# Contents

# 1 Esercizio 1 - Multiplexer

L'esercizio 1.1 richiede la rappresentazione di un multiplexer 16:1 tramite la composizione di multiplexer 4:1, quindi definiamo il module del componente mux_4_1 definito in modo Dataflow, e poi per le proprietà della modularità definiamo il module mux_16_1 come composizione dei precedenti, tramite il costrutto *for..generate*.

Il multiplexer 4:1 ha come ingressi quattro bit, identificati col vettore a(0 to 3), e $\lceil \log_2(n) \rceil$ segnali di abilitazione, dove n è il numero di segnali di ingresso. In questo caso, quindi, due segnali di abilitazione ed uno di uscita.

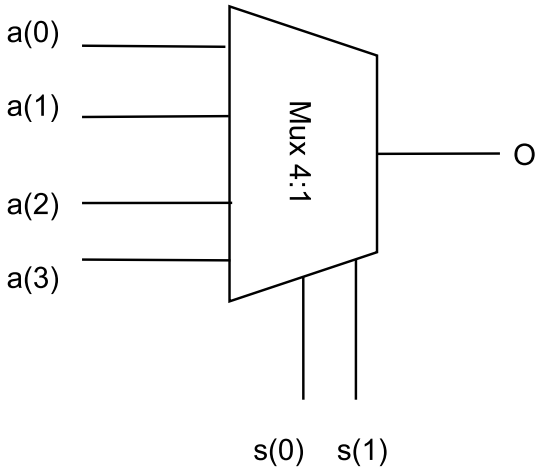


Figure 1: Mux 4:1

Il codice in VHDL per descrivere il comportamento di questo componente è il seguente:

```
architecture DataFlow of mux_4_1 is

begin
    o <= a(0) when s = "00" else
         a(1) when s = "01" else
         a(2) when s = "10" else
         a(3) when s = "11" else
         '-';

end DataFlow;
```

Figure 2: Mux 4:1 Dataflow

Composto da un costrutto *when...else* che suddivide i diversi casi e gestisce anche tutti i casi non definiti con l'ultima clausola else senza alcuna condizione.

Definito l'elemento base del progetto, si passa a comporre il multiplexer 16:1 tramite un approccio strutturale, nel quale generiamo cinque mux_4_1: i primi quattro avranno gli ingressi interfacciati con l'esterno e, tramite segnali interni, le loro uscite sono collegate come ingressi dell'ultimo multiplexer che

costituisce l'uscita del sistema. La macchina completa presenta quindi 16 segnali di ingresso, 4 segnali di selezione ed un unico segnale di uscita:

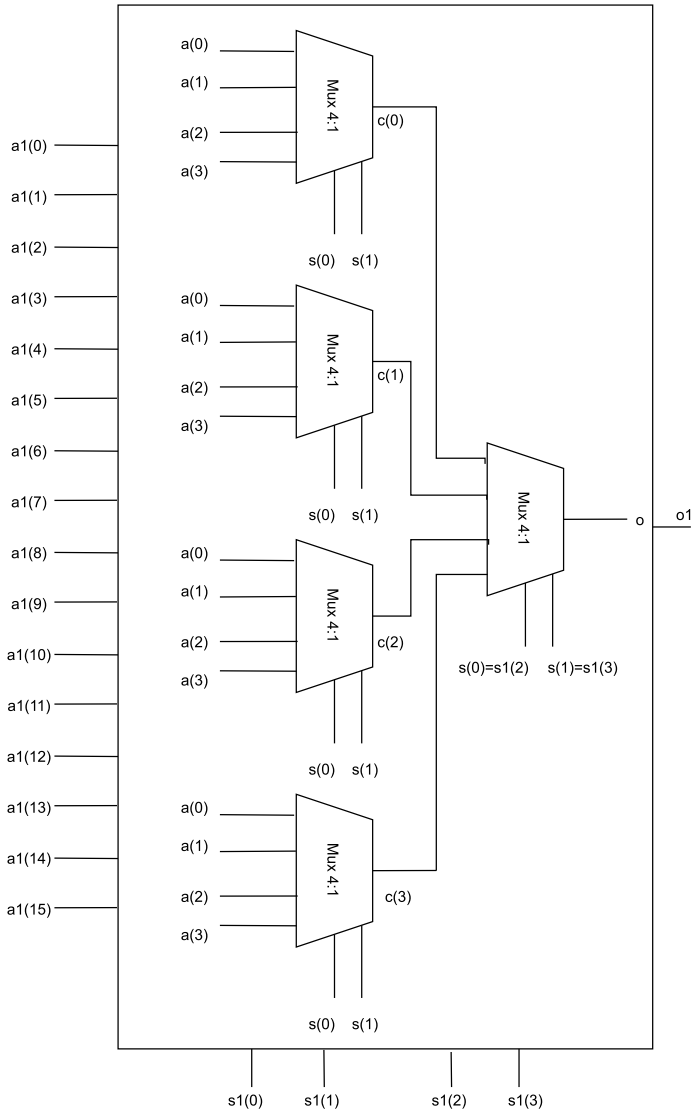


Figure 3: Mux 16:1

Lo schema sopra rappresentato è tradotto in linguaggio VHDL dal seguente codice, dove l'entità mux_16_1 è rappresentata con approccio strutturale e, tramite i segnali interni c(0 to 3), colleghiamo le uscite parziali dei primi quattro mux_4_1, identificati con la label mux_4_1_in, con i quattro ingressi dell'ultimo mux_4_1, identificato con la label mux_4_1_fin.

```
entity mux_16_1 is
    Port ( a1 : in STD_LOGIC_VECTOR (0 to 15);
           s1 : in STD_LOGIC_VECTOR (0 to 3);
           o1 : out STD_LOGIC);
end mux_16_1;

architecture Structural of mux_16_1 is
    COMPONENT mux_4_1 PORT (a : in STD_LOGIC_VECTOR (0 to 3); s : in STD_LOGIC_VECTOR (0 to 1); o : out STD_LOGIC);
    END COMPONENT;
    FOR ALL: mux_4_1 USE ENTITY WORK.mux_4_1 (Behavioral);
    signal c : STD_LOGIC_VECTOR (0 to 3);
begin
    mux_4_1_in : FOR i in 0 to 3 GENERATE m: mux_4_1
        port map (
        a(0 to 3) => a1(i*4 to i*4+3),
        s(0 to 1) => s1(0 to 1),
        o => c(i)
        );
    end GENERATE;

    mux_4_1_fin : mux_4_1
        port map(
        a(0 to 3) => c(0 to 3),
        s(0 to 1) => s1(2 to 3),
        o => o1
    );
end Structural;
```

Figure 4: Mux 16:1 Dataflow

Progettato il mux_16_1, è possibile testarlo attraverso un testbench. La prima cosa che bisogna specificare è che il corpo dell'entity è vuoto, questo perché non si tratta di oggetto che realizziamo, ma serve solo per effettuare la simulazione e verificare se il sistema realizzato funziona correttamente. Il testbench effettivamente non ha né segnali d'ingresso né d'uscita, ma sfrutta per i test i segnali interni definiti nel codice. Per testare il mux_16_1 definito precedentemente, abbiamo istanziato una uut (Unit Under Test) in cui colleghiamo le varie porte ai segnali (input, selection, output).

# 2 Esercizio 2 - Encoder BCD

# 3 Esercizio 3 - Riconoscitore di Sequenze

# 4 Esercizio 4 - Shift Register

# 5 Esercizio 5 - Cronometro

# 6 Esercizio 6 - Sistema di Testing

# 7 Esercizio 7 - Comunicazione con Handshaking

# 8 Esercizio 8 - Processor

# 9 Esercizio 9 - Interfaccia UART

# 10 Esercizio 10 - Switch Multistadio

# 11 Esercizio 11 - Divisore Restoring

# 12 Esercizio 12 - Interfaccia VGA