

Progetto di Reti Logiche

Prof. Fornaciari, Prof. Palermo e Prof. Salice
Anno Accademico 2025 - 2026

SPECIFICA per lo svolgimento del progetto Revisione del 24 Febbraio 2026

Descrizione generale

La specifica della “Prova Finale (Progetto di Reti Logiche)” per l'Anno Accademico 2025/2026 chiede di implementare un modulo HW (descritto in VHDL) che si interfacci con una memoria e che rispetti le indicazioni riportate nella seguente specifica.

Si richiede di progettare e implementare in VHDL un modulo hardware che gestisce una lista ordinata di task memorizzati in una memoria esterna. Ogni task è rappresentato da una coppia di campi: ID_TASK (6 bit) e PRIORITY (2 bit). Ad esempio, 00100111 indica il task con ID 001001 che ha priorità 11. Le priorità vanno da 0 (più alta) a 3 (più bassa). Non possono esistere due task con lo stesso ID_TASK (questo implica che non possono esistere più di 63 task nella lista). L'ID_TASK è sempre un numero positivo e 0 rappresenta una condizione di errore (lista vuota).

La lista dei task è memorizzata in una memoria con la seguente struttura:

- All'indirizzo 0 è memorizzato il numero di task attualmente presenti nella lista.
- A partire dall'indirizzo 1 sono memorizzati i task all'interno di un byte di memoria (ID_TASK & PRIORITY), che devono essere sempre mantenuti ordinati in base alla priorità (priorità più bassa numericamente = più alta gerarchicamente, e quindi prima nella lista). All'interno del byte, ID_TASK occupa i 6 bit più alti e PRIORITY i 2 bit più bassi. Ad esempio, 0x00001100 è il task con ID_TASK 3 e PRIORITY 0.

Il modulo deve supportare quattro operazioni, selezionate tramite un ingresso di controllo a 2 bit denominato OP:

- OP = 00: decrementa la priorità di tutti i task presenti nella lista. Ogni indicatore di priorità deve essere incrementato di 1, senza superare il valore numerico massimo (3 - priorità più bassa). Il valore di priorità satura a 3 (priorità minima). Tutti i task mantengono l'ordinamento che avevano prima dell'incremento del valore di priorità. In particolare, la lista dei task a priorità 3 vedrà prima tutti i task che avevano priorità 2 e poi quelli che già erano a priorità 3 (in pratica sarà sufficiente modificare i valori di priorità senza alcun riordino).
- OP = 01: Rimuove dalla lista il primo task (indirizzo 1). Tutti i task successivi devono essere spostati di una posizione, il numero di task deve essere decrementato, e il numero del task estratto deve essere fornito in uscita dal modulo. Il primo task della lista potrebbe avere una priorità qualunque (0, 1, 2 o 3). In caso di lista vuota, deve essere fornito in uscita il valore di ID_TASK 0 (0x000000). Il valore ID_TASK deve essere valido quando DONE viene portato a 1.

- OP = 10: Aggiunge un nuovo task alla lista, inserendolo in coda ai task a pari priorità, preservando l'ordine di tutti gli altri task. Se la lista è vuota, il task deve essere inserito in prima posizione.
- OP = 11: Svuota completamente la lista. Il valore in memoria all'indirizzo 0 deve essere posto a zero; il contenuto degli altri indirizzi può essere ignorato (quello che è stato scritto in passato rimane invariato) o resettato (tutti i valori sono posti a zero).

Il modulo utilizza un protocollo di hand-shake START-DONE per la sincronizzazione con la logica esterna. Il comportamento richiesto è il seguente:

- Il segnale START, generato dall'esterno, viene posto a 1 per avviare l'operazione selezionata tramite OP.
- START deve rimanere a 1 fino a quando il modulo non porta il segnale DONE a 1, indicando che l'operazione è stata completata.
- Quando DONE è a 1, la logica esterna deve riportare START a 0.
- Quando START torna a 0, il modulo può riportare DONE a 0 e prepararsi a ricevere una nuova operazione.
- Una nuova operazione può essere avviata solo quando DONE è a 0.

Dopo che il modulo viene resettato (quindi subito dopo la transizione RESET 1 -> 0), esso deve comportarsi come segue:

- Dopo un reset, la lista deve essere considerata vuota. Il modulo deve quindi scrivere il valore zero all'indirizzo 0 della memoria.
- Durante questa fase di inizializzazione, il segnale DONE deve essere mantenuto a 1, indicando che il modulo non è ancora pronto a ricevere operazioni, e solo dopo che l'azzeramento è stato effettuato, il modulo può riportare DONE a 0 e rendersi disponibile a nuove operazioni.

Il modulo deve garantire:

- Il corretto aggiornamento della memoria in seguito a ogni operazione.
- Il mantenimento dell'ordine dei task nella lista.
- La gestione appropriata dei casi limite, ad esempio lista vuota.
- Il rispetto del protocollo START-DONE come descritto.

Si precisa che la presente specifica descrive un comportamento completamente deterministico: a parità di sequenze di ingresso, le uscite generate e il contenuto della memoria nella parte valida (lista dei task) risultano invariati. Qualsiasi eventuale scenario che presenti ambiguità o comportamenti non univocamente determinati dovrà essere identificato e prontamente segnalato.

Interfaccia del Componente

Il modulo da implementare ha 4 ingressi primari, uno ad 1 bit (*i_start*), uno a 6 bit (*i_task_id*), uno a 2 bit (*i_task_priority*) e uno da 2 bit (*i_op*), e due uscite primarie, una da 1 bit (*o_done*) e una da 6 bit (*o_task_id*). Inoltre, il modulo ha un segnale di clock CLK, unico per tutto il sistema e un segnale di reset RESET anch'esso unico per tutto il sistema. Tutti i segnali sono sincroni e devono essere interpretati sul fronte di salita del clock. L'unica eccezione è RESET che, invece, è asincrono. RESET può essere generato in qualsiasi momento dell'esecuzione.

Il componente da descrivere deve avere la seguente interfaccia.

```
entity project_reti_logiche is
    port (
        i_clk      : in std_logic;
        i_rst      : in std_logic;

        i_start     : in std_logic;
        i_task_id   : in std_logic_vector(5 downto 0);
        i_task_priority : in std_logic_vector(1 downto 0);
        i_op        : in std_logic_vector(1 downto 0);

        o_done      : out std_logic;
        o_task_id   : out std_logic_vector(5 downto 0);

        o_mem_addr  : out std_logic_vector(15 downto 0);
        i_mem_data  : in std_logic_vector(7 downto 0);
        o_mem_data  : out std_logic_vector(7 downto 0);
        o_mem_we    : out std_logic;
        o_mem_en    : out std_logic
    );
end project_reti_logiche;
```

In particolare:

- il nome del modulo deve essere `project_reti_logiche` e deve essere presente una sola architettura per ogni entità; la violazione di queste indicazioni comporta l'impossibilità di eseguire il Test Bench e una conseguente valutazione di zero;
- *i_clk* è il segnale di CLOCK in ingresso generato dal Test Bench;
- *i_rst* è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- *i_start* è il segnale di START generato dal Test Bench;
- *i_task_id* è il vettore di bit rappresentante il ID_TASK generato dal Test Bench;

- i_task_priority è il vettore di bit rappresentante il PRIORITY generato dal Test Bench;
- i_op è il vettore di 2 bit rappresentante l'operazione da dover effettuare sulla lista di task;
- o_done è il segnale DONE di uscita che comunica la fine dell'elaborazione;
- o_task_id è il vettore di bit rappresentante il ID_TASK del task a priorità più alta estratto dalla memoria;
- o_mem_addr è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- i_mem_data è il segnale (vettore) che arriva dalla memoria e contiene il dato in seguito ad una richiesta di lettura;
- o_mem_data è il segnale (vettore) che va verso la memoria e contiene il dato che verrà successivamente scritto;
- o_mem_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_mem_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria, esso deve essere 0.

APPENDICE: Descrizione Memoria

NOTA: La memoria è già istanziata all'interno del Test Bench e non va sintetizzata

La memoria e il suo protocollo può essere estratto dalla seguente descrizione VHDL che fa parte del test bench e che è derivata dalla User guide di VIVADO disponibile al seguente link:

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado-synthesis.pdf

```
-- Single-Port Block RAM Write-First Mode (recommended template)
--
-- File: rams_02.vhd
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rams_sp_wf is
port(
    clk  : in  std_logic;
    we   : in  std_logic;
    en   : in  std_logic;
    addr : in  std_logic_vector(15 downto 0);
    di   : in  std_logic_vector(7 downto 0);
    do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then
                    RAM(conv_integer(addr)) <= di;
                    do <= di after 2 ns;
                else
                    do <= RAM(conv_integer(addr)) after 2 ns;
                end if;
            end if;
        end if;
    end process;
end syn;
```

ESEMPIO

L'esempio qui di seguito mostra il comportamento a seguito dei segnali di ingresso. Qui di seguito è presente la situazione della memoria a seguito dell'inserimento di diversi task (fase non mostrata nell'esempio). I valori non espliciti sono valori di memoria che non vengono considerati per il normale funzionamento.

SITUAZIONE INIZIALE (6 task già in tabella)

MEMORIA			
INDIRIZZO	VALORE	ID_TASK	PRIORITY
0	0x00000110		
1	0x01000000	16	0
2	0x01011000	22	0
3	0x01011100	23	0
4	0x01010101	21	1
5	0x01110010	28	2
6	0x00010011	4	3
7			

OPERAZIONE: 00 (incremento valore di priorità)

MEMORIA			
INDIRIZZO	VALORE	ID_TASK	PRIORITY
0	0x00000110		
1	0x01000001	16	1
2	0x01011001	22	1
3	0x01011101	23	1
4	0x01010110	21	2
5	0x01110011	28	3
6	0x00010011	4	3
7			

OPERAZIONE: 10 (aggiunge un task) - ID_TASK: 19 - PRIORITY: 2

MEMORIA			
INDIRIZZO	VALORE	ID_TASK	PRIORITY
0	0x00000111		
1	0x01000001	16	1
2	0x01011001	22	1
3	0x01011101	23	1
4	0x01010110	21	2
5	0x01001110	19	2
6	0x01110011	28	3
7	0x00010011	4	3
8			

OPERAZIONE: 01 (rimuove il primo task)

MEMORIA			
INDIRIZZO	VALORE	ID_TASK	PRIORITY
0	0x00000110		
1	0x01011001	22	1
2	0x01011101	23	1
3	0x01010110	21	2
4	0x01001110	19	2
5	0x01110011	28	3
6	0x00010011	4	3
7			

OPERAZIONE: 11 (svuota la lista)

MEMORIA	
INDIRIZZO	VALORE
0	0x00000000
1	

ID_TASK PRIORITY

OPERAZIONE: 10 (aggiunge un task) - ID_TASK: 31 - PRIORITY: 3

MEMORIA	
INDIRIZZO	VALORE
0	0x00000001
1	0x01111111
2	

ID_TASK PRIORITY

OPERAZIONE: 10 (aggiunge un task) - ID_TASK: 15 - PRIORITY: 0

MEMORIA	
INDIRIZZO	VALORE
0	0x00000010
1	0x00111100
2	0x01111111
3	

ID_TASK PRIORITY

15 0

31 3

OPERAZIONE: 10 (aggiunge un task) - ID_TASK: 20 - PRIORITY: 0

MEMORIA	
INDIRIZZO	VALORE
0	0x00000011
1	0x00111100
2	0x01010000
3	0x01111111

ID_TASK PRIORITY

15 0

20 0

31 3

Progetto di Reti Logiche

Prof. Fornaciari, Prof. Palermo e Prof. Salice
Anno Accademico 2025 - 2026

NOTE DI AGGIORNAMENTO DELLA SPECIFICA

In questa pagina potete trovare le modifiche fatte alla specifica del progetto dal suo primo rilascio. Tutti i cambiamenti con data annessa saranno riportati qui sotto e sono mantenuti in rosso nel testo.

Errata Corrigere:

- Aggiornamento 24.02.2026:
 - Il numero massimo di task nella lista non è 31 come nella versione originale ma 63. Questo è derivato dal fatto che il numero di bit per il task_id è 6.