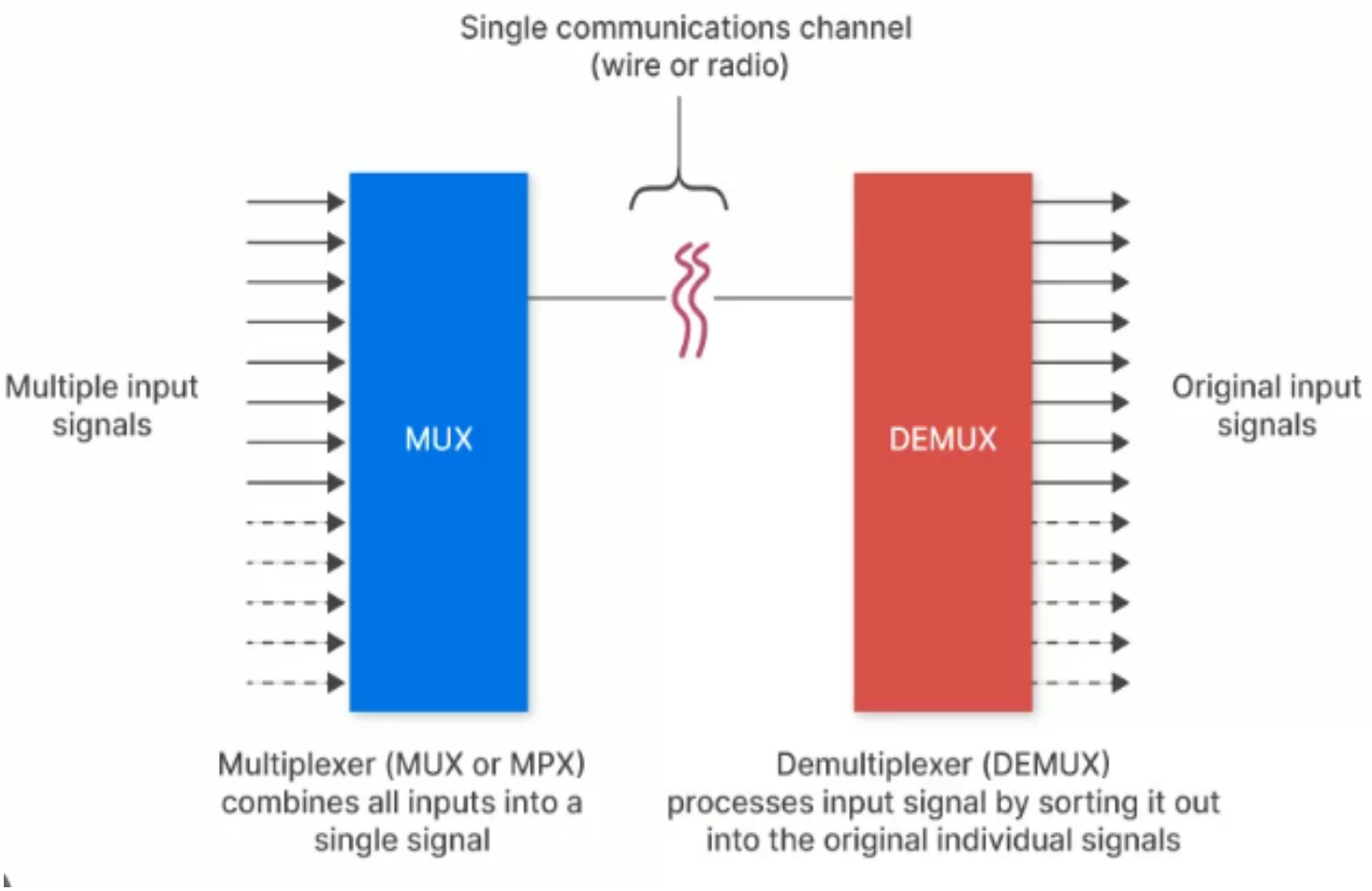
LAB 3: Parameterized Multiplexer And Demultiplexer

Date: 11 de mayo de 2025



Proffesor: Dr. Francisco Javier Rodríguez Navarrete

Students: Cesar Eduardo Inda Ceniceros

Alonso Emmanuel López Macías

Team: 14

Introduction

In digital systems, multiplexers and demultiplexers are fundamental components for managing and redirecting data. A multiplexer (MUX) allows the selection of one among multiple input signals to be routed to a single output, while a demultiplexer (DEMUX) performs the inverse operation: it directs a single input signal to one of several possible outputs. The selection is made through control signals that determine which line is active at any given time.

The main objective of this lab is the design and implementation of a parameterized multiplexer and demultiplexer using the Verilog hardware description language. Parameterization enables the modules to be reusable and adaptable to different data widths (WIDTH) and numbers of channels (CHANNELS), promoting a more flexible and scalable design. Additionally, constructs such as two-dimensional arrays, generate blocks, and procedural descriptions are used to achieve this parameterization. The purpose is to strengthen understanding of buses, conditional logic, and generative structures in Verilog, applying these concepts both in simulation and in physical implementation on an FPGA. Tests are conducted with various parameter combinations, and the results are validated through testbenches and direct observation of the hardware behavior.

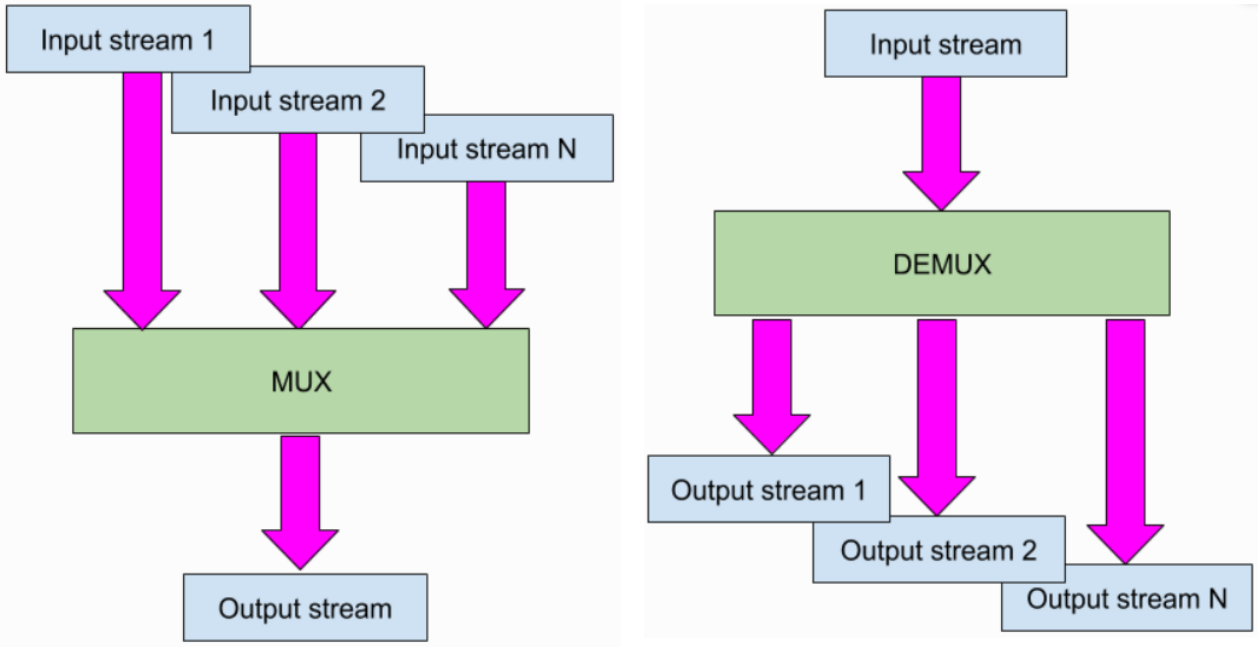


Figure: 1 Mux And Demux

Requirements

Step1.- With the base of multiplexers and demultiplexers we studied in class, define a way for the multiplexer/demultiplexer to work. Parameterization will be the following:

- *Parameterization of the width of the data input for each of the channels.*
- *Parameterization of the number of channel inputs.*
- *Parameterization of the input and output buses and any internal circuitry as needed.*

We will define two parameters called WIDTH and CHANNELS that will be used.

Step2.- Create a bus that can be indexed based on these two parameters, for example for a multiplexer we can have the following:

The bus will be connected to the inputs and then a selector logic will select the right slice of data to output. For the demultiplexer, something similar will be needed but the other way around. This is a way to make a parametrized logic with 2-D packed arrays Another way would be to create instances of muxes as needed using the generate-for or unrolled logic or also use generate-if to select one of the 3 configuration and instances. Get creative and use the power of Verilog to achieve this goal.

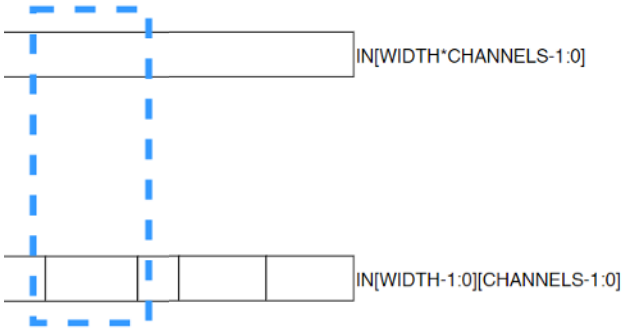


Figure: 2 Configuration

Step3.- Create a testbench to verify that the multiplexer and demultiplexer work. Test it with the following parameterized configurations:

- 32-bit WIDTH, 4 CHANNELS
- 1-bit WIDTH, 8 CHANNELS
- 18-bit WIDTH, 16 CHANNELS

Remember, each of these settings has a different parameterization and a different synthesized circuit. For this, a testbench that verifies all 3 types is mandatory since the 32-bit, 4 CHANNEL, and 18-bit, 16 CHANNELS won't fit in the I/O of the FPGA.

Step4.- Implement the circuit on the FPGA with 8 CHANNELS and a data WIDTH of 1.

For the multiplexer:

- Use the following switches for input SEL: SW [2] to SW [0]
- Use the following switches for each 1-bit input channel up to 8 channels: SW [17] to SW [10]
- Use the following green leds for output signal: LEDG [0]

In the LEDG light we should be able to see which of the selection inputs of the switches we are selecting via the SEL.

For the demultiplexer:

• Use the following switches for input SEL: SW [2] to SW [0]
• Use the following switches for the 1-bit input: SW [17]
• Use the following green leds for each of the 8-channel output signal: LEDG [7] to LEDG [0]

Development

For the design specifications, we also considered the following:

Parameters:

- **WIDTH**: Width of each channel (number of bits per channel)
- **CHANNELS**: Number of channels

For the purpose of this practice, the first step was to create the logic of the “multiplexer” module, in this case called “mux”, which made it possible to begin the implementation. The first step was to create the main logic of the practice, is the main entity.

NOTE: For the project the reference to “inputs” en some cases are “wires” by default because the tool assigns this value in automatic only in the cases when use a instance or other considerations we need to define a “wire” or “reg” situation.

The first logic for ***“mux. v”:***

```
module mux #(
    parameter WIDTH = 1,
    parameter CHANNELS = 4
)(
    input  wire [WIDTH*CHANNELS-1:0] in,     //Aqui se concatenan las entradas
    input  wire [$clog2(CHANNELS)-1:0] sel,
    output wire [WIDTH-1:0] out
);

    assign out = in[sel*WIDTH +: WIDTH];

endmodule
```

Figure: 3 mux. v

The second logic consideration was for ***“demux. v”:***

```
module demux #(
    parameter WIDTH = 1,
    parameter CHANNELS = 4
)(
    input  wire [WIDTH-1:0] in,
    input  wire [$clog2(CHANNELS)-1:0] sel,
    output reg  [WIDTH*CHANNELS-1:0] out
);

    integer i;

    always @(*) begin
        out = 0;
        for (i = 0; i < CHANNELS; i = i + 1) begin
            if (i == sel)
                out[i*WIDTH +: WIDTH] = in;
        end
    end

endmodule
```

Figure: 4 demux. V

The third part was the consideration for each one “top” in this case we necessary to define the top for the mux, ***“top_mux_fpga.v.”,*** this file represent the debug in the fpga for see the “mux” implementation.

```
module top_mux_fpga(
    input  wire [17:0] SW,
    output wire [7:0] LEDG
);

    wire [7:0] in_mux = SW[17:10];
    wire [2:0] sel = SW[2:0];
    wire out;

    mux #(.WIDTH(1), .CHANNELS(8)) uut (
        .in(in_mux),
        .sel(sel),
        .out(out)
    );

    assign LEDG[0] = out;
    assign LEDG[7:1] = 7'b0;

endmodule
```

Figure: 5 top_mux_fpga.v.

The four part was the consideration for each one “top” in this case we necessary to define the top for the demux, ***“top_demux_fpga.v.”,*** this file represents the debug in the fpga for see the “demux” implementation.

```
module top_demux_fpga(
    input  wire [17:0] SW,
    output wire [7:0] LEDG
);

    wire [2:0] sel = SW[2:0];
    wire in_demux = SW[17];
    wire [7:0] out;

    demux #(.WIDTH(1), .CHANNELS(8)) uut (
        .in(in_demux),
        .sel(sel),
        .out(out)
    );

    assign LEDG = out;

endmodule
```

Figure: 6 top_demux_fpga.v.

Testbench

To carry out this testbench, two files were required: one to be used as the top module for the testbench, and the other to include the 'tb' file with the necessary instances for the Model Sim Altera process. In this case, we will describe the code and how each of these elements was tested.

“tb_mux_demux”.

```
`timescale 1ns/1ps

module tb_mux_demux;

    // Parámetros para prueba
    parameter WIDTH = 32;
    parameter CHANNELS = 4;
    localparam SEL_WIDTH = $clog2(CHANNELS);

    reg [WIDTH*CHANNELS-1:0] mux_in;
    reg [SEL_WIDTH-1:0] sel;
    wire [WIDTH-1:0] mux_out;

    reg [WIDTH-1:0] demux_in;
    wire [WIDTH*CHANNELS-1:0] demux_out;

    // Instancias
    mux #(WIDTH, CHANNELS) uut_mux (
        .in(mux_in),
        .sel(sel),
        .out(mux_out)
    );

    demux #(WIDTH, CHANNELS) uut_demux (
        .in(demux_in),
        .sel(sel),
        .out(demux_out)
    );

    integer i;
    initial begin
        // Prueba de todos los canales
        mux_in = 0;
        for (i = 0; i < CHANNELS; i = i + 1) begin
            mux_in[i*WIDTH +: WIDTH] = i + 1;
        end

        for (i = 0; i < CHANNELS; i = i + 1) begin
            sel = i;
            #10;
            $display("MUX SEL=%0d -> OUT=%h", sel, mux_out);
        end

        // Prueba del demux
        for (i = 0; i < CHANNELS; i = i + 1) begin
            sel = i;
            demux_in = 32'hA5A5A5A5;
            #10;
            $display("DEMUX SEL=%0d -> OUT=%h", sel, demux_out);
        end

        $finish;
    end

endmodule
```

Figure: 7 tb_mux_demux

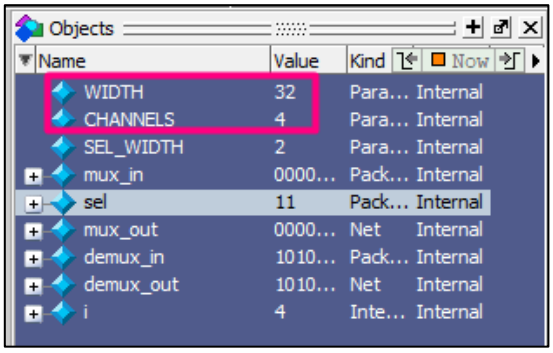
For test the main logic in combination with the “testbench” this need to be a main entity.

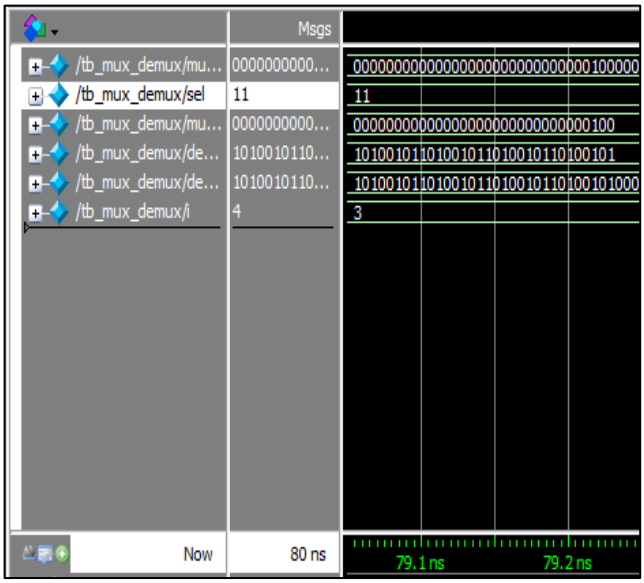
```
/*Este modulo es para referenciarlo al "testbench" - "tb_mux_demux" y poder trabajar en conjunto la simulación
en Model Sim.*/

module top_mux_demux #(
    parameter WIDTH = 32,
    parameter CHANNELS = 4

)(
    input wire [WIDTH*CHANNELS-1:0] mux_in,
    input wire [WIDTH-1:0] demux_in,
    input wire [$clog2(CHANNELS)-1:0] sel,
    output wire [WIDTH-1:0] mux_out,
    output wire [WIDTH*CHANNELS-1:0] demux_out

);


    mux #(WIDTH, CHANNELS) u_mux (
        .in(mux_in),
        .sel(sel),
        .out(mux_out)
    );


    demux #(WIDTH, CHANNELS) u_demux (
        .in(demux_in),
        .sel(sel),
        .out(demux_out)
    );

endmodule
```

Figure: 8 top_mux_demux

The test bench trials consisted of verifying the requirements specified.

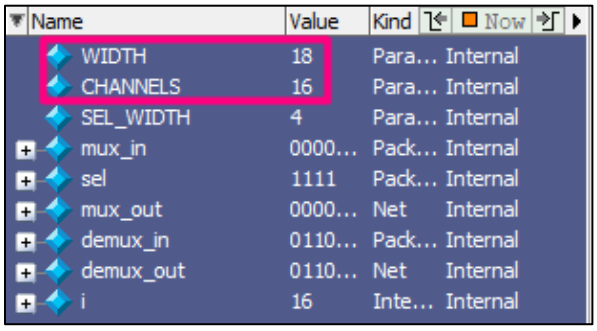
1.- WIDTH = 32 and CHANNELS = 4;

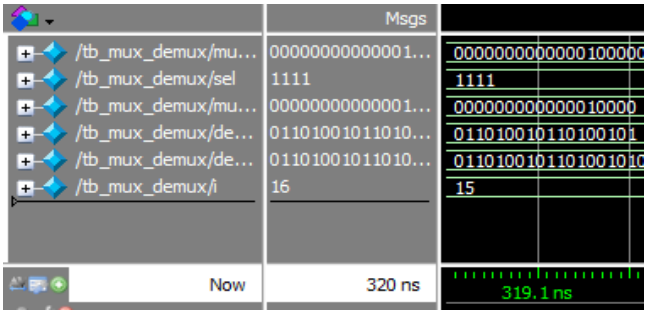





```
# MUX SEL=0 -> OUT=00000001
# MUX SEL=1 -> OUT=00000002
# MUX SEL=2 -> OUT=00000003
# MUX SEL=3 -> OUT=00000004
# DEMUX SEL=0 -> OUT=0000000000000000000000a5a5a5a5
# DEMUX SEL=1 -> OUT=00000000000000000a5a5a5a500000000
# DEMUX SEL=2 -> OUT=00000000a5a5a5a50000000000000000
# DEMUX SEL=3 -> OUT=a5a5a5a5000000000000000000000000
# ** Note: $finish    : D:/Verilog/Lab3/tb_mux_demux.v(71)
```

1.- WIDTH = 18 and CHANNELS = 16;






```
MUX SEL=0 -> OUT=00001
MUX SEL=1 -> OUT=00002
MUX SEL=2 -> OUT=00003
MUX SEL=3 -> OUT=00004
MUX SEL=4 -> OUT=00005
MUX SEL=5 -> OUT=00006
MUX SEL=6 -> OUT=00007
MUX SEL=7 -> OUT=00008
MUX SEL=8 -> OUT=00009
MUX SEL=9 -> OUT=0000a
MUX SEL=10 -> OUT=0000b
MUX SEL=11 -> OUT=0000c
MUX SEL=12 -> OUT=0000d
MUX SEL=13 -> OUT=0000e
MUX SEL=14 -> OUT=0000f
MUX SEL=15 -> OUT=00010
DEMUX SEL=0 -> OUT=00000000000000000000000000000000000000000000000000000000000000000000000000001a5a5
DEMUX SEL=1 -> OUT=00000000000000000000000000000000000000000000000000000000000000000000000696940000
DEMUX SEL=2 -> OUT=00000000000000000000000000000000000000000000000000000000000000001a5a5000000000
DEMUX SEL=3 -> OUT=00000000000000000000000000000000000000000000000000000000696940000000000000000
DEMUX SEL=4 -> OUT=00000000000000000000000000000000000000000000000001a5a5000000000000000000000000
DEMUX SEL=5 -> OUT=00000000000000000000000000000000000000000000000696940000000000000000000000000
DEMUX SEL=6 -> OUT=00000000000000000000000000000000000000000001a5a5000000000000000000000000000000
DEMUX SEL=7 -> OUT=00000000000000000000000000000000000000696940000000000000000000000000000000000
DEMUX SEL=8 -> OUT=00000000000000000000000000000001a5a5000000000000000000000000000000000000000000
DEMUX SEL=9 -> OUT=00000000000000000000000000696940000000000000000000000000000000000000000000000
DEMUX SEL=10 -> OUT=0000000000000000000001a5a5000000000000000000000000000000000000000000000000000
DEMUX SEL=11 -> OUT=00000000000000000696940000000000000000000000000000000000000000000000000000000
DEMUX SEL=12 -> OUT=000000000000001a5a5000000000000000000000000000000000000000000000000000000000
DEMUX SEL=13 -> OUT=00000000696940000000000000000000000000000000000000000000000000000000000000000
DEMUX SEL=14 -> OUT=0000001a5a5000000000000000000000000000000000000000000000000000000000000000000
DEMUX SEL=15 -> OUT=69694000000000000000000000000000000000000000000000000000000000000000000000000
** Note: $finish    : D:/Verilog/Lab3/tb_mux_demux.v(71)
   Time: 320 ns  Iteration: 0  Instance: /tb_mux_demux
```

FPGA Implementation

Once the program was completed, the compilation process was carried out in order to generate the pin assignment plan and correctly map the pins according to the program's requirements.

Flow Summary	
<<Filter>>	
Flow Status	In progress - Fri May 16 15:41:03 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	lab3_muxdemux
Top-level Entity Name	top_mux_demux
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	192
Total registers	0
Total pins	322
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure: 9 Compilation Brief

The pins were assigned

Node Name	Direction	Location
LEDG[0]	Unknown	PIN_E21
LEDG[1]	Unknown	PIN_E22
LEDG[2]	Unknown	PIN_E25
LEDG[3]	Unknown	PIN_E24
LEDG[4]	Unknown	PIN_H21
LEDG[5]	Unknown	PIN_G20
LEDG[6]	Unknown	PIN_G22
LEDG[7]	Unknown	PIN_G21
SW[0]	Unknown	PIN_AB28
SW[1]	Unknown	PIN_AC28
SW[2]	Unknown	PIN_AC27
SW[3]	Unknown	PIN_AD27
SW[4]	Unknown	PIN_AB27
SW[5]	Unknown	PIN_AC26
SW[6]	Unknown	PIN_AD26
SW[7]	Unknown	PIN_AB26
SW[8]	Unknown	PIN_AC25
SW[9]	Unknown	PIN_AB25
SW[10]	Unknown	PIN_AC24
SW[11]	Unknown	PIN_AB24
SW[12]	Unknown	PIN_AB23
SW[13]	Unknown	PIN_AA24
SW[14]	Unknown	PIN_AA23
SW[15]	Unknown	PIN_AA22
SW[16]	Unknown	PIN_Y24
SW[17]	Unknown	PIN_Y23

Figure: 10 Pin Planner

The RTL view can be observed as follows.

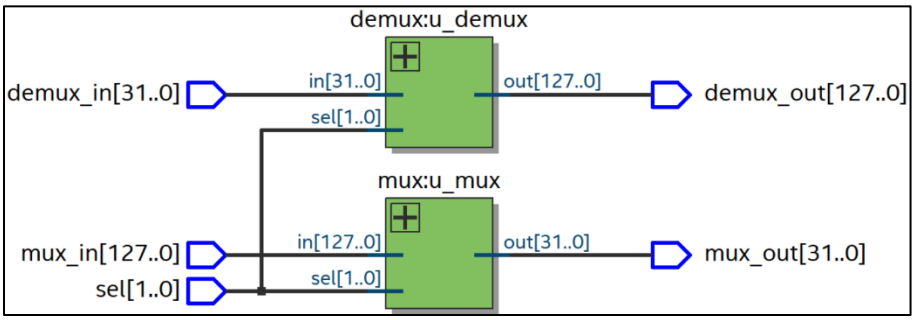


Figure: 11 RTL

FPGA Results

The overall behavior of the ***multiplexer*** is represented by the following image. For debug mux need ***"top_mux_fpga".v*** set as top entity.

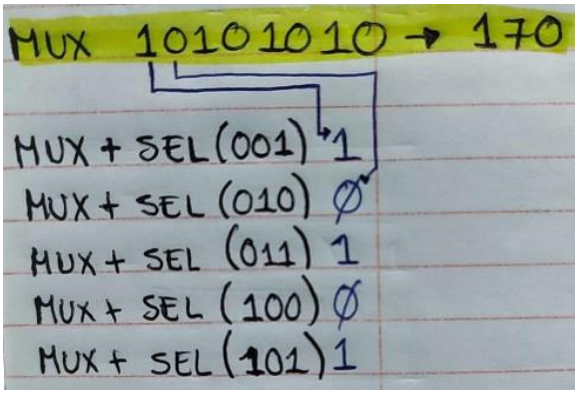
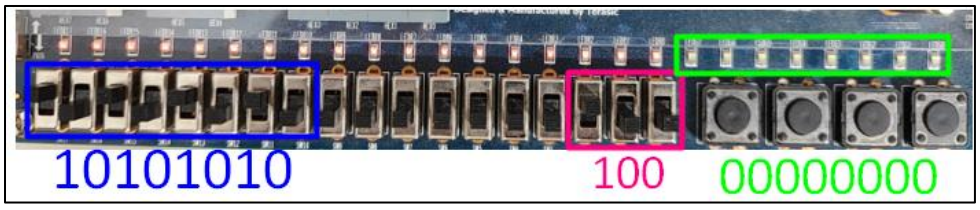
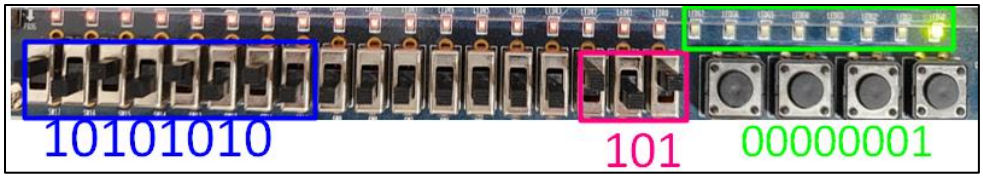


Figure: 12 Mux_Testing

The result obtained for ***mux*** are the next:





The overall behavior of the ***demultiplexer*** is represented by the following image. For debug demux need ***"top_demux_fpga.v.*** set as top entity.

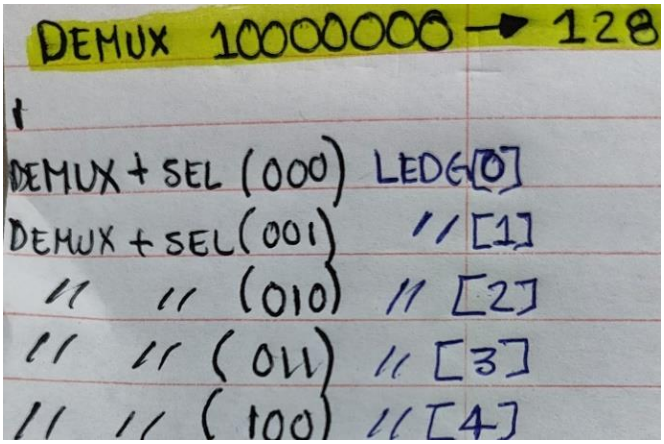
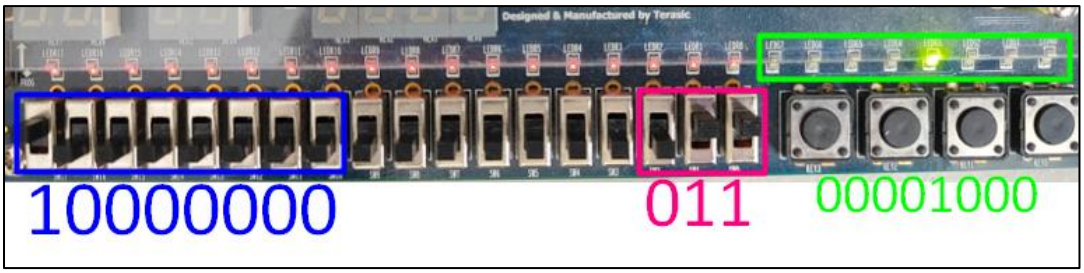
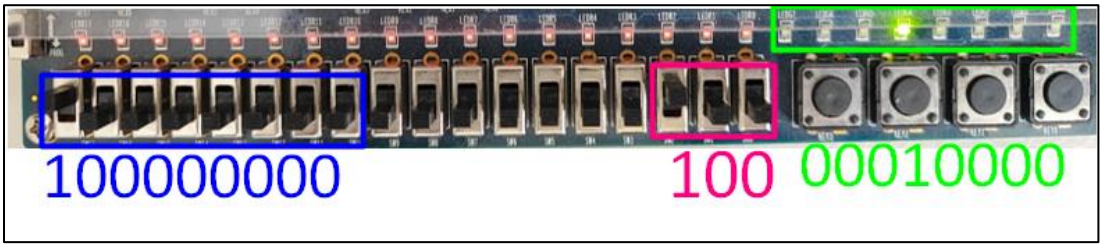


Figure: 13 Demux_Testing

The result obtained for ***demux*** are the next:





GitHub Repository

https://github.com/CeicGitHub/Fundamentals_Of_Digital_Design_FPGA_2/tree/Lab3_Mux_Demux

Conclusion / Issues

Cesar Eduado Inda Ceniceros

Some of the challenges I faced while completing the lab included managing two different module implementations to ensure the correct operation of the multiplexer and demultiplexer. This required me to consider key aspects during programming. Another important point was ensuring that the testbench had a unified top-level module to avoid using multiple files. One of the issues I began encountering was properly assigning values to instantiate the elements. When adding various components in Model Sim, I ran into several errors due to those mechanisms. Personally, I learned a lot from this practice. It became much clearer to me the difference between both components and how to better understand their logical operation from another perspective. By creating the testbench and modifying values within it, I gained a deeper understanding of how it works and its importance in any Verilog project.

Alonso Emmanuel Lopez Macias

I believe there were some confusing situations when trying to understand how to instantiate certain elements in the testbench, as some aspects weren't entirely clear at first. As I progressed through the lab, I was able to realize this and understand it better. I gained a deeper understanding of how the multiplexer and demultiplexer function, and the logic behind them. For example, when creating the top module for the testbench, I noticed how practical it can be to define that element within the same file. However, it's essential to always consider the program's details to avoid errors that could cause issues when testing one module or the other. It was interesting to observe in the simulation how the bit-width and channels behaved, especially when parameterized and adjusted according to the various requirements of the lab.