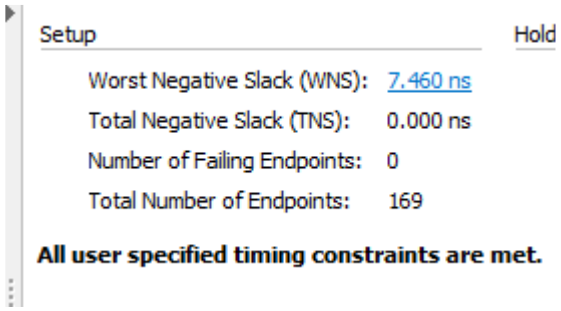
3.1 Report di sintesi

Vediamo raffigurato il grafico di utilizzazione post sintesi della componente. Nessun latch è stato utilizzato.

Resource	Utilization	Available	Utilization %
LUT	41	134600	0.03
FF	105	269200	0.04
IO	63	285	22.11

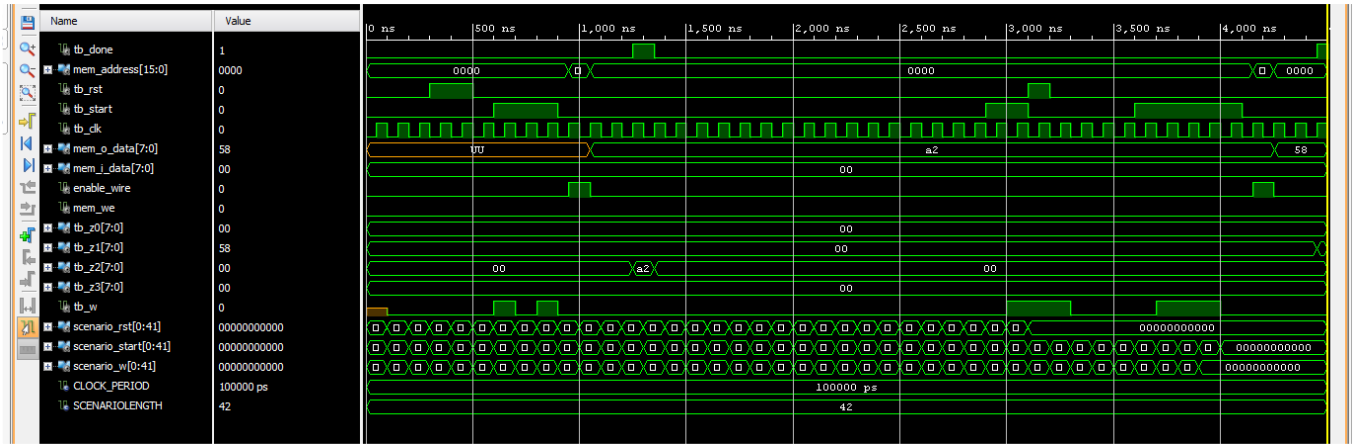


Qui invece leggiamo il tempo di scarto della componente sintetizzata su un costraint di 10ns.

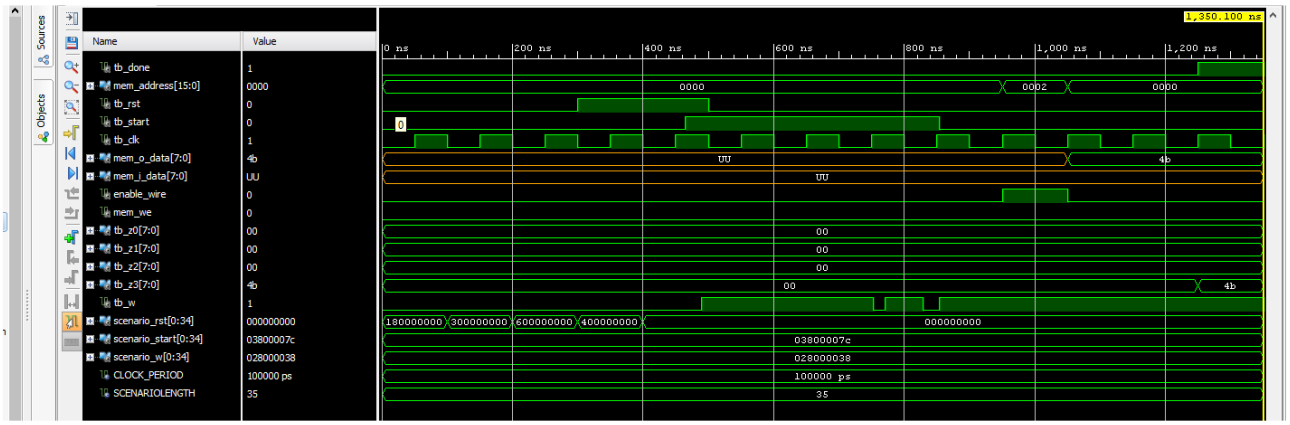


3.2 Simulazioni (test massivi e corner cases)

Oltre ai test benches forniti come esempio, ne sono stati generati ulteriori, al fine di analizzare il comportamento sotto stress e in specifici casi corner, e dei quali riporterò i piu significativi.



Il primo dei test benches proposti, era stato generato principalmente per assicurarsi del corretto funzionamento del reset. Mi ha tuttavia mostrato un errore che altrimenti non avrei notato, e che avrebbe invalidato la prova. Il caso in questione riguarda l'arrivo di bits su *i_w* con *i_start* in stato basso, che non era stato correttamente gestito.



Il test bench a seguire riguarda invece l arrivo asincrono di bit sul segnale *i_w*, correttamente gestito come si può notare dalle forme d'onda.

Tra gli altri testbenches sono, poi, stati verificati il caso di reset asincrono, di messaggio a dimensione nulla, massima e infine con tutti zeri.

4. Considerazioni finali

In conclusione, progettare questa componente è stata una sfida stimolante, inizialmente avvertita con ostilità a causa dei linguaggi e strumenti mai utilizzati in precedenza, ma diventata poi motivo di approfondimento e constatazione, una volta presa confidenza con il materiale di lavoro.

Imparare il linguaggio VHDL richiede di stravolgere la concezione classica di programmazione, per come siamo abituati a studiarla, cosa non semplice da fare senza un coinvolgimento attivo alla materia.

Poter poi mettere mano concretamente a tali dispositivi elettronici, senza limitarsi alla teoria sottostante, è una sensazione assai gratificante, nonchè estremamente utile a fini extra-accademici e lavorativi.